CRACK DETECTION WITH DEEP LEARNING:
AN EXEMPLARY STUDY OF DATA DESIGN IN ARCHITECTURE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÇAĞLAR FIRAT ÖZGENEL

IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
BUILDING SCIENCE IN ARCHITECTURE

MAY 2018

Approval of the thesis:

CRACK DETECTION WITH DEEP LEARNING: AN EXEMPLARY STUDY OF DATA DESIGN IN ARCHITECTURE

submitted by **ÇAĞLAR FIRAT ÖZGENEL** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Building Science Graduate Program, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar ____________________
Dean, **Graduate School of Natural and Applied Sciences**

Prof. Dr. F. Cana Bilsel ____________________
Head of Department, **Department of Architecture**

Prof. Dr. Arzu Gönenç Sorguç ____________________
Supervisor, **Department of Architecture, METU**

Examining Committee Members:

Assoc. Prof. Dr. Ayşe Tavukçuoğlu ____________________
Department of Architecture, METU

Prof. Dr. Arzu Gönenç Sorguç ____________________
Department of Architecture, METU

Prof. Dr. Birgül Çolakoğlu ____________________
Department of Architecture, İTÜ

Prof. Dr. Mine Özkar Kabakçıoğlu ____________________
Department of Architecture, İTÜ

Assist. Prof. Dr. Elif Sürer ____________________
Modelling and Simulation, METU

Date: 30.05.2018

I hereby declare that all information in this thesis document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Çağlar Fırat ÖZGENEL

Signature:

ABSTRACT

CRACK DETECTION WITH DEEP LEARNING: AN EXEMPLARY STUDY OF DATA DESIGN IN ARCHITECTURE

Özgenel, Çağlar Fırat

Ph.D. in Building Science, Department of Architecture

Supervisor: Prof. Dr. Arzu Gönenç Sorguç

May 2018, 181 pages

Dramatic increase of available data in the last 20 years transformed the role of data in artificial intelligence algorithms for problem solving. Deep learning embodies potentials for both finding novel correlations within data, and improvement in decision making process in its massiveness. Thus, this approach is prominent in processing such massive data by removing the necessity of explicitly determining features relevant to the solution. Reformulation of the problem in terms of determining which data represent the problem and evaluating the results emerge as the primary challenges in deep learning applications.

Within the scope of this thesis, data design term is introduced to describe end to end process of problem solving with deep learning algorithms which is suitable for broad range of applications including problems in architecture. Data design defined as a holistic approach embracing the process from problem (re)formulation to evaluation of the results considering the interrelations of decisions made throughout the process. In this context, data design in architecture is exemplified with the task of crack detection in buildings in order to minimize subjectivity in the course of evaluating the results. For this purpose, the relation between data and deep learning framework, case specific evaluation requirements and strategies for enhancing the performance are inspected through image classification and semantic segmentation applications for crack detection. Concordantly, this study contributes to the literature not only with the introduction and framing of data design but also with the proposal of crack detection specific evaluation metrics for both image classification and segmentation applications and a novel method is proposed employing quad tree and deep learning algorithms in conjunction for semantic segmentation of objects with limited visual features. As a result, data design and respective consequences are discussed in depth and demonstrated regarding the case dependency, decisions taken in the course of implementation and their influences to both process and the results.

Keywords: data design, deep learning, convolutional neural networks, crack detection, semantic segmentation

ÖZ

DERİN ÖĞRENME İLE ÇATLAK TESPİTİ: MİMARLIKTA VERİ TASARIMI ÖRNEK ÇALIŞMASI

Özgenel, Çağlar Fırat

Doktora, Yapı Bilimleri, Mimarlık Bölümü

Tez Yöneticisi: Prof. Dr. Arzu Gönenç Sorguç

Mayıs 2018, 181 sayfa

Son 20 yıldaki ulaşılabilir verilerin hızlı bir şekilde artması, problem çözme için yapay zeka kullanımında verinin rolünü de değiştirmiştir. Derin öğrenme, verinin fazlalığı ile hem verilerde yeni korelasyonlar bulma hem de karar verme görevlerindeki performansı arttırmak için potansiyeller içermektedir. Bu nedenle, bu yaklaşım, çözüm ile ilgili görülen özelliklerin açık bir şekilde belirlenmesi gerekliliğini ortadan kaldırarak, bu tür büyük verilerin işlenmesinde öne çıkmaktadır. Problemi hangi verinin temsil ettiğinin belirlenmesi ile problemin yeniden yapılandırılması ve sonuçların değerlendirilmesi, derin öğrenme uygulamalarında birincil zorluklar olarak ortaya çıkmaktadır.

Bu tez kapsamında, mimarlık problemleri de dahil olmak üzere geniş bir uygulama yelpazesi için uygun olan derin öğrenme algoritmaları ile problem çözme sürecini baştan sona tanımlamak için veri tasarım terimi önerilmiştir. Veri tasarımı, problemin yeniden formüle edilmesinden sonuçların değerlendirilmesine kadar olan ve süreç boyunca alınan kararların karşılıklı ilişkilerini göz önünde bulundurulduğu bütüncül bir yaklaşım olarak tanımlanmıştır. Bu bağlamda, mimarlıkta veri tasarımı, sonuçların değerlendirilmesinde öznelliği en aza indirgemek için binalarda çatlak tespiti ile örneklenmiştir. Bu amaçla, veri tasarımı ve derin öğrenme arasındaki ilişki, duruma özel değerlendirme gereksinimleri ve performansın artırılmasına yönelik stratejiler, çatlak tespiti için görsel sınıflandırması ve semantik bölütleme uygulamaları ile incelenmiştir. Buna paralel olarak, bu tez çalışması, sadece veri tasarımının tanıtılması ve çerçevesinin çizilmesi ile değil, aynı zamanda görüntü sınıflandırması ve bölütleme uygulamaları için çatlak tespitine özel değerlendirme ölçümlerinin önerilmesi ile literatüre katkıda bulunmaktadır ve sınırlı görsel özelliklere sahip nesnelerin semantik bölütlenmesi için dördün ağaç (quad tree) ve derin öğrenme algoritmalarının beraber kullanıldığı yeni bir yöntem önerilmektedir. Sonuç olarak, veri tasarımı ve ilgili sonuçları derinlemesine tartışılmakta ve duruma özel olma, uygulama sürecinde alınan kararlar ve bunların hem sürece hem de sonuçlara olan etkilerine açıklanmıştır.

Anahtar Kelimeler: veri tasarımı, derin öğrenme, evrişimli sinir ağları, çatlak tespiti, semantik bölütleme

in the memory of Cotton Grandma

ACKNOWLEDGEMENTS

I would firstly like to express my gratitude to my supervisor Arzu Gönenç Sorguç not only because of her academic guidance, motivation and criticism in the course of finding the best but also for being a great mentor showing how to be a decent researcher, teacher and team member. I am honored to be her student for a life time and I feel privileged to work with her in numerous projects, courses and research which provided invaluable experience. Without her support and mentoring, this study would not have been possible.

I also want to thank to jury members, Assoc. Prof. Ayşe Tavukçuoğlu, Prof. Dr. Birgül Çolakoğlu, Prof. Dr. Mine Özkar Kabakçıoğlu and Assist. Prof. Elif Sürer for their valuable discussions and comments both during the research and in my thesis defense.

I owe gratitude to my workfellows and members of DDTeam, firstly to my dearest diplococcus Müge Kruşa Yemişcioğlu for being my sister that I never had. Endless discussions, support and her endurance in my grumpy times provided energy, motivation and determination. I would also like to thank Serkan Ülgen and Fatih Küçüksubaşı for their friendship, being supportive throughout the study. I enjoyed and learned a lot from every moment we shared. I also thank to Orkun Sönmez and members of METU Faculty of Architecture.

I want to thank to Serkan Alkan, Barış Yazıcı, Arsev Aydınoğlu, JeanPiere Demir, Gizem Yetiş and Murat Akın for their support and their hospitality.

Lastly, I owe a very important debt to my beloved wife Elif Özgenel for standing with me, her endless love and understanding. She is the hidden hero of this study by constantly motivating and supporting me. I also thank Püskül, Björn, and all little friends for their calming and soothing energy.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGEMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiv

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xix

CHAPTERS

1.INTRODUCTION 1

- 1.1 Problem Statement 2
- 1.2 Hypothesis 7
- 1.3 Objectives and Scope 8
- 1.4 Methodology 9
- 1.5 Significance and Contributions 9

2.LITERATURE SURVEY 11

- 2.1 Traditional Machine Learning vs Deep Learning 12
- 2.2 Crack Detection as a Machine Learning Application in Architecture 17

3.THEORIES AND POSTULATE 33

- 3.1 Pre and Post Deep Learning Phases 35

3.2 Convolutional Neural Network as an Exemplary Deep Learning Method 37

3.2.1 Loss (Error) Function and Gradient Descend 41

3.2.2 Parameter Update Methods 42

3.2.3 Overfitting Problem 44

3.2.4 Convolutional Neural Networks (CNN) Architecture 46

3.2.5 Utilization of CNN in Custom Problems 52

4.CLASSIFICATION OF CRACK IMAGES – COMPARATIVE ANALYSIS OF PRETRAINED CONVOLUTIONAL NEURAL NETWORKS 55

4.1. Parameters effecting performance of CNN’s on crack detection 56

4.1.1. Dataset 56

4.1.2 Number of Epochs for Training 60

4.1.3 Network dependent parameters: Number of convolutional layers and number of learnable parameters 60

4.2 Evaluation Metrics 63

4.3 Results and Discussions 66

5.SEMANTIC SEGMENTATION OF CRACK IMAGES QUADP (A NOVEL QUADTREE INTEGRATED DEEP LEARNING ALGORITHM) 73

5.1. Semantic Segmentation Approaches 74

5.2 Semantic Segmentation of Cracks 77

5.3 QuadP 78

5.3.1 Issues caused by nature of cracks and semantic segmentation task 82

5.3.2 Resolutions for addressing the issues 83

5.4 Performance inspection of QuadP 90

5.4.1 Evaluation Metrics 91

5.4.2 Results and Discussions 92

6.CONCLUSION 99

6.1 General Discussions 99
6.2 Recommendations for Future Work 106
REFERENCES 109

APPENDICES

I. PRETRAINED NETWORK ARCHITECTURES 117
II. CLASSIFICATION TEST RESULTS 151
III. CRACK BASED SEMANTIC SEGMENTATION METRIC RESULTS 167
CURRICULUM VITAE 177

LIST OF TABLES

Table 1. Major characteristics of prominent AI approaches 3

Table 2.List of studies utilizing machine learning for crack classification................ 21

Table 3. Test cases and respective challenges.. 59

Table 4. Number of images in datasets used for training and validation 59

Table 5. Pretrained networks and respective number of convolution layers and learnable parameters.. 63

Table 6. Accuracy vs confidence weighted accuracy on sample test cases 65

Table 7. Highest scoring test and validation CwA scores for pretrained networks ... 66

Table 8. Mean accuracy, mean CwA scores for test cases... 68

Table 9. computational time for 28K training set for 1 epoch 71

Table 10. Issues and corresponding approaches 84

Table 11. Test cases and dataset sizes for semantic segmentation evaluation 91

Table 12. Global semantic segmentation results for 4 test cases 93

Table 13. Class based semantic segmentation results for 4 test cases 94

Table 14. Number of objects and mean orientaion scores for 4 test cases................. 95

Table 15. Research questions, objectives and contributions of the study 101

Table 16. AlexNet Architecture 117

Table 17. VGG16 Architecture 118

Table 18.VGG19 Architecture 119

Table 19. GoogleNet Architecture 120

Table 20.ResNet50 Architecture 123

Table 21. ResNet101 Architecture 127

Table 22. ResNet152 Architecture 135

Table 23. Fully Convolutional Network Architecture 147

Table 24. SegNet Architecture 148

Table 25. Test 1 Classification Accuracy Results 151
Table 26. Test2 Classification Accuracy Results 153
Table 27. Test 3 Classification Accuracy Results 155
Table 28. Test 4 Classification Accuracy Results 157
Table 29. Test 1 Classification confidence weighted Accuracy Results 159
Table 30. Test 2 Classification confidence weighted Accuracy Results 161
Table 31. Test 3 Classification confidence weighted Accuracy Results 163
Table 32. Test 4 Classification confidence weighted Accuracy Results 165
Table 33. Crack based semantic segmentation metric results (per sample) 167

LIST OF FIGURES

Figure 1. Machine Learning Workflow (drawn by author) 13
Figure 2. ML and DL workflow (Gill, 2017) 15
Figure 3. Practical challenges for crack detection, a) the painting as the noise at the background (left), b) shadow shadowing crack and present noise to be misclassification (middle), b) jointing at left presents noise (right) 19
Figure 4. Abstracted workflow for image classification with traditional ML (drawn by author) 20
Figure 5. Black box workflow representation of CNN classification (drawn by author) 20
Figure 6. k-Nearest neighbor classification (drawn by author) 23
Figure 7. Support Vector Machine (drawn by author) 26
Figure 8. Decision Tree (Alpaydin, 2010) 27
Figure 9. Neural Network Structure (drawn by author) 28
Figure 10. Convolutional Neural Network Architecture (Yakopcic, Alom and Taha, 2016) 30
Figure 11.Abstraction of deep learning phases and decision taken at each phase (drawn by author) 34
Figure 12. Simple representation of a typical ANN (drawn by author) 38
Figure 13. ANN node output calculation (Ghanghau, 2017) 39
Figure 14. Backpropagation algorithm (drawn by author) 41
Figure 15. Visualization of gradient descent algorithm with respect to two features. (drawn by author) 42
Figure 16. High convergence and low convergence rates for parameter update (drawn by author) 43

Figure 17. Illustration of underfitting, optimum and overfitting function estimations (drawn by author) 45
Figure 18. Accuracy vs Epoch graph showing overfitting (drawn by author) 46
Figure 19. A simple representation of CNN workflow (Lecun *et al.*, 1998) 47
Figure 20. Visualization of convolution operation on a7x7 input with3x3 kernel (drawn by author) 48
Figure 21. ReLU, tanh(x) and Sigmoid function (drawn by author) 50
Figure 22. Exemplary max pooling operation with spatial extent of 2 and stride of 2 (drawn by author) 50
Figure 24. GoogleNet inception module (Szegedy *et al.*, 2015) 61
Figure 25. ResNet residual block (He *et al.*, 2016) 62
Figure 26. Network performance (top 1 accuracy) vs computational cost (number of operations) for well-known CNN's (Canziani, Paszke and Culurciello, 2016) 62
Figure 27. Mean difference between CwA and accuracy scores (drawn by author) . 67
Figure 28. Mean accuracy and mean CwA vs %10 intervals (drawn by author) 67
Figure 29. FCN32, FCN16, and FCN8 layer architecture (Long, Shelhamer and Darrell, 2014). 74
Figure 30. DeconvNet architecture (Badrinarayanan, Kendall and Cipolla, 2015) ... 75
Figure 31. SegNet architecture (Badrinarayanan, Handa, et al., 2015) 76
Figure 32. Pixel based data storage (raster images), subdivided image patches for CNN and quadtree division result (drawn by author) 79
Figure 33. CNN and quadtree division in QuadP flowchart (drawn by author) 81
Figure 34.Issues caused by the nature of cracks in adaptation of quadtree division to CNN. Object corresponding to the region border (top left), losing object connectedness and jaggedness of object outlines (top right), losing features due to excessive focusing (bottom left and bottom right) (drawn by author) 82
Figure 35. Quadtree and control meta axes decomposition and resulting decomposition by merging two (drawn by author) 84
Figure 36. Two axes system method on an object corresponding to region boundary (drawn by author) 85

Figure 37. Region scores overlaid on crack image (top left), region scores (top right), Segmentation image based on quadtree division (bottom left), Region scores with 0.5 threshold (bottom right) (drawn by author)..87
Figure 38. 3D Reconstruction and linear interpolation of pixel probabilities to obtain probability surface (drawn by author)...88
Figure 39. Probability map obtained by linear interpolation (top left), probability map obtained by 0,5 threshold (top right), semantic segmentation result (bottom left), image segmentation overlaid on original image (bottom right) (drawn by author).............88
Figure 40. QuadP flowchart (drawn by author) 89
Figure 41.Comparison of network outputs versus original image and ground truth. a) Input image, b) manually drawn ground truth, c) QuadP result, d) FCN result, e) SegNet result (drawn by author) 96

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
BF	Boundary F-score
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CwA	Confidence weighted Accuracy
DAG	Directed Acyclic Graph
DL	Deep Learning
FC	Fully Connected
FCN	Fully Convolutional Network
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoT	Internet of Things
IoU	Intersection over Union
K-NN	K-Nearest Neighbor
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
NAG	Nesterov Accelerated Gradient
NB	Naïve Bayes
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RF	Random Forest

SVM	Support Vector Machine
UAV	Unmanned Aerial Vehicle
VOC	Visual Object Classes

CHAPTER 1

INTRODUCTION

> *"Science, as well as technology, will in the near and in the farther future increasingly turn from problems of intensity, substance, and energy, to problems of structure, organization, information, and control."*
>
> *John von Neumann (1949)*

"How human mind works" is one of the oldest questions and dates back to Aristotle. Since then research on "human mind" escalate continuously and especially starting from the 19th century, they attain a new level with the advances in mathematics and psychology. At the beginning of 20th century, research on the problem-solving act of human mind became one of the key subjects in parallel with the question of whether it is possible for machines to think like a human. This idea was first officially manifested in Turing's (1950) efficacious study titled "Computing Machinery and Intelligence". Following this, famous researchers, Herbert Simon, Allen Newell, John McCarthy, Arthur Samuel and Marvin Minsky, gathered under the umbrella term Artificial Intelligence (AI). The introduction of the term AI enabled the encapsulation of several studies on human-machine interaction, human intelligence, and machine intelligence. Human problem-solving ability, which is one of the key features of human intelligence was also conceived as the measure of machine intelligence. Hence, many studies focused on problem-solving processes in terms of finding ways of the redefinition of the problem, specifying the constraints, determining the objectives and the relevant data. While problem-solving process started to find its projection in algorithms and models, what is the processed information (i.e. data) have remained a puzzling question in AI. Turing's studies on data once more brought the importance of the role of data in problem-solving through questions like "what is the data", "how can it be processed", "how can the data be validated/verified", "what does the data mean". Since then, understanding and controlling data has been one of the major research fields of AI.

The history of AI shows fluctuations in parallel with the available technologies and funding, but the studies never ceased. In the last decades, the advancements of data crunching and storing technologies, and correspondingly exponential accumulation of data praise the importance of AI. This appraisal is observed in a number of studies and papers in the field (Shoham *et al.*, 2017). Today, AI infiltrated our daily and professional lives, embedded in devices and software such as smart voice assistants in mobile devices, autonomous driving vehicles or diagnostic tools in medicine.

It should be noted that in this changing role of AI, not only the mathematical basis of algorithms embedded in AI models but the data incorporated within them plays an important role. Hence, today, one of the major tasks in AI studies is to define and utilize the *useful data* in the huge data reservoir.

In this dissertation, the framing of data within the abundance for a given task is referred as ***data design***. The data design term connotes but is not limited to the selection of data. The related consequences in terms of the perception of the task, validation of the AI and the outcome are also an inherent part of data design and are discussed in depth in the following sections of this study.

1.1 Problem Statement

Artificial intelligence, since the very first days, has always been a problem-solving process in machines which is realized through algorithms. The essential characteristic of AI algorithms is to fully or partially model the problem-solving processes and the human mind. This eventually leads to two parallel, but at the same time separate implementation areas; namely narrow AI and broad AI[1]. The major difference between these two study fields lies on the definition of the task; narrow AI is mostly employed in tangible problems which can be considered as problems for which either objectives, constraints or goals can be defined whereas broad AI studies focused on replicating the human mind and skills regardless of any specific task. Most of the contemporary implementations of AI belongs to narrow AI in which definition of the problem and handling the data are essential to the success of AI algorithms. In that sense, there are several approaches, which successfully employed AI algorithms, dating back to 1940's when the AI was first proposed. Among these, expert systems, evolutionary algorithms, and machine learning are the prominent ones that are still in use today. The major characteristics of these approaches are shown in Table 1 indicating the process and data:

Table 1. Major characteristics of prominent AI approaches

	Expert Systems	**Evolutionary Algorithms**	**Machine Learning**
Reflected Information	Field Knowledge	Fitness Function	Implicit field expertise for developing the system structure
Input Data	Input data for a new case	Constraints	Previous occurrences for training, new data representing a case for test
Process	Through If-else clauses	Optimization of generated data	Optimization of system
Data flow	Data are fed forward through a set of rules	Data are generated and optimized with respect to the fitness function	Learns from the data via feedforward and backward mechanisms
Validation	Once the system is constructed no need for validation	Due to hard performance metrics	The accuracy of the input-output match
Use cases	Fields which the knowledge can be modeled with if-else clauses	Fields which the fitness function/objective function can be modeled	Cases which appropiate data are present
Example	Medicine, Law	Well-defined generative problems, such as component optimization	Clustering, decision making, classification tasks

---

[1] Narrow AI is also known as applied or weak AI whereas broad AI is referred to as strong or full AI

As it is shown in Table 1, the way how the problem is declared into AI, such as either by defining constraints, embedding field knowledge or training through previous solutions determines the differences among algorithms. The role of data in all these algorithms has also shown differences since data can be the input like in the expert systems, or it can be self-generated within the algorithm as in evolutionary algorithms. It should also be noted that each of these methods are developed by approaching the problem-solving task from a different perspective and is strong in different problem domains. While expert systems are widely utilized in problem areas where the knowledge can be modeled by rule-based procedures, evolutionary algorithms are prominent in well-defined generative problems such as optimization tasks in engineering and/or architecture, and machine learning is popular in fields in which appropriate amount of data are present. The increase of data, in terms of both quality and quantity, makes machine learning a subject of interest for exploiting the potentials hidden in the mass of data. It is important to pay attention that even though the algorithms mentioned above have been employed for decades, they are still open to improvement and continuously optimized with the emerging technologies and methods and with respect to available data.

Starting from 1980's the exponential growth of data which is usually referred as Big Data, can be considered as a game changer in existing AI strategies. Even though there are numerous definitions of Big Data, one of the most cited definition is made by U.S. National Institute of Standards and Technology (NIST) as:

> **Big Data** consists of extensive datasets—primarily in the characteristics of volume, variety, velocity, and/or variability—that require a scalable architecture for efficient storage, manipulation, and analysis. (NIST Big Data Public Working Group, 2015)

While first three characteristics; namely volume, variety and velocity are self-explanatory, variability (also used as veracity) denotes the uncertainty hence the reliability of data. The preprocessing of raw data is actually a refinement act analogous to a purification process for which irrelevant and/or redundant data are excluded. In that sense, the large amount of Big Data also means a complex data refinement process which becomes an actual challenge of contemporary AI implementations.

Although the term is usually used to define a large amount of data in a positive connotation, it should also be understood that it is mostly unstructured i.e. raw, and needed to be refined in order to bring in compliance with the algorithm in the context of the task. In this respect, the appropriateness and reliability of data play a more important role than the plentitude only. Wisely selected and carefully preprocessed data with limited volume may have more potential for machine learning implementations than huge data streams without the refinement process have in some cases. Hence, it is necessary to point it out that Big Data or any data, which is useful for the task, is important in AI and in this regard, AI implementations should not be confined with Big Data applications.

Despite the concerns presented above in relation with Big Data and its use, all the conventional machine learning algorithms still benefit from the abundance of data as long as the aptitude of data and the problem task is ensured since learning action is achieved through finding the patterns. In this respect, traditional machine learning implementations necessitate data refinement as not being capable of operating on raw data. For that reason, such algorithms are often utilized in conjunction with a feature extraction operation where the user defines the relevant features for the solution of the problem. Deep learning algorithms are differing from traditional ML algorithms as conducting feature extraction autonomously within the algorithm.

Deep learning algorithms have gained popularity as a response to the demand of Big Data processing (e.g. Alsheikh *et al.*, 2016; Wilamowski, Wu and Korniak, 2016; Zhang *et al.*, 2018), improving the machine learning process by allowing to achieve complex data relations which are not necessarily conceived by human mind. In that sense, deep learning algorithms compared with traditional ML approaches necessitates new mindsets and redefinition of the statement of the problem into the machine. In traditional ML approaches, users are required to explicitly define the relevant features of the problem and craft the data accordingly. Contrarily, deep learning algorithms conduct feature extraction phase within the algorithm, forcing researchers to redefine what raw data[2] is and how the data should be used in a task. Consequently, the associative links present in humanly ways of problem-solving become obsolete in deep learning. The user has the opportunity to explore the features which may seem irrelevant for the problem and utilize these unlikely features to either improve the solution and/or make an introspection towards the accustomed way of problem-solving. This new mindset presents a dichotomy for the user; as being a potential for exploring new possibilities and at the same time a challenge with the necessity of breaking the routine.

Contemporary DL implementations succeeded in various fields such as visual perception (Chatfield *et al.*, 2014; Szegedy *et al.*, 2015; Wu, Zhong and Liu, 2017), autonomous control(Chen *et al.*, 2015), and language translations (Collobert and Weston, 2008) and even have shown promising results in generative tasks such as generation of photorealistic images (Goodfellow *et al.*, 2014). Among these study fields, computer vision is one of the leading implementation areas which DL approaches have already proven themselves with high success rates. On the other hand, the user is forced to abandon the accustomed way of problem-solving strategies thus complicating the ability to trace the flow of information and correspondingly the evaluation of the outcome. Thus, most of the studies are focused on very well defined tasks such as object detection which the objects have plentiful features. Implementations on cases with scarce features and tasks requiring subjective interpretation are rather limited. In this sense, problem definition, evaluation, and validation which are implicitly involved in designating the data have great importance.

In this implementation process, determining the *useful* data within the bulk of Big Data is a very fuzzy and a challenging task, which not only requires field expertise but also AI literacy and being capable of devising quantitative mechanisms for validation of the system and the outcome. Today most of the research is conducted employing the accuracy based metrics[3] for validation of the results rather than questioning the

---

[2] In this study, raw data is defined as a multilayered unstructured bulk of data which needs to be elaborated. Elaboration of raw data does not mean solely data selection but instead trying to get the best suitable data for the given task.

[3] For exemplary discussions on shortcomings of accuracy based metrics see (Goertzel, 2015; Powers, 2015)

precision requirement specific to the task, and the relationship between data provided and performance. The subjectivity, which is inherently involved in the selection of data, is accepted as de facto and often attributed to field expertise or intuition. This research also discusses the subjects regarding the validation metrics and the relevance of data selection in deep learning algorithms.

Consequently, this thesis addresses the importance of designating the data in deep learning algorithms and this process of data elaboration is defined as **data design** rather than selection or crafting the data. In this respect, data design in deep learning algorithms is exemplified through crack detection in buildings using visual data. The reason of selecting crack detection as the case study is that the task is challenging for DL algorithms as the visual data have limited features for discrimination of cracks; albeit a straightforward task for humans, easing validation of the outcome and the algorithm without the need of subjective interpretation. Yet, the case provides a basis for scrutinizing the data design in terms of data selection, data and performance relation and evaluation of outcome and algorithm. In addition, potentials of novel algorithms for performance improvements are discussed and exemplified through the case study.

1.2 Hypothesis

The hypothesis of this research can be stated as, the success of AI implementations and raw data processing frameworks together with improvement in the process depends on the data design which requires field expertise related with the problem, AI literacy and reformulation of the problem through the data. These competences are also the core requirements for making improvements in the process.

Data design is scrutinized by means of following research questions:

1. How data design influences output and evaluation of DL algorithm?
2. How can metrics for the evaluation of the results be determined?
3. Is it possible to decide on optimal values for number and quality of data to guide data designers?

4. What is the relationship between data design and the structure of DL framework?

1.3 Objectives and Scope

Architecture has both a challenging and benefitting position in the utilization of deep learning algorithms in accordance with data design. Since architecture faces a broad range of problem from well-defined to wicked with varying complexities and in different scales, architecture poses a challenge for AI and DL frameworks. On the other hand, the abundance and operability on interrelated multidimensional data make architecture a strong candidate for effective utilization of deep learning algorithms. Although the potential application area of deep learning and thus data design is enormous in architecture, the complexity of the problem and the requirement of subjective interpretation is directly reflected to the evaluation of the outcome. Hence, as the subjectivity involved in data design increases, the evaluation of the outcome also becomes a personal task and not possible to quantify. Meanwhile, the deep learning algorithms to be utilized in problems of architecture remains the same in terms of the mathematical basis.

The aim of this research is to reveal the challenges in the deep learning implementations while focusing on the significance of data design in architecture. In this respect, within the scope of this thesis, the research is focused on the working principles of DL and data design process through a tangible task which is crack detection in buildings and topics which require a subjective evaluation of the outcome such as sketching or designing are excluded.

The main objectives of the research are:

- To exemplify and highlight the constituents of data design
- To demonstrate the potentials of deep learning algorithms in problems of architecture
- To explore case specific metrics for the output evaluation and validation of the DL frameworks
- To identify case specific limitations and potentials of DL algorithms

- To explicate the working principles of DL to provide AI literacy
- To demonstrate how DL frameworks can be used in conjunction with other algorithms for precision advancements
- To provide a holistic understanding of data design

1.4 Methodology

In order to achieve a holistic understanding of data design, the discussion is pursued over a controlled case study. The crack detection in buildings by means of visual inspection is a straightforward task for human cognition. Hence the evaluation of the outcome involves minimum subjective interpretation of the data. In that sense, the research is purely quantitative and empirical based on statistical results.

In order to understand the significance of data design, firstly, a comparative study regarding the traditional ML and DL frameworks in relation to the selection, handling and control of data. In this regard, the potentials and challenges of data design is highlighted with respect to the differences and similarities of ML and DL frameworks. The study is then focused on machine learning implementations for crack detection task to establish a benchmark regarding the state of art. Secondly, data design is postulated and explained in detail. Correspondingly, working principles of DL are inspected considering the interaction between DL structure and data design. The study is continued with the in-depth application of crack detection by means of DL approaches and the relationship between framework, data design and performance of the system are inspected. Thereafter, the study is deepened with focusing on pixel-wise predictions and a novel method for crack segmentation is proposed by the utilization of DL algorithms and quadtree algorithms to demonstrate how to utilize DL algorithms in conjunction with existing frameworks to increase the performance and to exploit the potentials of data design.

1.5 Significance and Contributions

Utilization of DL frameworks in custom tasks is a holistic process for which the success is governed with the coherence of provided data, DL algorithm and evaluation methods of the results. Data design comprises this process end to end, and aims to achieve a complete understanding for DL implementations by addressing significant aspects which determines the success of the application. These aspects include but not limited to the subjectivity involved in both data selection and evaluation of the results, the relationship between DL framework and data, and strategies for task specific precision enhancements. In that sense, introduction of data design is the most significant contribution of this study. It is believed that data design approach provides a frame for further studies especially for tasks requiring subjective interpretation.

Furthermore, this thesis contributes to literature by exemplifying case specificity of the DL applications through crack detection in buildings. For that purpose, task specific evaluation metrics are devised and proposed for two classification problems at different scales; namely image and pixel classification. While confidence weighted accuracy metric enables comparison of frameworks with respect to both correctness and confidence of predictions, crack metrics (i.e. number of crack objects and mean orientation) provides a tool for measuring the success of pixel level predictions with respect to the adequacy of the results for post processing.

Moreover, a novel method is proposed with the combination of DL and quadtree algorithms for pixel level predictions of crack regions. In that sense, the proposed method not only sets a precedent for task specific precision enhancement strategies, but also constructs a new mindset for quadtree subdivision operation in conjunction with DL algorithms. Although the method is developed for crack detection task, it is also applicable to objects with visually similar features with cracks such as blood veins or tree branches.

CHAPTER 2

LITERATURE SURVEY

"More data beats clever algorithms, but better data beats more data."

Peter Norvig

The dramatic increase in the available data and advancements in ML and DL algorithms create a huge demand for experts having insight and experience to be able to select and process necessary data for the given task. Today, such experts have different names like data architect, data engineer, data scientist, feature engineer for which their task has broadened with the introduction of Big Data demanding new skill sets which transforms data science as well.

Data science is defined by NIST (2015) as:

> … the extraction of actionable knowledge directly from data through a process of discovery, or hypothesis formulation and hypothesis testing. Data science can be understood as the activities happening in the processing layer of the system architecture, against data stored in the data layer, in order to extract knowledge from the raw data.

In the realm of this new definition, the major role of these experts is to extract, refine and decide on the relevant data regarding the available dataset for the given problem. Yet the accessibility of data is ambiguous and Big Data is not open access to all experts and researchers. All definitions regarding data science and corresponding roles presume that a relevant part of Big Data is provided and does not encapsulate the selection of that part or acquisition/creation of dataset from scratch.

This chapter of the study presents the current literature on machine learning studies in terms of attaining, managing and handling data. For this purpose, differences and similarities between traditional machine learning (ML) and deep learning (DL) in terms of designating, handling and controlling the data are examined. The discussion is elaborated through inspection of machine learning implementations for crack detection in buildings as an exemplary case for architecture and a specific literature on crack detection is present.

2.1 Traditional Machine Learning vs Deep Learning

Several definitions of machine learning are present in the literature. Among these definition, two of them are highly acknowledged describing the main principles of the machine learning idea. Arthur Samuel, who introduced the term in 1959, defines machine learning as *the field of study that gives computers the ability to learn without being explicitly programmed* (Samuel, 1959). A more technical definitive explanation is made by Tom Mitchell. Mitchell (1997) describes machine learning as systems which are optimized with respect to the patterns in training data in order to make a prediction for new query:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

The generic structure of any ML algorithm can be illustrated in Figure 1 regarding Mitchell’s definition.

The experience (E) referred above is the provided data to be trained determining the success or failure of the algorithm in accordance with the given task (T). The level of convergence of the trained data for the expected results is denoted by performance (P) which is implicitly defined in the definition of the task. Definition of task (T), thus the performance (P) plays a more significant role in ML problem-solving. It should be noted that, each training dataset (E) has a different potential for any problem, ill-defined or well-defined. The very same training dataset (E) may change the algorithms' performance. Similarly, when the task and performance definitions are changed, the ML algorithm may yield different outputs, or it may even fail to provide an acceptable solution for the concerned problem.

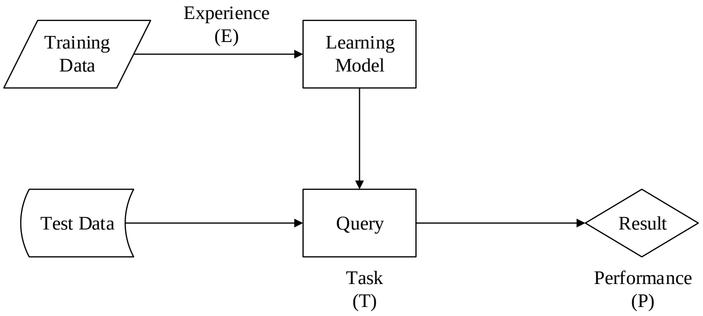


Figure 1. Machine Learning Workflow (drawn by author)

Traditional ML techniques are operated with the predetermined number of input features. Hence, such techniques require a data preprocessing phase where the user extracts the features which is believed to be relevant with the task in other terms user makes an abstraction of the problem. The presumptions of the user regarding the relevance of inputs and the task are reflected to the framework. After data input, data are processed by means of probability calculations and depending on the number of features utilized in the solution. It is possible to trace the information flow in a comprehensible way and to intervene at mid-stages towards optimization. Nevertheless, as the complexity of the problem and number of features included increase, it becomes harder to follow the connections. The output of the framework is strictly dependent on features that users determine prior to the execution of the algorithm. Hence, traditional ML methods reflect how the user approaches the problem. Concordantly, the evaluation of the output is based on the performance criteria defined by the user while selecting the features constituting the task.

One of the most important characteristics of traditional ML techniques is the specification of all the relevant (in the eyes of the user) features to the problem task.

Haque (2007) articulates this problem while elaborating Pask’s conversation theory and relevance to design as:

> … if a designer specifies all parts of a design and hence all behaviors that the constituent parts can conceivably have at the beginning, then the eventual identity and functioning of that design will be limited by what the designer can predict. It is therefore closed to novelty and can only respond to preconceptions that were explicitly or implicitly built into it.

In such cases, ML algorithms conduct a straightforward task, finding the contributions/influence of features to the solution. This approach guarantees the desired type of outcome, at least in terms of revealing a correlation between inputs and outputs. Thus, defining every features of the problem does not necessarily connote a negative meaning especially if the aim is to utilize ML for very well defined tasks and to exploit the computational capacity of the framework.

Contrarily to traditional ML frameworks, DL algorithms omit the preprocessing phase which the relevant features are determined. As a result, the user selects the data which are believed to represent the task. The comparison of ML and DL workflows are illustrated in Figure 2.

As shown in Figure 2, the feature extraction phase is conducted within the framework and optimized throughout the process. The associative links constructed for the problem-solving strategy is broken and the user is forced to focus on the representative power of the data rather than the relevance of features. As the raw data constitute more information than selected features for traditional ML frameworks, the process is often incomprehensible to the user and referred as black box algorithms. Being black box is not caused by the model itself but as a result of the complexity of data relations which is beyond the data crunching capacity of humans. Thus, it is not possible to intervene during the process and extract mid process instances. Yet the outputs at the end of each iteration can be observed and reused as a starting point for a new training session. The output is determined by the provided data and as the algorithm is based on finding the patterns to achieve a generic function resembling the input data, the evaluation is embedded to the data provided.

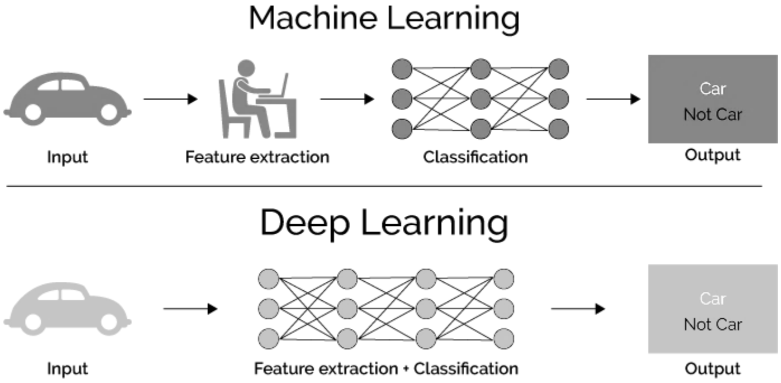


Figure 2. ML and DL workflow (Gill, 2017)

As the user cannot intervene the data process, DL literacy emerges as an essential skill for controlling and collaborating with intelligent machines. Without a profound understanding of the framework which the task is conducted, the user has only the liberty of data selection. In such cases, the success (if the algorithm succeeds) of the outcome is somewhat coincidental and have a capacity of improvement in the event of deeper understanding and deliberate selection of the framework, redefinition of the task and data selection with respect to the working principles of chosen DL algorithm.

The choice of data has the primary importance for the utilization of DL as the data provided directly influence the output and the evaluation criteria are implicitly defined within the dataset provided. There is no ultimate answer on which data is apt for the completion of problem task and each dataset results in different solutions. Hence, choice of data remains a challenging task involving subjective interpretation.

Working principle of DL algorithms is based on finding patterns among the data provided. By definition, with each new dataset obtained as the result of data design, framework obtains a generic function for the representation of the data. It is especially promising for ill-defined tasks which the task can’t be transcribed in a procedural way. In that sense, the output of the framework can be utilized as a tool for broadening the perception of the problems highlighting the importance of certain features seemingly irrelevant to the task or a tool for introspection about intuitive actions taken while manually solving the problems.

In response to this, the evaluation of the outcome is a major challenge. As the performance of the output is not bounded with the performance metrics of the problem task but based on the resemblance to the input data, the subjectivity embedded in the selection of data is pursued to the data evaluation. For well-defined problems which objective evaluation of the output is possible, as in optimization problems, it is possible to quantify the performance of the framework. On the other hand, as the problem becomes ill-defined, the evaluation of the framework becomes a qualitative task based on the preferences of the user. Turing (1949) refers to this problem in his famous quote regarding sonnet writing machines as:

> ... I do not see why it [a computer] should not enter any one of the fields normally covered by the human intellect, and eventually compete on equal terms. I do not think you can even draw the line about sonnets, though the comparison is perhaps a little bit unfair because a sonnet written by a machine will be better appreciated by another machine.

In such cases, DL algorithms can be utilized as personalized co-workers for ill-defined problems which we can only communicate by means of data.

Architecture, which embodies tasks with varying scales, complexities, and subjective interpretation requirements, provides a broad field of study for DL implementation. It is not always possible to explicitly determine the features of the solution or structuring the problem solving act in procedural way in problems of architecture. Thus, being able to train learning models without the necessity of explicitly defining the constituents of the problem poses a great potential for architecture. On the other hand, it is not possible to cover the diversity of architectural problems due to three major challenges namely; lack of publicly available data for all problems, lack of adequate hardware to process the data (even if data were publicly available) and most importantly the objective measurement of success especially for tasks requiring subjective interpretation.

For these reasons, the discussions and elaboration are pursued by means of focusing on a well-defined task; namely crack detection from visual data, in order to concretize data design with quantifiable and measurable results. Crack detection is a straightforward task for human perception and therefore the algorithm and the results can be evaluated without subjective interpretation. Still, it poses a challenge for the DL algorithms due to the lack of discriminative features defining cracks. Hence, DL algorithms, as well as the computer vision algorithms, are prone to confuse shadows, foreign objects, and drawings with cracks. For the case implementation, CNN framework is utilized as its capability to process raw data while conserving spatial relations which is crucial in tasks regarding architecture. As CNN's use similar building blocks and mode of operation regardless of the task, crack detection is utilized to scrutinize a complicated subject in simple terms. For this purpose, traditional ML and DL (i.e. CNN) implementations for crack detection are examined in Chapter 2.

2.2 Crack Detection as a Machine Learning Application in Architecture

Advancements in machine learning influence fields which are seeking autonomous conduction of tasks. Increasing capabilities of machine learning frameworks hold promise for high speed, high accuracy and high precision and autonomous predictions. Recent studies and applications prove that computer algorithms can be more powerful in decision-making tasks without being objected to subjective bias or in other terms human error as in manual conduction of the task (Lecun, Bengio and Hinton, 2015; Schmidhuber, 2015).

Building inspection is one of the fields which requires the minimization of error due to the importance of the possible implications. In this respect, machine learning algorithms have been utilized in the building inspection with the aim of increasing precision and accuracy.

The process of building inspection can be broken down with abstraction in three phases with increasing complexity; as detection of defect (predicting existence), analysis of defect (predicting metrics – numeric prediction) and inferring from results. Cracks, as being one of the most information bearing sign of structural failure, can also be detected, analyzed and evaluated by means of machine learning algorithms. Even though crack detection can be conducted with respect to various data such as infrared thermography and/or acoustics based responses, visual data captured with still camera is still the most feasible in terms of the equipment required and majority of the inspections of buildings are conducted with respect to visual data due to ease of data acquisition.

Despite the fact that visual inspection is the most favorable approach, processing of visual data is not a straightforward task and have challenges caused by the nature of the task. Particularly, crack detection in the built environment is a challenging task for any computer vision and machine learning method. The success of machine learning algorithms is originated by the ability of framework to learn and use the discriminative features defining classes of objects. In the case of crack detection, the number of such discriminative features are limited. Hence, the features defining cracks can be easily confused with any object having irregular and jagged edges having high contrast with surface texture. The practical challenges caused by the nature of the cracks can be listed as below and illustrated in Figure 3:

a) Orientation and spatial positioning of cracks are unpredictable. Hence, it is not possible to make an inference from directionality of high contrast regions to classify cracks.

b) Discriminative crack features are easily confused with noise in the background texture, foreign objects and/or irregularities in application such as exposure of jointing

c) Inhomogeneous illumination of the surface causes occlusion of crack segments endangering the conservation of crack continuity (Zou *et al.*, 2012; Wang *et al.*, 2017)

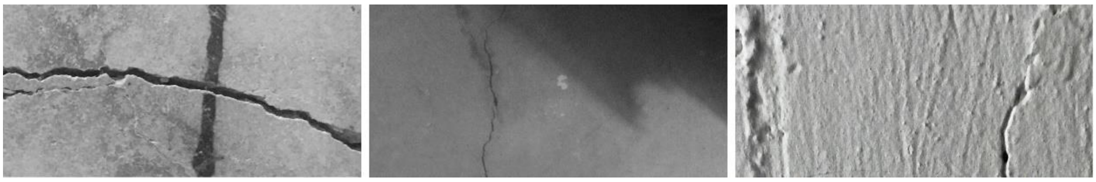


Figure 3. Practical challenges for crack detection, a) the painting as the noise at the background (left), b) shadow shadowing crack and present noise to be misclassification (middle), b) jointing at left presents noise (right)

Due to the challenges mentioned above, an adaptive framework which is able to discriminate cracks and any other objects and/or surface texture, in other words, a framework which is not susceptible to noise in data is the ultimate goal for autonomous crack detection. The task of crack detection can be evaluated as a classification problem whether it is conducted for predicting the existence of cracks in an image or pixel-wise prediction of crack regions. Machine learning field offers numerous approaches to classification problem with increasing accuracy due to the continuous research and improvements in the field.

By nature, classification with respect to multiple features is a multidimensional problem. When visual data captured by means of cameras is considered, raw data contains vast amount of data represented as pixels. As the computational cost and memory constraints are directly related to the number of features, it is not feasible to treat each pixel as a feature in the course of utilization of machine learning classifiers (Koch *et al.*, 2015). Hence, several steps are followed to reduce the dimensionality of the data. These steps are defined and can be abstracted as image segmentation and feature extraction in common practice. As a result, the workflow for image classification can be illustrated as having three stages; namely image segmentation, feature extraction, and feature classification (Sinha and Fieguth, 2006; Gonzalez, Woods and Eddins, 2009; Wu, Liu and He, 2015).

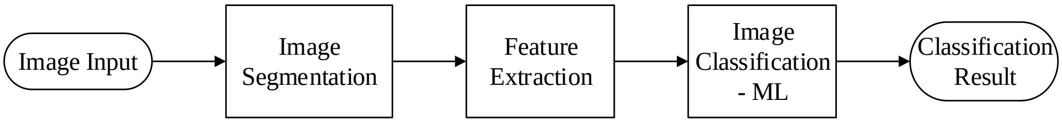


Figure 4. Abstracted workflow for image classification with traditional ML (drawn by author)

Among these steps, image segmentation is used as a tool to be able to conduct feature extraction and commonly perceived as a part of feature extraction step. Image segmentation and feature extraction steps are often conducted with manual or semi-autonomous/adaptive methods which require user input. Even though there are studies aiming conduction of image segmentation and feature extraction, the separate conduction of the steps results in accumulated error of the classification. As a result, either the overall process involves manual decisions or the process is conducted in series of autonomous algorithms which the error is cumulatively increased. In such cases, obtained framework is often applicable to certain cases and lose performance with varying conditions. Studies, which embraces crack detection workflow as series of operation, are focusing on selection and finding the optimum combination of methods.

Convolutional neural networks (CNN's) pose a different approach to fragmental conduction of classification task. CNN's conduct the feature extraction and classification tasks within single framework without the need of image segmentation preprocessing. Hence, the case-specific bias caused by manual decisions in feature extraction step and cumulative accumulation of error is avoided in CNN's. Abstracted black box representation workflow of CNN's is illustrated in Figure 5.

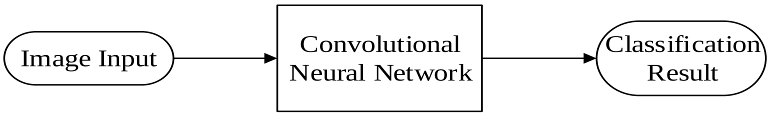


Figure 5. Black box workflow representation of CNN classification (drawn by author)

There are several studies in the literature proving the applicability of CNN's to crack detection in built environment task (e.g. Gopalakrishnan *et al.*, 2017; Liu *et al.*, 2017). Furthermore, CNN applications are extendable to perform localization and semantic segmentation of images. Localization by means of CNN's aims to draw a bounding box to the classified objects while semantic segmentation of images deals with labeling of pixels with respect to the object class they belong and is different from image segmentation mentioned in classification workflow. Image segmentation is used as a preprocessing tool for reducing the dimensionality of input data and does not aim to conduct pixel-wise labeling as the classification is conducted in later steps of the workflow. Hence, while semantic segmentation is a goal to achieve, image segmentation refers to initial step required before feature extraction and utilization of machine learning classifiers.

A comprehensive but not complete list of studies which uses machine learning methods in correspondence with segmentation and feature extraction methods, and studies utilizing CNN are shared in Table 2.

Table 2.List of studies utilizing machine learning for crack classification

Reference	**Application Area**	**Segmentation & Feature Extraction**	**Classification**
(Liu *et al.*, 2002)	Tunnels	- Discriminant analysis method - Threshold	- Support vector machine
(Sinha and Fieguth, 2006)	Pipelines	- Statistical feature extraction	- Neural network - K-Nearest neighbors
(Kabir, Rivard and Ballivy, 2008)	Bridges	- Wavelet transform - Statistical feature extraction	- Neural network
(Yang and Su, 2008)	Sewer pipes	- Wavelet transform - Co-occurrence matrix	- Neural networks - Support vector machine
(Moon and Kim, 2011)	Generic concrete	- Median subtraction - Gaussian low pass filter - Threshold - Morphological operations	- Neural network

Table 2. List of studies utilizing machine learning for crack classification (cont'd)

Reference	Application Area	Segmentation & Feature Extraction	- Classification
(Zhang *et al.*, 2014)	Subway tunnels	- Average smoothing - Morphological operations - Threshold - Statistical feature extraction	- Neural network - Support vector machine - K-nearest neighbors
(Lattanzi and Miller, 2014)	Generic concrete	- Wavelet transform - Canny edge detector - Statistical feature extraction	- Naïve Bayes - Decision trees - K-nearest neighbors
(Wu, Liu and He, 2015)	Sewer pipes	- Wavelet transform - Contourlet transform - Maximum response filter bank	- Decision trees
(Gibert, Patel and Chellappa, 2015)	Railways	Convolutional Neural Network	
(Schmugge *et al.*, 2015)	Power Plants	- Morphological operations - Linelet-based segmentation (Naïve Bayes)	- Neural network
(Santur, Karaköse and Akın, 2016)	Railways	- Principal component analysis - Singular value decomposition - Histogram mean	- Decision trees
(Zhang *et al.*, 2016)	Roads	Convolutional Neural Network	
(Ersoz, Pekcan and Teke, 2017)	Pavements	- Threshold - Median filtering - Morphological operations	- Support vector machine
(Li *et al.*, 2017)	Bridges	- Region-based active contour - Statistical feature extraction	- Support vector machine
(Gopalakrishnan *et al.*, 2017)	Pavements	Convolutional Neural Network	
(Liu *et al.*, 2017)	Buildings	Convolutional Neural Network	
(Cha, Choi and Büyüköztürk, 2017)	Generic concrete	Convolutional Neural Network	
(Eisenbach, Stricker and Debes, 2017)	Roads	Convolutional Neural Network	
(Wang *et al.*, 2017)	Pavements	Convolutional Neural Network	
(Pauly *et al.*, 2017)	Pavements	Convolutional Neural Network	
(Küçüksubaşı, 2017)	Buildings	Convolutional Neural Network	

The aim of the list is not to present a corpus of machine learning applications but to provide an insight on how broad the possible combinations of methods are. As the scope of this study is focused on machine learning algorithms for crack detection, the studies mentioned in Table 2 are categorized with respect to the machine learning methods employed and exemplary studies are briefly discussed. The image segmentation and feature extraction methods are discussed in relation to the machine learning methods which they are utilized in conjunction with. Case-specific machine learning applications and hybrid methods are discarded for categorization and such methods are grouped under the parenting approach. The only exception among the specialized methods is convolutional neural networks as CNN combines image segmentation and feature extraction steps in itself; hence operated differently from neural networks.

K-Nearest Neighbors(Cover and Hart, 1967)*:* As one the simplest methods in machine learning, K-NN frameworks are trained with only feeding labeled input data. When an unlabeled datum is classified with respect to the training data, neighboring *k* data points, which is a user-determined number, are located and the majority of the data points determines the class of the unlabeled test data.

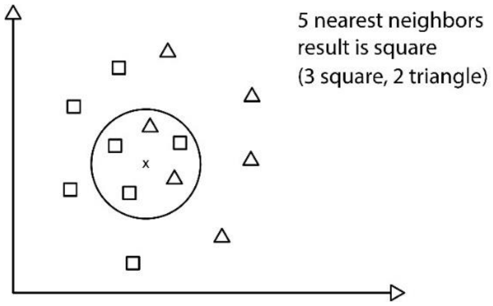


Figure 6. k-Nearest neighbor classification (drawn by author)

The studies utilizing the k-NN method for crack classification generally used for benchmarking and comparative analysis of the performance of studies proposal rather than employing k-NN as the primary approach.

Sinha *et al.* (2006) used the neuro-fuzzy network, which is essentially neural network, for classification of buried pipe defect. For comparative analysis of proposed method, the k-NN method is used. For feature extraction, study utilized statistical feature extraction methods and determined a series of features to calculate from segmented image such as area of regions, number of objects, minor and major axis lengths. They proposed a neuro-fuzzy network as a modified version of artificial neural network operated in conjunction with a neuro-fuzzy classifier and projection neural network. In their comparative analysis for classification accuracy of k-NN, fuzzy k-NN and neural network variations, k-NN scored %81 accuracy for crack/hole classification while the maximum accuracy obtained with other methods is %94.1. (Sinha and Fieguth, 2006)

Similarly, Zhang, *et al.* (2014) applied k-NN, support vector machine and two neural network based methods as radial basis function neural network and extreme learning machine for classification of cracks in subway tunnels. The study utilized several methods subsequently such as average smoothing, morphological operations such as black top hat transformation, applying threshold for image segmentation and used statistical methods for feature extraction based on standard deviation from shape distance histogram. Even though test accuracies of utilized classifiers are similar, extreme learning machine scored highest with %91,6 followed by support vector machine, radial basis function neural network and k-NN classifiers. K-NN scored %88,7 for this experiment (Zhang *et al.*, 2014).

Naïve Bayes: Naïve Bayes algorithm is operated similarly to k-NN algorithm and checks for the vicinity of new data for neighboring training data. However, Naïve Bayes algorithm also considers a priori probability depending on the previous observations. Class prediction is made by taking the maximum probability with respect to the number of data points within the predetermined vicinity and its multiplication with prior probability.

Lattanzi *et al.* (2014) focused on the performance of image segmentation and feature extraction steps. They used k-means approach which constructs clusters with respect to the mean value of clusters and compared with Canny edge detector and Haar wavelet transformation. The segmentation results are tested with Naive Bayes, decision trees, k-NN and the overall classification performance is inspected with Bayesian networks, decision trees, neural networks and k-NN. Naïve Bayes classifier results in conjunction with Canny edge operator (Canny, 1986) performed better than other classifier – segmentation couples for image variance test. On the other hand, for the segmentation effectiveness, Naïve Bayes results are lower than decision tree and k-NN scores (Lattanzi and Miller, 2014).

Schumugge *et al.* (2015) utilized a kernel based filtering for segmentation named as linelets. After applying line filter, the line segments are joined by training a Naïve Bayes classifier to check the line segments within the vicinity of a threshold to obtain continuity. Morphological operations are applied to the image in parallel for comparison of two methods. Both segmentation results are tested with neural networks and anomaly classifiers. It is reported that linelet segmentation, which utilizes Naïve Bayes classifier, performed %38 better than morphology based segmentation (Schmugge *et al.*, 2015). Study of Schumugge *et al.* is significant as the study aims to conduct image segmentation with machine learning with the utilization of filters and Naïve Bayes classifiers.

Support Vector Machine: Support vector machines are proposed by Vapnik and Lerner in 1963(Vapnik and Lerner, 1963). The main idea behind the support vector machine is to classify data by constructing hyperplanes dividing the classes. For this purpose, the distance of data to hyperplane is used as a measure to be maximized.

A modified version of SVM's is kernel SVM (Boser, Guyon and Vapnik, 1992), which operates as the same way as linear SVM's the data is processed with a kernel function to be able to handle nonlinear classification. It should be noted that SVM operates in feature space and requires feature extraction to process images.

Liu *et al.* preprocessed the image with two thresholds method which one threshold is determined with Discriminate Analysis Method and second threshold is applied with respect to the calculated intensity gradient vector. In addition, a balancing operation is conducted with respect to the gravity centers of sub-images. After extracting the features, support vector machine is used for the classification (Liu *et al.*, 2002).

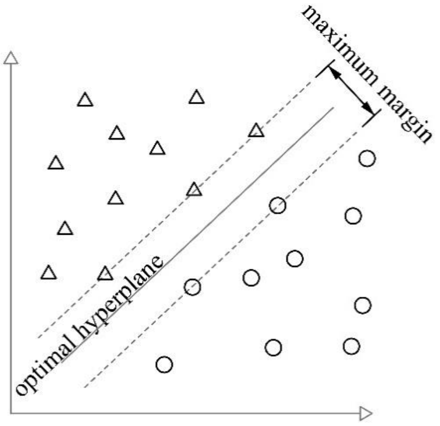


Figure 7. Support Vector Machine (drawn by author)

Yang *et al.* (2008) used wavelet transformation and co-occurrence matrix for image segmentation and feature extraction steps. The extracted features are trained with support vector machine and two neural network based classifiers as back-propagation neural network and radial basis network. Support vector machine with radial kernel is observed to perform better than the backpropagating neural network, radial basis network and support vector machine with polynomial kernel (Yang and Su, 2008).

Ersöz *et al.* (2017) are focused on crack detection from images captured by autonomous aerial vehicles. Image segmentation is conducted manually by setting threshold for each training image and geometric properties of image regions are calculated to extract features. The features are then used for classification by means of SVM (Ersoz, Pekcan and Teke, 2017). Even though reported accuracy is 97%, manual threshold approach for image segmentation introduces a bias towards the dataset. On the other hand, the studies mentioned above proves the performance of SVM if the features are determined carefully.

Decision Trees: Decision tree classifiers are operated to construct a tree-like structure having the most influential feature as the main node and other features are represented as branches of the trees to conclude with resulting classes. The entropy of the system is checked and the hierarchy of the features are constructed to minimize the entropy. Decision trees are mainly decision support algorithms.

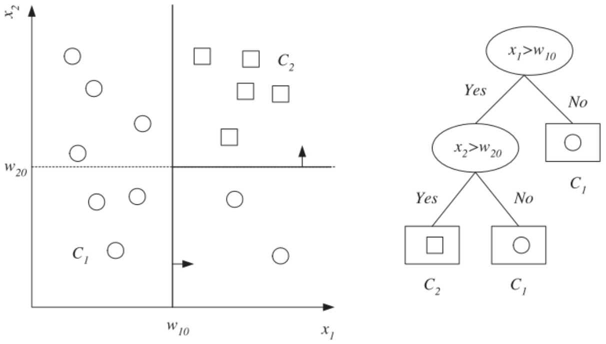


Figure 8. Decision Tree (Alpaydin, 2010)

Wu *et al.* (2015) use contourlet transformation which is a wavelet transformation technique after transferring input image to grayscale image. After contourlet transformation, the image is divided into high-pass image and low pass image to be processed with directional filters. By this way, it is aimed to obtain smoother edge detection. The preprocessed images are used for feature extraction by means of co-occurrence matrix and Tamura features and extracted features are classified with several ensemble methods as AdaBoost, Random Forest, Rotation Forest and RotBoost which are all based on decision tree idea. The mentioned ensemble methods are operated to boost the performance of a classifier by means of supporting with several other classifiers. The differentiation between methods are caused by the construction of decision trees and have the results are brought together. The results of ensemble methods are compared with neural network based methods (multilayer perceptron, radial basis function neural network) and support vector machine. The best result is obtained with RotBoost method which is trained with statistical feature vectors with %89,96 accuracy (Wu, Liu and He, 2015).

Santur *et al.* (2016) also used Random Forest method which is an ensemble method based on decision trees. Even though the study is focusing on railways, image classification of visual data to detect defects is similar to crack detection task in buildings. Several methods are used for dimensionality reduction such as principal component analysis, kernel principal component analysis, singular value decomposition and histogram matching separately to observe the influence of feature extraction step on the accuracy. The resulting features are used for training random forest algorithm. As a result the combination of principal component analysis and histogram matching yields 85% accuracy (Santur, Karaköse and Akın, 2016).

Neural Networks: Among other classifiers, neural networks are relatively the most complex method. Neural networks are multilayered structures which each layer contains several nodes. At each node, a simple linear function is operated. Neural networks learn from training data by adjusting the weight/influence of the functions taking place in nodes and conducts classification. Misclassified samples are used to calculate the error and propagated back to revise the influence of the nodes. Each node at the layer is connected to the nodes at the following layer. Hence a fully connected structure is established to construct a relation between each feature.

Kabir *et al.* (2008) used neural networks for classification of damages in bridge infrastructures. For image segmentation and feature extraction, wavelet transformation and texture analysis are conducted respectively and extracted features are used for training artificial neural network. They compared the accuracy results with respect to the input type, i.e. grayscale, color and infrared images and obtained between %70.6 to %84.1 accuracy for three different datasets (Kabir, Rivard and Ballivy, 2008).

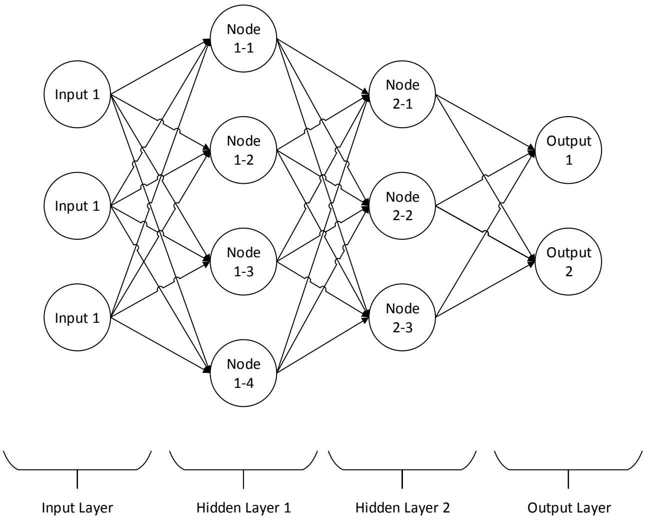


Figure 9. Neural Network Structure (drawn by author)

Similarly Moon *et al.* (2011) used series of preprocessing methods such as median subtraction, Gaussian low pass filter, threshold for segmentation and morphological operations for feature extraction. The resulting features are used for training artificial neural network. The proposed workflow achieved %90.25 accuracy as an average of two test cases (Moon and Kim, 2011).

The studies present above embrace multi-step process for conduction of crack classification with respect to visual data. As the training data depending on the case studies, selected methods and determined parameters vary, it is not possible to draw a conclusion regarding which method is more suitable for which step. As a controlled study on the comparison of classifiers, Enterazi-Maleki *et al.* (2009) constructed 29 different datasets with increasing number continuous and discrete variables and investigated the performance of well-known machine learning classifiers (Entezari-Maleki, Rezaei and Minaei-Bidgoli, 2009). The samples belonging to variables (features) are randomly generated to avoid any bias caused by the data selection. It is observed for datasets with high number of samples, decision trees, k-NN and SVM methods are highly efficient. While Naïve Bayes classifier performs worse, neural networks are not included in the comparison.

In a similar study, Huang *et al.* (2003) compare decision trees, SVM and Naïve Bayes classifiers with respect to accuracy and area under curve metrics. The study reported SVM, Naïve Bayes, and C4.4 (decision tree method) scores are comparable while C4.5, which is another method for constructing decision trees, is outperformed by other classifiers (Huang, Lu and Ling, 2003). The dataset utilized in this study is randomly generated by different from the dataset used in the study of Enterazi-Maleki *et al.* and reflects real-world problems.

Even though such studies provide an insight about the optimal conditions which the classifiers perform, case dependent variables such as selection of features, previous operations enabling reducing dimensionality are dominant on the resulting performance of the overall framework. A comprehensive study on crack detection which performs a grid search to determine which factors are more dominant and under which conditions the machine learning algorithms perform better is not present to the best of authors knowledge.

Convolutional Neural Networks: Convolutional neural networks are operated differently from other machine learning classifiers as the framework contains feature extraction step in itself and does not require image segmentation as a preprocessing step. CNN is based on neural network architecture. Yet, while neural networks are bounded with high computational cost due to the fully connected structure of neural networks, CNN's don't have to be fully connected and contains multiple layers in its architecture. While leading layers operate for extracting features, final layers of the CNN architecture is composed just like neural networks and operates as a classifier.

CNN's can be evaluated as a kernel-based neural network architecture as at each layer of the network, the image is convolved with a series of filters. The output of the convolutions is treated like the nodes of the basic neural networks and their weights are adjusted similarly. There are two approaches for working with CNN's as training a network from scratch, which refers to the determination of the number of layers and weights are randomly initialized and transfer learning which refers to the utilization of a pretrained network with respect to another dataset which may be completely different from the task in question.

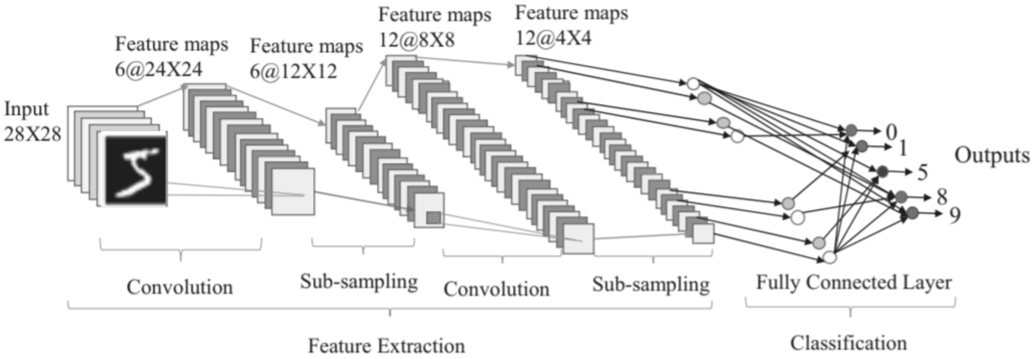


Figure 10. Convolutional Neural Network Architecture (Yakopcic, Alom and Taha, 2016)

Studies of Zhang *et al.* (2016), Eisenbach *et al.* (2017), Pauly *et al.* (2017) and Cha *et. al.* (2017) can be given as examples of studies which constructs and trains CNN's from scratch. The networks constructed in these studies have relatively limited number of layers with respect to the network configurations which utilizes pretrained networks. Zhang *et al.* (2016) constructed a CNN with 6 convolution layers and trained the network with 600K and tested with 200K road images while Eisenbach *et al.* (2017) utilized a network with 11 convolution layers and used datasets with sizes of 4,9M for training and 1,2M for testing. In the study of Eisenbach *et al.*, (2017) the network proposed by Zhang *et al.* (2016) is compared with their result and 11 layered CNN is reported to perform slightly better than the network with 6 convolution layers. Pauly *et al.* (2017) also investigates the relation between the number of convolution layers and the networks' performance by comparing accuracies of 6 layered and 7 layered networks. CNN containing 7 convolution layers is reported to achieve %91,3 accuracy performing better than the network containing 6 convolution layers (Pauly *et al.*, 2017).

The study conducted by Cha, *et al.* (2017) used a framework with 4 convolutional layers for concrete crack detection in building cases. The study investigates the relationship between the influence of training dataset size and the network is trained with several dataset sizes varying from 2K to 40K images. It is advised to utilize more than 10K images for training based on validation scores (Cha, Choi and Büyüköztürk, 2017).

Gopalakrishnan *et al.* (2017) focused on transfer learning approach by utilizing pretrained networks and fine tuning for crack detection task. VGG-16, a highly acknowledged pretrained CNN, is utilized for distress detection in pavements. The network is trained with 760 images and tested with 212 images in total. For comparative analysis, classifier layers of CNN are replaced with random forest, extremely randomized trees, SVM and logistic regression classifiers. The study reported %90 accuracy for the original version of the pretrained network as the highest scoring option (Gopalakrishnan *et al.*, 2017).

Similarly, Küçüksubaşı (2017) focused on crack classification by means of transfer learning for the path planning of an autonomous UAV building inspection system. Author utilized Inception v3 network which is fine-tuned with 1040 image samples and tested on a dataset containing 64K images. The reported accuracy of Inception v3 network is %97 which is a considerable achievement proving the applicability of pretrained networks on crack classification task (Küçüksubaşı, 2017). The size of training dataset required for achieving high accuracies for pretrained networks is remarkably less than the required size for training from scratch.

Even though the number of studies utilizing CNN is gaining momentum, most of the studies focus on impact of one or two variables governing the performance of the algorithms. A comprehensive study focusing on multi-variables such as the influence of dataset size, the number of layers and the number of learnable parameters in a holistic approach, is not present to the best of authors knowledge.

Moreover, the relationship between data and the performance of the frameworks employed is investigated with respect to the quantity of the data (i.e. how many images are employed) rather than the quality (i.e. at which extent does data represent the generic case). In that sense, the selection of data remains as an elusive act often bounded with the expertise and/or intuition of the user who provides data to the framework.

This thesis focuses on the improvement of the performance of CNN algorithms not only in terms of finding optimal values for the variables and parameters of CNN algorithms but also by means of inspecting the influence of data selection to achieve a holistic understanding of visual data processing with DL algorithms.

CHAPTER 3

THEORIES AND POSTULATE

> *"In deep learning, the algorithms we use now are versions of the algorithms we were developing in the 1980s, 1990s. People were very optimistic about them, but it turns out they didn't work too well. Now we know the reason is they didn't work too well is that we didn't have powerful enough computers, we didn't have enough data sets to train them."*
>
> *Geoffrey Hinton (2016)*

In this thesis, a new term "data design" is introduced to describe the data handling process from a broad perspective starting from the redefinition of the task as a problem of learning from data, to the evaluation of the result for the given task. In that sense, data design is not solely determination/selection of data, but also includes reformulation of the problem, evaluation of frameworks and results, and utilizing or devising appropiate metrics for the task. Despite the crucial role of data design in the success of traditional machine learning (ML) and deep learning (DL) approaches, it is not considered as an overall process explicitly and it is mostly present implicitly in the algorithms without any further assessment. However, holistic understanding of the overall process is crucial for the success of the frameworks. In that sense, data design necessitates expertise, experience and literacy in relation with the given task and DL.

Development of an apt DL framework for any task requires holistic understanding of not only available data and technology, but also the entire process including redefinition of the problem accordingly. Concordantly, data design, like all design problems, is context-sensitive and it is not possible to provide generic strategies fitting for all kinds of tasks. Hence, establishing the link between task and the framework is essential in DL implementations.

Any DL implementations can be abstracted as a three phase process, namely; pre DL phase, which the user selects, preprocesses, and provides the data; DL phase, which the framework is trained with respect to the data provided, and post DL phase which the results and thus the performance of the framework is evaluated. The conformity between the decisions taken at each phase and the task plays a determining role in the performance of the framework. Three phases, decisions and how are they interrelated in these phases are shown in Figure 11.

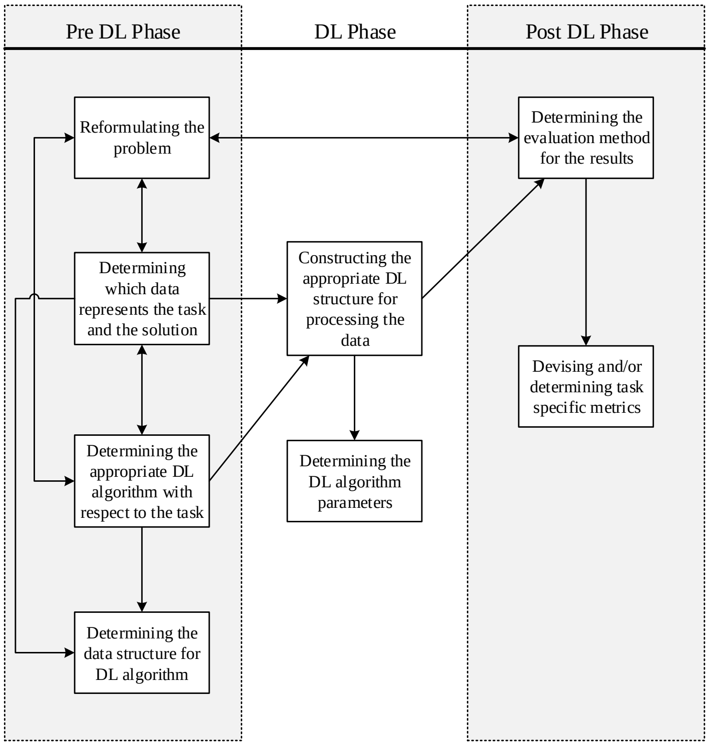

Pre DL Phase
DL Phase
Post DL Phase
Reformulating the problem
Determining which data represents the task and the solution
Determining the appropriate DL algorithm with respect to the task
Determining the data structure for DL algorithm
Constructing the appropriate DL structure for processing the data
Determining the DL algorithm parameters
Determining the evaluation method for the results
Devising and/or determining task specific metrics

Figure 11.Abstraction of deep learning phases and decision taken at each phase (drawn by author)

Data design as proposed in this study, embraces all of the three phases, yet this study puts emphasis on pre and post DL phases which the adaptation of DL algorithms for a specific task takes place. In pre and post DL phases user actively makes decisions on how to reflect task as a DL problem. Although constructing DL structure and determining the related parameters are dependent on the data selected, DL phase is more context free with respect to pre and post DL phases. In this regard, firstly pre and post deep learning phases will be examined to inspect the case dependency of the actions and decisions in the scope of data design. Then, convolutional neural networks are investigated as an exemplary framework of DL applications.

3.1 Pre and Post Deep Learning Phases

Deep learning algorithms together with possibility of using raw data change preprocessing of training data. In this context, the major difference between traditional ML algorithms and DL algorithms relies on the necessity of feature extraction prior to training. In traditional ML algorithms, it is possible to directly reflect the features which are believed to be relevant to the task with traditional ML algorithms. Contrarily, DL implementations operate with raw data and determines the relevant features within the algorithm. Hence, the users are obliged to formulate the task not by determining the features but by means of designating the relevant data. In the course of reformulation of the task and designing the data, subjective assessment of relevant data is necessitated, regardless of the complexity of the task. Reformulation of the problem requires revisiting the question of which data represents the case rather than which properties (features) of the case contributes to the solution. In that sense, accustomed way of problem analysis in terms of differentiating constituents of problem becomes obsolete.

On the other hand, elimination of feature extraction process emerges as a liberty for tasks requiring subjective interpretation and creativity such as arts. In such cases, users are able to customize frameworks regarding their personal taste by designating data solely with respect to their preferences. The capability of personalization for tasks involving subjectivity makes data design and working with raw data appealing.

Regardless of the complexity and subjectivity involved in the reformulation of the problem, the role of data design is crucial; prior experiences, knowledge on the subject even intuition determines the performance of the framework.

One of the most important tasks in pre DL phase is the determination of the framework and establishing the compatibility of data accordingly. There are several DL algorithms developed to respond various needs. For instance, recurrent neural networks are more suitable for task comprising time data, whereas convolutional neural networks are more appropriate for tasks requiring conservation of spatial relations. While data representing the task impose selection of a compatible algorithm, computational limitations of the hardware and technical limitations of the selected algorithm constraints the data input. Hence, the selected data are required to be preprocessed in order to be operable with the DL framework. It should be noted that the preprocessing operation referred herein is not a feature extraction operation but instead transforming the data without altering the information embodied. Thus, the preprocessing operation can be regarded as an act of craftsmanship requiring comprehension of framework limitations and handling the data accordingly without soiling. In this context, determining the relevant data for the solution of the task is not only a problem of representation but also a matter of usability. Hence, pre DL phase is composed of series of actions aiming translation of a specific task to machine. These actions are intrinsically interacting actions and are required to be handled integrally.

Reformulation of the problem and data provided to DL framework correspondingly have direct effect on the results of the given task by implicitly defining the evaluation. Case specific nature of implementations necessitate case specific evaluations. Subjectivity involved in the pre DL phase is reflected to the post DL phase as a matter of determining case specific evaluation methods. In this respect, defining the desired outcome and devising relevant metrics for measuring the performance of the DL framework is an inevitable task.

As discussed in Chapter 2, evaluating the results for straightforward problems, which the subjective interpretation has minimal contribution in data selection and problem definition, is also a straightforward task. On the other hand, as the subjectivity increases in pre DL phase, the evaluation of the results is bounded with the personal preferences of the user providing the data. For example, while it is possible to assess the success of DL implementation for face detection task with accuracy based metrics, the success of a music composing DL implementation can ideally be evaluated by the user who selects the data for training the DL algorithm.

Within the scope of Chapter 3, DL phase is scrutinized through convolutional neural network as an example of deep learning algorithms. By this way, it is aimed to investigate the decisions specified in Figure 11 and reveal how the framework is effected by pre and post DL phases and influence the results.

3.2 Convolutional Neural Network as an Exemplary Deep Learning Method

Convolutional neural network (CNN) is a widely used method for the analysis of visual data in machine learning. Although the majority of the applications are based on visual data, CNN’s have proven themselves in processing volumetric data, in other words spatial data in three dimensional space. For this purpose, CNN’s have the most potential for application to problems in architecture and therefore selected as the framework to be employed as an exemplary deep learning algorithm for implementations in architecture.

CNN studies are hugely inspired by visual cognition studies in 1950’s. The studies of Hubel and Wiesel (1959) showed that receptive fields of neurons in visual cortex are sensitive to some specific visual stimuli. It is observed that different parts of the visual cortex of cats are stimulated in the presence of varying illumination patterns. It is concluded that the visual cortex is composed of columnar structures which are sensitive not to every perceived dot in the visual field but to various shapes such as edges at different orientations (Hubel and Wiesel, 1959). This columnar architecture works together to perceive objects. This idea forms the basis of CNN's.

As CNN is a method of machine learning, the basic procedure of implementation is teaching the system numerous data (training) and then make predictions on new data with respect to the learned patterns among training data which is used for teaching the system (testing). In the case of CNN's, the data is either images or any data represented in image form, i.e. 3D matrix with height, width, and channels.

CNN's are based on artificial neural network architecture (ANN) and employs the same approach of ANN's while training and making predictions. It is essential to grasp the working principles of ANN's to construct a fully integrated understanding of CNN's.

The artificial neural network is a machine learning classifier which user feeds data to the system. The system is trained with the provided data to capture patterns and differentiating features to finally determine different classes of outputs. the terminology and methods explained in this section are limited to the terminology and methods utilized in CNN applications even though ANN's have larger application area.

ANN's are constructed with a biological analogy. With reference to neurons firing at different rates in the presence of diverse events, so-called neurons in ANN's have also different values with different input data. These neurons transmit the values to form neural pathways to obtain desired output values.

A simple representation of a typical ANN system is shown below:

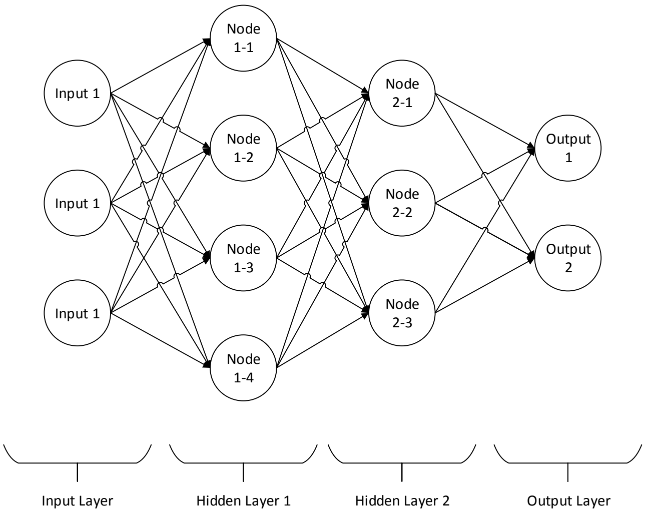


Figure 12. Simple representation of a typical ANN (drawn by author)

As can be seen from Figure 12, there are three main distinct regions as an input layer, hidden layer(s) and an output layer. Hidden layer(s) is the region where nodes (neurons or perceptron) resides and they have simple mathematical functions to map input data to output data.

For each of the nodes, the function is represented as:

$$\mathrm{y} = \mathrm{wx} + \mathrm{b} \tag{1}$$

Where w is the weight and b is the bias of the input. The bias term in this function does not define the bias of the overall system but instead defines a shift in solution space while weight parameter defines the slope of a linear function.

In the presence of multiple hidden layers, the nodes of following layers take inputs from the layer before them and are sequentially operated. The function of the neural network in total can be defined as:

$$g = f_L \circ \ldots \circ f_1 \tag{2}$$

where each subscript represents a layer and *f* represents the function of a node. As can be seen, the general function of the neural network is a nested function of each node and have more representational power even though the constituents of the general function are linear functions. The output of a node is a function of the input connections' weight, bias of the node followed by an activation function which maps the output between a definite range. Most of the activation functions map output value to 0-1 range. The calculation of the output of a single node is illustrate as below:

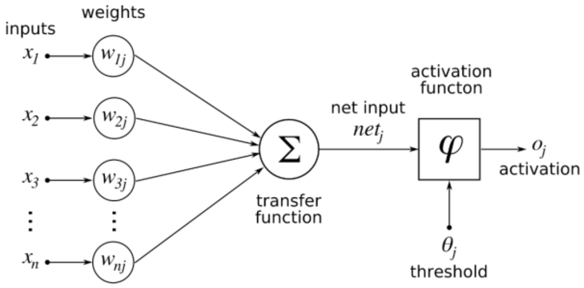


Figure 13. ANN node output calculation (Ghanghau, 2017)

As a result, the output of a node can be generalized with the equation:

$$x_{out} = f(\sum_i x_i w_i + b) \tag{3}$$

Where $f(x)$ is the activation function, b is the bias of the node, w_i is the weight of the connection, x_i is the input.

Once the outputs of the network are calculated with respect to the input value, weights of the connections and bias parameters for each node; the first phase of learning is completed for one iteration. This stage is named as forward pass where the input data is passed forward through the network.

In most of the cases, first iteration will score poorly with high rates of mismatch between output and desired classes. As the ultimate goal is to train the network to obtain desired outputs, the weight and bias parameters must be optimized to correctly map input data to desired output. For this purpose, a method named as backpropagation is employed. Backpropagation algorithm consists four stages for the optimization of weight and bias parameters of nodes and can be characterized with (1) the forward pass, (2) calculation of loss which is the calculation of discrepancy between the output and desired outcome, (3) backpropagation which traces the error backwards step by step to determine the contribution of bias and weight parameters to the error and (4) parameter update to minimize loss. This process is iterated until system provides desired outcomes, in other terms when the loss is minimized.

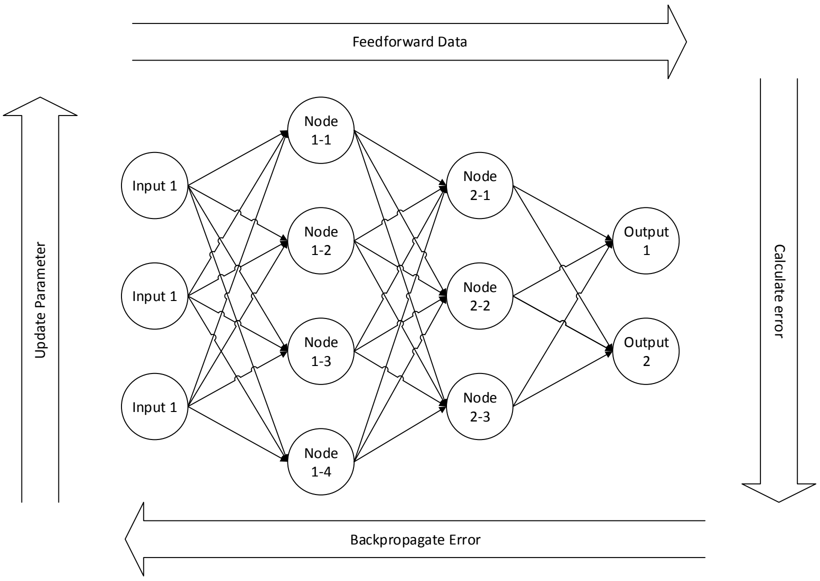


Figure 14. Backpropagation algorithm (drawn by author)

3.2.1 Loss (Error) Function and Gradient Descend

The loss function determines how different the output data and desired output are. The most generic loss function, which is also known as *mean squared error* can be defined as follows:

$$C(w,b) \equiv \frac{1}{2n}\sum_x ||y(x) - a||^2 \tag{4}$$

where *C* is the loss (cost or objective) function *n* is the number of inputs, *y(x)* is the calculated output with respect to input *x* and *a* is the expected output. Both *C(w,b)* and *y(x)* are dependent on weight and bias parameters.. The aim of the neural network is to incrementally decrease the loss value and converge to minimum.

After one forward pass, the loss function is calculated to be minimized by means of gradient descent algorithm. The aim of the gradient descent algorithm is to find global minimum for loss function and for this purpose gradient vector of loss function is calculated with respect to weight and bias variables included in the network. 2-dimensional visualization of gradient descent algorithm is shown in Figure 15.

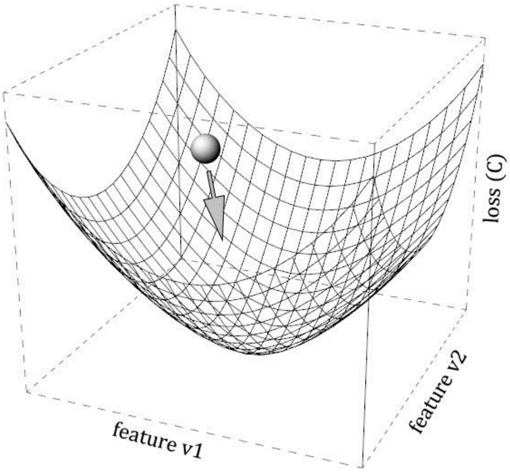


Figure 15. Visualization of gradient descent algorithm with respect to two features. (drawn by author)

As there are multi-layers and multi-nodes in the network, the problem is actually a multidimensional problem. The gradient vector of loss function can be generalized as:

$$\nabla C \equiv (\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m})^T \tag{5}$$

as *C* is the loss function and *v* represents the weight and bias parameters. The slope of the gradient vector is calculated by taking derivative of loss function with respect to the concerned parameter. Once the slope of the gradient vector is determined, it is possible to propagate in that direction to ensure loss function takes a smaller value in the next iteration. This procedure is called backward pass and is the determination of the contribution of the weight of layers to the loss function. After this step, the parameters are updated with respect to gradient vector.

3.2.2 Parameter Update Methods

There are several update methods in the literature. The fundamentals of parameter updating are shared to provide an understanding regarding the process.

The simplest form of update function which is also known as the *vanilla update* is as follows:

$$v \rightarrow v' = v - \eta \nabla_{\theta} C(\theta) \tag{6}$$

where η is learning rate, which is a user-defined parameter, can be defined as the step size at each iteration for updating parameter *v*. Vanilla update can be perceived as taking a step with fixed length towards the direction which the loss decreases. With high learning rates, there is a risk of making big steps to miss global minimum and with low learning rates reaching the minimum can take high computational time.

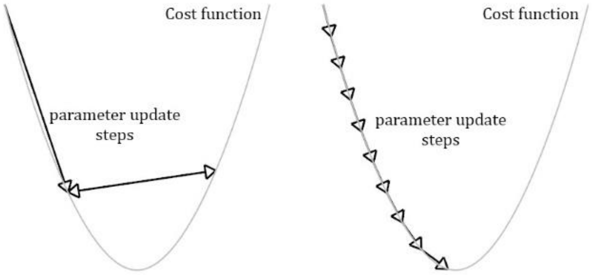


Figure 16. High convergence and low convergence rates for parameter update (drawn by author)

As can be seen from Figure 16, when step size/learning rate is too high, it is possible to oscillate between values and not being able to reach minimum loss value. On the other hand, when the step size/learning rate is too small, it requires many iterations to converge to minima resulting in increased computation time. Vanilla update disregards the direction of previous updates which poses a problem when training samples have considerable diversity which results in slow convergence rates. Hence, the performance of networks employing vanilla update is highly dependent on the choice of user and wise-choice for learning rate requires expertise in the implementation of such models. In order to overcome the shortcomings of vanilla update method, more adaptive update rules are developed. The simplest version of adaptive update rule is momentum update.

Momentum update utilizes momentum parameter for taking the history of previous updates into account. Hence, consistency between iterations is established. The generic formula of parameter update with momentum is as follows:

$$v_{t+1} = v_t - \mu v_{t-1} - \eta \nabla_v C(v) \tag{7}$$

where μ is momentum and v_{t-1} is the previous parameter update.

Several more momentum update based parameter update methods exist such as *Nesterov Momentum*, or Nesterov Accelerated Gradient(NAG)(Nesterov, 1983). Even though momentum update based methods provide responsiveness at different levels, the learning rate remains fixed among parameters. When a vast amount of parameters is considered, adaptive learning rates are desired to have more diversity among layers and parameters to converge generic results.

Methods such as *Adagrad* (Duchi, Hazan and Singer, 2011), *Adadelta* (Zeiler, 2012), *Adam* (Kingma and Ba, 2014), Adamax (Kingma and Ba, 2014), and Nadam (Dozat, 2016) aims to provide more responsiveness throughout to training and among different parameters. The main advantage of methods having adaptive learning rates is that there is no need for manually tuning the learning rate with respect to the problem. Instead, the update rule calculates what the learning rate should be for each parameter and step.

It should be noted that the choice of method for parameter update is highly dependent on the problem to be solved, the size and variance of datasets. In addition, utilization of vanilla update with wisely guessed variables may provide better results than more advanced methods with poorly selected variables. Nonetheless, in terms of computational time, adaptive methods are proven to be faster than non-adaptive methods (Ruder, 2016).

3.2.3 Overfitting Problem

Like all statistical models, the ANN’s have a risk of overfitting. Overfitting can be described as fitting undesirably well to a particular set of data and possibly failing to obtain a generic function for new data to be predicted. In case of neural network training, if the network learns all features specific to the data used during the training, the solution may not be accurate for new data to be inspected. In Figure 17, underfitting, optimum and overfitting function estimations are illustrated.

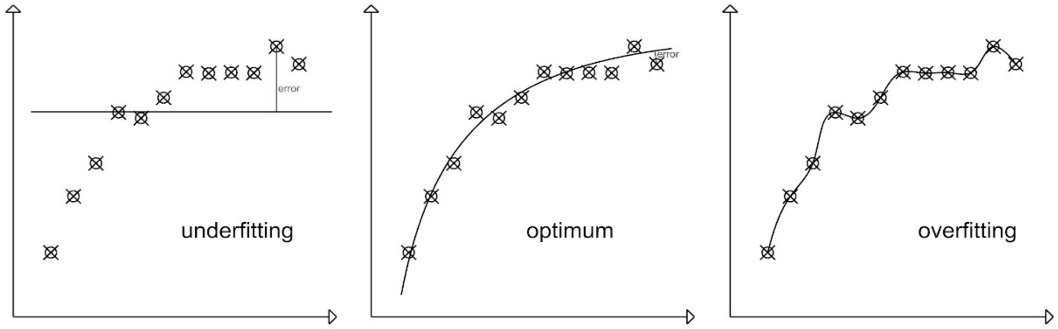


Figure 17. Illustration of underfitting, optimum and overfitting function estimations (drawn by author)

Obviously, the error rates in overfitting situation are smaller than the optimum situation for training data. However, when the data distribution is inspected, it is more likely that new data will not reside on the function estimate in overfitting condition and the optimum case will have lower rates of error and more representational power in the presence of new data. In order to overcome the risk of overfitting, there are several approaches utilized both while preparing the data and constructing the network. One of the most commonly used approaches chosen in the process of preparing training data is to divide the dataset into partitions such that major part of the data is used for training while the rest or a minor part is used for calculation of loss function. This minor part is called validation set. Another minor part of the dataset can be separated optionally or new dataset representing the real case can be constructed for cross-validation. The graphs which plot accuracy versus iteration or epoch (in other terms training cycles, which all training data are inspected by the network for once), shows whether overfitting occurs or not during the training process. In Figure 18, an exemplary graph showing accuracy scores over number of epochs for overfitting concerns. Ideally, validation accuracy follows training accuracy at very similar rates.

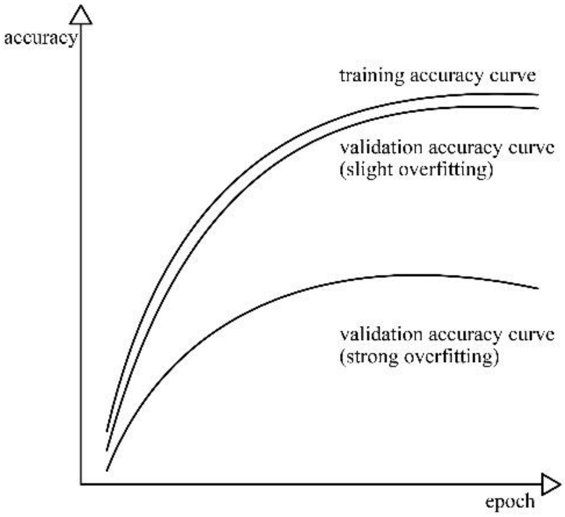


Figure 18. Accuracy vs Epoch graph showing overfitting (drawn by author)

Decision made at the pre DL phase regarding the selection of the framework and the data are directly reflected in accuracy vs epoch or error vs epoch graphics. If the complexity of the data and the framework are not incompatible, the corresponding graphics may reveal underfitting or overfitting. In such cases, data and framework preferences must be revisited to establish compatibility.

3.2.4 Convolutional Neural Networks (CNN) Architecture

CNN's operate with the same principle with ANNs in terms of having hidden layers which weight and bias factors are assigned and optimized throughout the training, having stages of forward pass, calculation of loss, backward pass and parameter update. In contrast, CNN layers do not need to be fully connected and have specialized hidden layers for specific tasks. In addition, in case of CNN’s the inputs are represented as 3D arrays or tensors having height, width and feature channels (depth) while ANN’s are commonly utilized with limited number of inputs in feature space. CNN applications are mainly focused on but not limited to image-based analysis and processing.

A typical CNN consists 4 major layer types as convolution layer, ReLU (Rectified Linear Unit) layer, pooling layer and fully-connected layer and these layers are called building blocks of CNN's. A simple representation of CNN workflow is presented below.

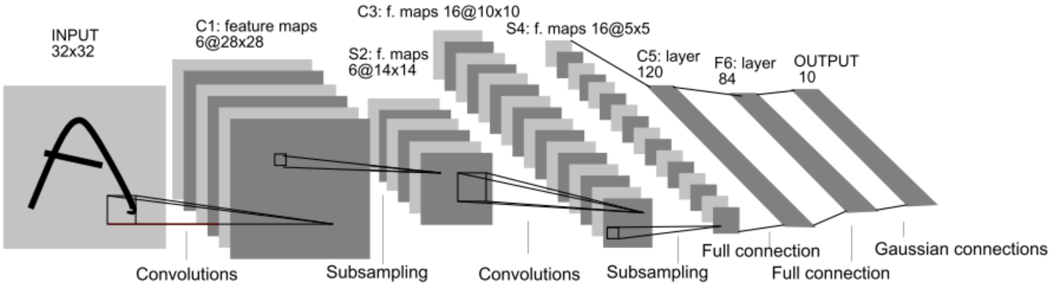


Figure 19. A simple representation of CNN workflow (Lecun *et al.*, 1998)

As in ANN, CNN’s may also have multiple layers of each type of building blocks and generally have a layer pattern as follows:

$$INPUT \rightarrow \big[[CONV \rightarrow RELU] * N \rightarrow POOL\big] * M \rightarrow [FC \rightarrow RELU] * K \rightarrow FC$$

(Karpathy, 2018)

Where N, M and K denotes number of repetitions of the building block groups which they are multiplied with. N is bigger than 0 and usually is also bigger than 3; M is bigger or equal to 0 and K is bigger than 0 and usually is also bigger than 3.

It should be noted that the spatial dimensions of data gradually decrease due to convolution and pooling operations finally resulting in a probability for each possible prediction classes. As the size reduction is strictly bounded with the mathematics behind these operations, input and output sizes determine the number of operations and parameters governing these operations.

Convolution: The very first and the most important operation which forms the basis of CNN’s, is convolution operation. In CNN’s, images are represented as numerical values and for each pixel in the image, corresponding row and column cell has a value denoting its color or brightness. The matrices representing the images can be either 2D or may have more dimensions with respect to the color mode.

Convolution operation is the multiplication of image pixel data with a convolution operator by taking dot product. Convolution operators, which are also known as kernels, filters or feature detectors, are also matrices having smaller dimensions and are multiplied by iterating from left to right and top to bottom.

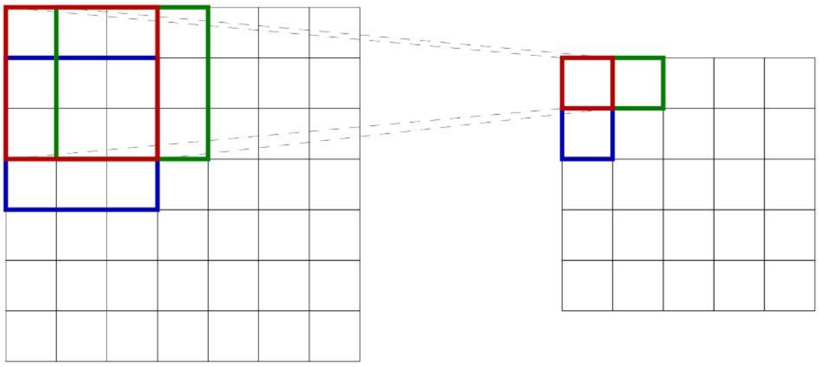


Figure 20. Visualization of convolution operation on a7x7 input with3x3 kernel (drawn by author)

Single convolution operations are commonly used in computer vision with the utilization of several convolution operators for different purposes such as edge detection, blurring, sharpening etc. In CNN's, multiple convolution operators are initialized and given weight and bias values resulting in 3D or 4D tensors with dimensions of width and height of the image (*W*, *H*), optionally color channel of the image (*C*) and the number of filters (*D-depth*) used in convolution operation. These operations are named as feature extraction. In CNN applications multiple convolution operators are used in a single layer to obtain different features of the images. For example, the system workflow shown in Figure 19 consists 3 different convolution operations in the first layer. The output of the convolution operation is called *Feature Map, Convolved Feature* or *Activation Map.* Convolution operation has two major hyperparameters[4] apart from spatial parameters as stride and zero-padding. Stride is the number of pixels to shift after multiplication of filter with the corresponding window (receptive field) and zero-padding is the number of pixels to pad the image in order to include borders of the image or to acquire an output with desired dimensions for next operation.

As data are propagated through the network, extracted features of previous input are fed to the next convolution operation. As a result, network initially is able to detect edges and blobs, followed by primitive shapes and continue with more so-called abstract features representing spatial relations between object components and details in the image. For instance, for face detection task, more abstract features correspond to nose and eyes and how they come together to form a face.

One of the important implementation detail is named as parameter sharing which dramatically reduces the number of parameters to compute. In convolutional layers, the weight is shared among the height and width dimension of the input. As a result, the number of weights is reduced by the input width and height dimension with the assumption of spatial locations of input share the same weight and bias parameters. This implementation detail enables CNN to work on big image inputs and expand the network in terms of the number of layers and filters without exceeding the computational limitations.

Introduction of Non-Linearity: The output of convolution operation may consist negative values for image brightness data caused by matrix multiplication. Rectified Linear Unit (ReLU) function conserves positive values while replacing zero values with negative pixel brightness data. Other functions serving the same purpose such as *tanh* or *sigmoid* functions can also be used with respect to the application. Introduction of non-linearity operation uses non-linear functions to represent real-world data. In addition, these functions also contribute to the reduction of computational time.

---

[4] User defined parameters

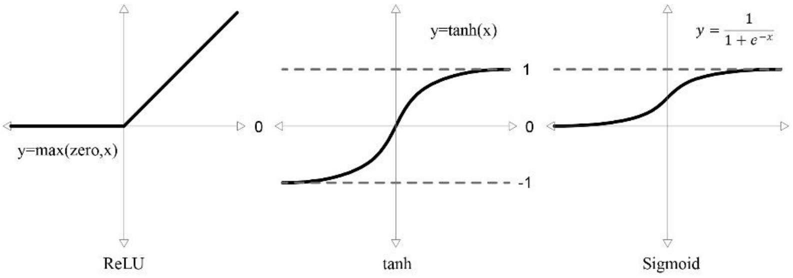


Figure 21. ReLU, tanh(x) and Sigmoid function (drawn by author)

Pooling(Subsampling): Pooling operation is used to reduce the spatial dimensions of the input while conserving the important information. For this purpose, maximum, average or sum of an area which is defined by the window value is taken. For example, for a 3x3 window value, 9 pixel values are taken and an output value is obtained by taking average, sum or maximum of these 9 values. Then next 3x3 neighborhood is selected by striding the window on the input image. This step is especially important as pooling operation reduces data size while conserving the important information; hence increases computational performance. Also, the network becomes invariant to small distortions and transformation.

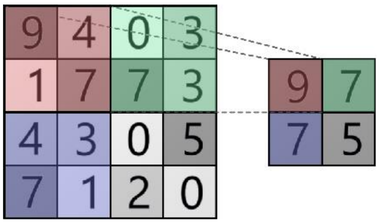


Figure 22. Exemplary max pooling operation with spatial extent of 2 and stride of 2 (drawn by author)

Fully Connected Layers: Fully connected layers are the part where the system reduces the output of the preceding layers to the number of classes which the system gives a prediction score of. Fully connected layers are the last block of layers before prediction so network combines all the information gathered to make an integrated prediction.

For this purpose, fully connected part of the CNN’s are operated as ANN as a classifier. Hence, the number of parameters (weights) dramatically increases with respect to convolutional layers.

Treating the output | Softmax Classifier: Once the output of the last fully connected layer size is reduced to number of classes to predict from, either loss is calculated (during training) to backpropagate error and optimize weights or a classification function is fit (during the test) to output best matching class and class score. For different classification problems such as binary classification, multi-class classification, different loss functions are implemented. However, all of these functions can be simplified to the loss function defined in the previous section. For testing or prediction purposes utilization of softmax classifier is the most common application as it is differentiable and the sum of probabilities is equal to 1 which yields a percentage distribution among classes. Softmax function is defined as follows:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{8}$$

Where z is the class scores and k is the number of classes. The negative logarithm of softmax function is taken to obtain cross entropy loss which is to be minimized in the course of training. Cross entropy function is shown as below:

$$L_j = -\log \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{9}$$

It should be noted other classifier mentioned in Chapter 2 can be utilized instead of softmax classifier. However, the method of choice does not affect the CNN architecture.

Apart from essential building blocks of CNN’s, there are several other blocks widely utilized in CNN implementations. The absence of such blocks does not prevent CNN’s from operating but blocks such as batch normalization or dropout layers increase the performance of CNN’s in many cases and contribute to avoidance of overfitting.

Dropout layers freeze a portion of network nodes and respective input and output connections to these nodes. In this way, the operation effectively eliminates the possibility of any node or feature to dominate which have possibility to cause overfitting. *Batch normalization layer* normalizes the input of the following layer with zero means. This provides standardization of data through the network enabling fast learning without overfitting.

3.2.5 Utilization of CNN in Custom Problems

It is possible to either construct the CNN architecture from scratch by fine-tuning each and every parameter such as the number of layers, number of convolution operations, determination of convolution operators. The construction of CNN architecture is often defined as black art and even though there are certain patterns for layer formations and rule of thumb advices, the best performing CNN architecture for a custom problem is highly dependent on the constituents of the problem i.e. size and variety of dataset, the nature of classification task. In addition, fine-tuning every parameter of the CNN from scratch requires high computational cost and time. Apart from computational concerns, the size and variety of the dataset also determine the performance of CNN. For training a network from scratch, thousands of images per class is required in order to avoid overfitting even if all the methods mentioned above regarding overfitting are employed.

Alternatively, several CNN's which are trained on different datasets are available for fine-tuning. The process of fine-tuning some or all parameters of a pretrained network is called "*transfer learning*". Such pretrained networks are usually trained on public data provided for annual challenges such as Imagenet Large Scale Visual Recognition Challenge (ILSVRC) provided by Imagenet (Stanford Vision Lab, 2018) which the dataset has 1,2 million images from 1000 classes or Pascal Visual Object Classes Challenge hosted by the University of Oxford[5] (Oxford Robotics Institute, 2018) which the dataset has approximately 20000 images for training and testing from 20 classes. As can be seen from the number of classes and number of images, provided datasets are trained for generic purposes and general use. The classes include both animate and inanimate objects.

---

[5] Pascal VOC Challenge lastly took place in 2012. However, the database provided is still widely used for training networks.

However, the variety provided in such datasets does not guarantee that the pretrained networks trained in these datasets will perform well in custom task and is mostly dependent on the similarity of custom problem dataset and dataset used in pretrained network. If the datasets are dissimilar, and custom dataset contains enough number of samples (~500 images per class for transfer learning) for each class, then pretrained network can be fine-tuned for the custom problem. The ability to fine-tuned pretrained networks for custom tasks increases the applicability of CNN’s in everyday tasks without the requirement of supercomputers and high computational time for training. In addition, owing to the transfer learning researchers don’t need to struggle for network configuration process for utilization of CNN is custom tasks.

Information on fundamentals of how CNN’s work shared so far is based on generic applications. However, the network performance is highly dependent on data design, i.e. size and variance of the dataset, and application case together with the configuration of the network utilized. As mentioned in chapter 1 and 2, CNN algorithms; and thus deep learning algorithms are inspected through the crack detection case study in order to achieve a holistic understanding not only from theoretical perspective but also by means of hands-on practice. In this context, the CNN implementations are investigated in two different tasks and scales as; crack classification (whether an image contains cracks or not) and crack semantic segmentation (whether a pixel belongs to a crack or not) in Chapter 4 and 5 respectively.

CHAPTER 4

CLASSIFICATION OF CRACK IMAGES – COMPARATIVE ANALYSIS OF PRETRAINED CONVOLUTIONAL NEURAL NETWORKS

> *"With too little data, you won't be able to make any conclusions that you trust. With loads of data you will find relationships that aren't real…"*
>
> *Douglas Merrill*

Image processing is one of the driving fields of convolutional neural network (CNN) research even though these frameworks are also employed in several applications other than image processing. As described in Chapter 3, it is possible to either construct a network from scratch or employ a pretrained network in the course of transfer learning for utilization of CNN's in custom task. Requirements of these two approaches vary especially in terms of quantity of data and challenges in the configuration of the network structure. Within the scope of Chapter 4, transfer learning approach is adopted due to requirement of less data for training and simplicity of the case specific implementation. In that sense, it is aimed to focus on data rather than CNN structure in the scope of data design.

Transfer learning eases utilization of CNN's in custom tasks with their already built configuration and learned parameters with respect to some generic dataset. However, high performance of the pretrained network on the new case is not guaranteed and is highly dependent on the dataset provided to the network and the complexity matching between the task and the network. For the sake of example, a highly acknowledged pretrained network may perform poorly if sufficient number of data samples with adequate variance is not provided. Similarly, if the pretrained network is designed for complex classification tasks, network may be subjected to overfitting when utilized on much simpler tasks.

Even though there are several studies on utilization of CNN's in crack detection task, most of these studies are based on constructing CNN's from scratch and networks utilized in these studies have limited number of layers. The studies focusing on using pretrained networks on crack detection task have proven the applicability of transfer learning for this task. Yet, a comprehensive study inspecting multiple dimensions, such as influence of dataset size, number of convolution layers, learnable parameters, of this complex task is not present. Within the scope of Chapter 4, a comprehensive analysis on performance of pretrained networks on crack detection and the parameters effecting this performance is conducted. The performances of the networks are inspected in four test cases with varying similarities to training dataset to evaluate the applicability of learned features to diverse cases.

4.1. Parameters effecting performance of CNN's on crack detection

Within the scope of Chapter 4, it is aimed to reveal which properties of CNN architecture and how the decisions taken throughout the process effect performance. For this purpose, multidimensional performance analysis on pretrained networks for crack detection task is conducted and effects of the variables listed below are inspected.

4.1.1. Dataset

Size of training dataset and variance among the data play a crucial role in the performance of the network. It is known that pretrained networks require less data compared to the networks which are trained from scratch, as pretrained networks have already adjusted weights by learning from vast amount of data. For transfer learning applications, it is assumed that layer weights are only fine-tuned to adapt to new cases and fast convergence of layer weights is expected. Common practice refers to hundreds to thousands of training data per class (in crack classification task: crack and background) is sufficient for obtaining highly accurate predictions. However, as pretrained networks are often trained for generic classification tasks which performs prediction among numerous object classes with high level features, whether the common practice is applicable to binary problem of crack presence or not is unknown. Furthermore, for subjects having low level features, risk of overfitting is imminent with the increasing number of training data.

The dataset utilized in the present study is based on 550 full resolution images with 3024x4032 pixel dimensions. The full resolution images are subdivided into 224x224 pixel image patches in order to conform with CNN input sizes. 500 of the mentioned images are captured from walls and floors of several buildings in METU Ankara Campus from approximately 1 meter away from the surface and camera facing directly to the target surface. In addition, all 500 images are captured at similar times of the day and the year to have similar illumination conditions. However, dataset have variance in terms of surface finishes, e.g. exposed concrete, plastering and paint. 40K 224x224 pixel images are extracted from the first set of images constituting 500 images and will be named base dataset hereafter. Base dataset contains equal number of positive and negative images and used for training and validation of networks. The base dataset is publicly shared (Özgenel, 2018).

The second set of images contains test data for three different cases and captured at different times of the day and year from different buildings with varying materials. Second dataset contains images from concrete surfaces from buildings, concrete surfaces from pavements and brickwork surface from buildings. Each of the cases are discussed in detail in correspondence with the results. 500 224x224 pixel images are extracted for each test case resulting in a total of 1500 images from 50 full resolution images.

Training dataset: The base dataset is partitioned into three parts as training, validation and test datasets with 0.7, 0.15 and 0.15 ratios as convention. As a result, 28K training, 6K validation and 6K test datasets are obtained. While conserving 6k test dataset, training and validation datasets are randomly reduced to 21K, 14K, 7K, 3,5K, 1,75K, 0,7K and 0,35K. Even though, the resulting dataset sizes don't enable to conduct a grid search to find optimum size of training dataset, it is possible to trace the effect of training dataset size on the performance of CNN's.

Validation dataset: Validation dataset is used to evaluate and monitor the network throughout the training. Learning curves of the training are inspected to detect whether overfitting occurs or not. The 0,7 to 0,15 ratio between training dataset and validation dataset is conserved for varying sizes of training dataset cases.

Test dataset: The performances of the trained networks are inspected on four distinct test datasets. First test dataset consists 6K images which are randomly selected from the base dataset and referred to Test1 case hereafter. The images in Test1 dataset have high resemblance with three training datasets and represents the case which the test images are visually similar to training dataset.

The second, third and fourth test cases are constructed from second dataset representing three diverse cases in terms of illumination, camera orientation and material. All of the test cases have 500 images which 250 consists crack images and 250 consists only background. Test cases are respectively, concrete material – pavement (Test2), concrete material – building (Test3) and brickwork – building (Test4). Test2 and Test3 investigates the transferability of learned features to similar cases but with varying illumination conditions and camera orientations. Test4 is the most challenging task as the background features such as brickwork jointing is visually similar to cracks. The challenges present at each test case are summarized in Table 3. It should be noted that the exemplary images shown in Test1 belongs to the base dataset which 70 and 15 percentage of the dataset is used for training and validation respectively.

Table 3. Test cases and respective challenges

Test Case	Challenge	Testing		Sample Data
		Positive	Negative	
Test1 | concrete - building	NA	3000	3000	
Test2 | concrete - pavement	- camera-surface distance - camera-surface orientation - illumination	250	250	
Test3 | concrete - building	- camera-surface distance - camera-surface orientation - illumination	250	250	
Test4 | brickwork - building	- camera-surface distance - camera-surface orientation - illumination - material texture	250	250	

The resulting sizes datasets are shown in Table 4:

Table 4. Number of images in datasets used for training and validation

Dataset Size	Training		Validation	
	Positive	Negative	Positive	Negative
28K	14000	14000	3000	3000
21K	10500	10500	2250	2250
14K	7000	7000	1500	1500
7K	3500	3500	750	750
3,5K	1750	1750	375	375
1,75K	875	875	188	188
0,7K	350	350	75	75
0,35K	175	175	38	38

4.1.2 Number of Epochs for Training

The relation between performance of networks and number of epochs, i.e. the number of iterations which network goes through all training samples, is inspected to observe how fast the networks converge to obtain high accuracy. As the number of epochs increases, networks have tendency to overfit to training samples. Yet, minimum number of epochs are highly dependent on the nature of the subject matter, variance among the dataset and similarity between training data and test data. Within the scope of this study, all networks are trained for 10 epochs with varying dataset sizes.

4.1.3 Network dependent parameters: Number of convolutional layers and number of learnable parameters

Number of convolutional layers and number of learnable parameters denote the complexity and measures for the *deepness* of the networks. As the configurations of pretrained networks are already established, it is not possible to conduct a grid search for these parameters. Yet, the selection of networks presents variance among number of layers and number of learnable parameters to enable analysis of how the complexity of networks effects the performance for crack detection task. Characteristics of the networks used in tests are described as below. It should be noted that the descriptions and respective measures of success is dependent to networks' performance on ILSVRC dataset which contains more than 1,2M images from 1000 object classes.

AlexNet (Krizhevsky, Sutskever and Hinton, 2012)***:*** AlexNet is the first CNN to perform considerably well in ILSVRC in 2012. Implementation of ReLU and dropout layers to avoid overfitting and decrease training time are major contributions to the field which are being widely used. The network has a simple architecture consisting five convolution and three fully connected layers. The layer structure is hierarchical which layers are structured in a linear way, one following the other.

VGG Net (Simonyan and Zisserman, 2015)***:*** VGG Net, which has VGG16 and VGG19 versions with different number of layers, has an influential role in the field as the study emphasized the importance of depth (in terms of number of layers) over the complexity of the content of the layers. The architecture uses consistently 3x3 convolutional layers together with pooling to decrease the spatial dimension. VGG Net is considerably larger than the AlexNet and VGG16 and VGG19 versions are shown in APPENDIX I. Even though VGG Net was not the winner of ILSRVC 2014 Challenge, it is used widely as a baseline with its robust structure and as a basis for deep learning tasks other than classification.

GoogleNet (Szegedy *et al.*, 2015)***:*** GoogleNet, as the winner of ILSRVC 2014 challenge, is one of the first studies which did not use hierarchical approach for the formation of layers. The introduction of inception module, which an input is processed in 4 different paths and are concatenated as one output, changed the way the CNN's are structured. Such networks are realized by employing directed acyclic graph method enabling non-hierarchical connection among layers and called as DAG Networks. Structure of the inception module is shown as below:

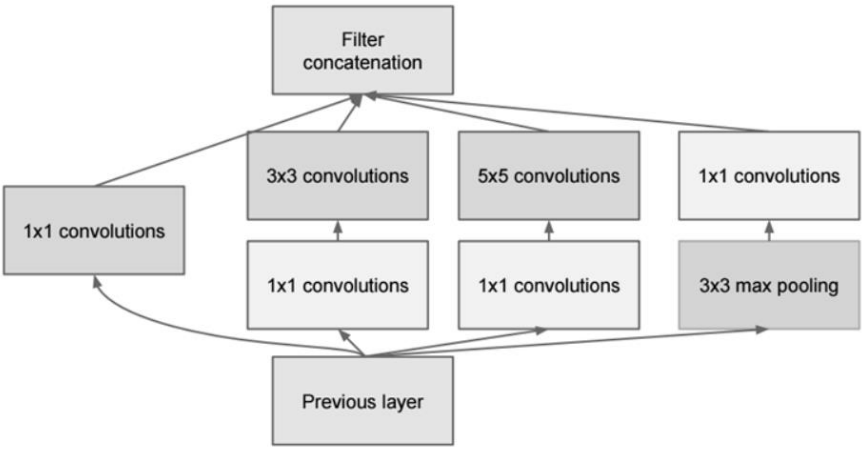


Figure 23. GoogleNet inception module (Szegedy *et al.*, 2015)

Apart from the inception module, GoogleNet uses single fully connected layer which dramatically reduces the number of parameters. GoogleNet has 12 times fewer parameters than AlexNet while having nearly 5 times more layers (over 100 layers). The full architecture of GoogleNet is shown in APPENDIX I. Inception v3 and v4 are also developed and achieved huge success which are based on GoogleNet architecture.

Microsoft ResNet (He *et al.*, 2016)***:*** ResNet, which is the winner of ILSVRC 2015 by surpassing %5 top5 error with %3,6 error rate, also employs DAG Network structure by constructing residual blocks. Residual blocks have two pathways which one path has series of convolution and ReLU layers while other path directly transmits the input data. The outputs of two paths are summed. Simple visualization of a residual block is shown as below and full ResNet architecture is shown in APPENDIX I.

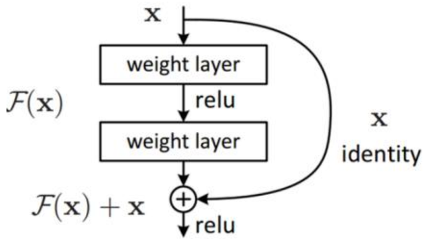

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 24. ResNet residual block (He *et al.*, 2016)

ResNet has three versions of ResNet 152, ResNet 101 and ResNet 50 which contains 152, 101 and 50 layers respectively.

Study of Canziani *et al.* (2016) summarizes the computational cost and performance of well-known CNN architectures and the results are shown as below:

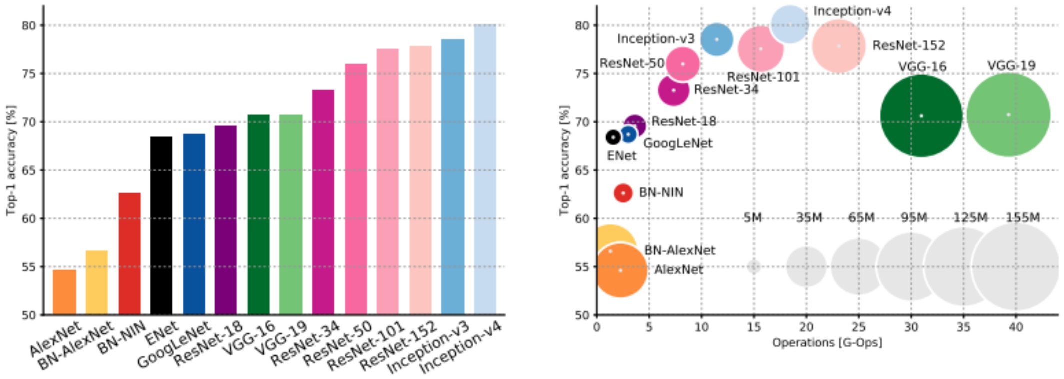


Figure 25. Network performance (top 1 accuracy) vs computational cost (number of operations) for well-known CNN's (Canziani, Paszke and Culurciello, 2016)

As can be seen from Figure 25, the performance of ResNet surpasses the performance of VGG, GoogleNet and AlexNet architectures while recent versions of GoogleNet (Inception v3 and v4) have higher performance for top 1 accuracy. VGG is seen to have the most computational cost among other while achieving the best performance among simple/plain networks having hierarchical layer connections.

Pretrained networks and respective number of convolutional layers and learnable parameters are shown in Table 5.

Table 5. Pretrained networks and respective number of convolution layers and learnable parameters

	# of Convolution Layers	**# of Learnable Parameters**
AlexNet	8	60M
VGG16	16	138M
VGG19	19	144M
GoogleNet	22	7M
ResNet50	50	25.6M
ResNet101	101	44.5M
ResNet152	152	60.2M

As the relation between number of convolution layers and number of parameters are not linear, it is possible to trace which one of the parameters have more influence on the performance of the networks. Other constituents of the networks such as utilization of batch normalization, number of pooling layers, layer configuration are not inspected within the scope of this study.

4.2 Evaluation Metrics

Confidence of the results obtained from networks provide valuable information regarding how well the networks learn the discriminant features of cracks and how confident they are in their predictions. Conventional evaluation metrics such as accuracy, precision and recall, F score are based on only decisions and it is not possible to make an evaluation regarding confidence of the networks. Formulas of the mentioned conventional metrics are as given below:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{10}$$

$$Precision = \frac{TP}{TP+FP} \tag{11}$$

$$Recall = \frac{TP}{TP+FN} \tag{12}$$

$$F - score = 2.\frac{precision.recall}{precision+recall} \tag{13}$$

Where TP, TN, FP, FN denote True Positive, True Negative, False Positive and False Negative and resemble (mis)matching of targeted class and prediction.

In order to take confidences of the networks into consideration for the evaluation of network performances, a novel metric which is named confidence weighted accuracy is proposed.

Performance of the networks are compared with respect to their confidence weighted accuracies. The formulas of mean accuracy and mean confidence weighted accuracy are as below:

$$accuracy = \frac{\sum_{i=1}^{N} \delta_{truth_i, prediction_i}}{N} \tag{14}$$

$$confidence\ weighted\ accuracy = \frac{\sum_{i=1}^{N} confidence_i * \delta_{truth_i, prediction_i}}{N} \tag{15}$$

where, $\delta_{i,j}$ is Kronecker delta function which output 1when *i* is equal to *j* and 0 when *i* is different than *j*. In other terms, $\delta_{i,j}$ outputs 1 when networks makes a correct prediction and 0 when network fails.

Confidence weighted accuracy (CwA) calculation contains more information regarding how well the networks learn crack features by taking the confidence of network into account. Contribution of CwA can be illustrated with the example cases below.

Table 6. Accuracy vs confidence weighted accuracy on sample test cases

Case		**Predictions**				**Accuracy**	**CwA**
1	Prediction	True	True	False	False	0.5	0.375
	Confidence	0.9	0.6	0.9	0.6		
2	Prediction	True	True	True	True	1	0.675
	Confidence	0.675	0.675	0.675	0.675		
3	Prediction	True	True	True	False	0.75	0.675
	Confidence	0.9	0.9	0.9	0.6		

Table 6 illustrates 7 cases which varies in number of predictions and confidence scores for these predictions. In the first case, 4 predictions are made, which two of these predictions are correct with 0,9 and 0,6 confidence scores respectively and two predictions are wrong with the same confidence scores, i.e. one high and one relatively low confident score for both correct and wrong prediction. While accuracy metric outputs 0,5 score denoting equal distribution of predictions, CwA output 0,375 by penalizing the low confidence of true prediction.

CwA scores for the second and third cases are both 0,675 although accuracy scores output 1 and 0,75 respectively. While third case has one wrong prediction, it is as valuable as the network making all correct predictions but with low confidence.

Effective range of accuracy and CwA metrics is conserved as networks can score between 0 and 1. Essentially, CwA opts for correct predictions with high confidence and penalizes both wrong predictions and low confidence scores.

One drawback of CwA metric can be defined as the calculation is based on whether the predictions are correct or not and disregards the ratio of classes, i.e. how many of the predictions are true positive, true negative, false positive or false negative. However, as the crack detection task is treated as a binary problem within the scope of this study and classes have equal importance. Furthermore, the datasets which CwA is analyzed are balanced in terms of number of samples per class. Hence, the information on contribution of confidence is chosen over information on class dependent behavior of predictions.

4.3 Results and Discussions

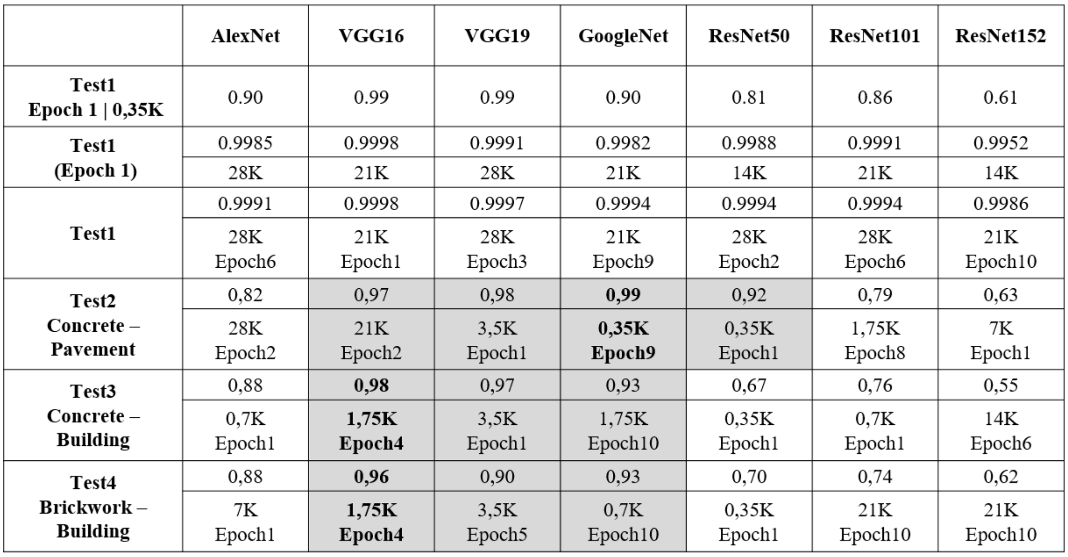
Each of the previously mentioned pretrained networks are trained with varying sizes of datasets for 10 epochs resulting in 560 (7 networks, 8 training datasets, 10 epochs) trained networks. These networks are subjected to 4 different test cases and confidence weighted accuracy results are stored in a 4D matrix with 2240 (7 networks, 8 training datasets, 10 epochs, 4 test cases) members. Both accuracy and confidence weighted accuracy scores are shared in Appendix 2.

The networks used in the scope of this study are pretrained on ImageNet (Stanford Vision Lab, 2018) data and obtained from MatConvNet (MatConvNet Team, 2018) website. All tests are conducted with MatConvNet library and Matlab 2017a (Mathworks, 2018) on a desktop workstation with 2 Intel Xeon E5-2697 v2 @2,7 GHz CPU cores, 64GB RAM and NVIDIA Quadro K6000 GPU. On the other hand, multidimensional analysis approach is applicable to any other pretrained network and any other programming language.

For the sake of simplicity, highest scoring cases per test case are shared in

Table 7 and all results are shared in Appendix 2. In addition, results are discussed per case and per parameter as below.

Table 7. Highest scoring test and validation CwA scores for pretrained networks



	AlexNet	**VGG16**	**VGG19**	**GoogleNet**	**ResNet50**	**ResNet101**	**ResNet152**
Test1 Epoch 1 | 0,35K	0.90	0.99	0.99	0.90	0.81	0.86	0.61
Test1 (Epoch 1)	0.9985	0.9998	0.9991	0.9982	0.9988	0.9991	0.9952
	28K	21K	28K	21K	14K	21K	14K
Test1	0.9991	0.9998	0.9997	0.9994	0.9994	0.9994	0.9986
	28K Epoch6	21K Epoch1	28K Epoch3	21K Epoch9	28K Epoch2	28K Epoch6	21K Epoch10
Test2 Concrete – Pavement	0,82	0,97	0,98	**0,99**	0,92	0,79	0,63
	28K Epoch2	21K Epoch2	3,5K Epoch1	**0,35K Epoch9**	0,35K Epoch1	1,75K Epoch8	7K Epoch1
Test3 Concrete – Building	0,88	**0,98**	0,97	0,93	0,67	0,76	0,55
	0,7K Epoch1	**1,75K Epoch4**	3,5K Epoch1	1,75K Epoch10	0,35K Epoch1	0,7K Epoch1	14K Epoch6
Test4 Brickwork – Building	0,88	**0,96**	0,90	0,93	0,70	0,74	0,62
	7K Epoch1	**1,75K Epoch4**	3,5K Epoch5	0,7K Epoch10	0,35K Epoch1	21K Epoch10	21K Epoch10

The discrepancy between accuracy scores and CwA scores is illustrated in Figure 26 and Figure 27 and shared in Table 8.

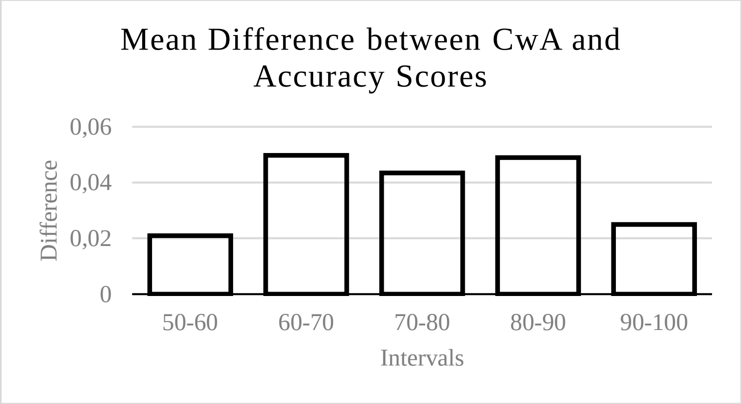


Figure 26. Mean difference between CwA and accuracy scores (drawn by author)

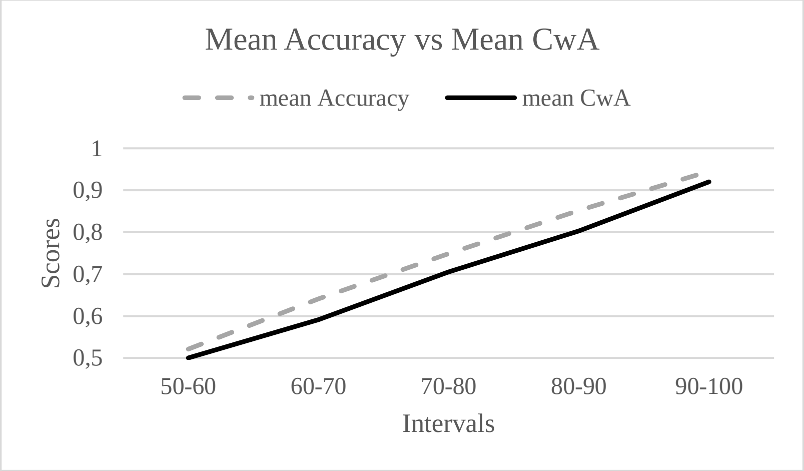


Figure 27. Mean accuracy and mean CwA vs %10 intervals (drawn by author)

As can be seen from the figures and table, the biggest difference in %10 intervals is observed in %60-%70 accuracy interval with %4.98 change. %90-%100 accuracy interval where the majority of the network scores concentrated the influence of CwA is around %2.5. However, %2.5 contribution of CwA calculation enable sorting among the performances of networks. In addition, the positive influence of CwA which increases accuracy scores denote that the networks have high confidence scores when making correct predictions while having low confidence scores for wrong predictions.

Table 8. Mean accuracy, mean CwA scores for test cases

Mean Accuracy	**50-60**	**60-70**	**70-80**	**80-90**	**90-100**
Test1	NaN	0.606	NaN	0.86	0.993
Test2	0.512	0.652	0.743	0.853	0.941
Test3	0.518	0.646	0.746	0.851	0.929
Test4	0.532	0.658	0.758	0.843	0.917
Mean	0.521	0.6408	0.7485	0.852	0.945

Mean CwA	**50-60**	**60-70**	**70-80**	**80-90**	**90-100**
Test1	NaN	0.5455	NaN	0.754	0.986
Test2	0.495	0.609	0.679	0.822	0.904
Test3	0.5	0.609	0.721	0.825	0.909
Test4	0.505	0.6	0.716	0.809	0.881
Mean	0.5	0.591	0.705	0.803	0.92

Difference	**50-60**	**60-70**	**70-80**	**80-90**	**90-100**
Test1	0	0.0605	0	0.106	0.007
Test2	0.017	0.043	0.064	0.031	0.037
Test3	0.018	0.037	0.025	0.026	0.02
Test4	0.027	0.058	0.042	0.034	0.036
Mean	0.021	0.0498	0.0435	0.049	0.025

Test1 CwA for first epoch with 0,35K dataset

Test 1 accuracies for first epoch with the least number of image samples are inspected to observe how fast the networks converge. AlexNet, VGG16, VGG19 and GoogleNet achieved more than 0,9 CwA. Especially fast convergence of VGG16 and VGG19 is significant when the dataset size is considered. On the other hand, even though ResNet networks scored much lower at first epoch, they scored over 0,9 CwA at the second epoch.

Test 1 CwA

As all networks scored over 0,9 CwA at the second epoch, later iterations are basically fine tuning to converge to CwA of 1. All networks benefitted from the larger dataset sizes scoring the highest CwA with over 21K image samples. However, when the scores present in Appendix 2 are inspected, the differences are barely noticeable and it is not possible whether overfitting occurs as the validation dataset is randomly chosen from the base dataset which the training datasets are also extracted from. Whether the learned features are transferrable to other cases or the networks overfit to the training samples are tested in Test2, Test3 and Test4 which have variations in terms of camera orientation, illumination and material.

Test 2 CwA for concrete material – pavement as application area

Even though the application area is different from the training datasets, the cracks in pavement have less shadowing and visually more discernable with respect to Test 3 case (concrete material- building as application area). As can be seen from Table 3, the images are more homogeneously illuminated resulting in high contrast among crack regions and material texture. When scores are inspected, while VGG networks and GoogleNet networks scored similar to the scores of Test1, AlexNet and ResNet networks scored much lower. Among ResNet networks, ResNet50 was the most successful one with 0,9 CwA. The tendency to overfit is observable with the increasing number of layer for ResNet family with decreasing CwA scores as 0,77 and 0,62 by respectively ResNet101 and ResNet152. Low score of AlexNet is attributed to the low number of convolution layers as VGG networks, which have similar architecture but with more convolution layers, achieved considerably high at Test2.

Test 3 CwA for concrete material – building as application area

Test 3 data is similar to training data and Test 2 data in terms of having concrete material texture. However, camera - surface orientation, illumination conditions and camera distance shows variance with respect to the training dataset. Test 3 can be considered as the test whether the networks achieved a generic solution for prediction of cracks on concrete surfaces. Similar to Test 2 case, VGG networks and GoogleNet scored over 0,92 with VGG16 achieved the highest score with 0,98. With the increasing variance ResNet family scored below 0,8 which is an indication of overfitting. In addition, the highest scoring ResNet50 and ResNet101 networks are trained with 0,35K and 0,7K training samples which supports overfitting suspicion. As the number of samples for training increases, scores of ResNet family networks decrease.

Test 4 CwA for brickwork material – building as application area

Test 4 is the most challenging task among test cases as the background texture and jointing for brickwork have high probability of misclassification as cracks. Hence, Test 4 reveals whether the networks operate based on contrast differences or are able to extract crack features and detect cracks regardless of the material cracks reside on. With lower scores but having the same trend, VGG 16 and GoogleNet achieved the highest scores among the pretrained networks which are utilized in the scope of this study. Especially, 0,96 score of VGG16 is promising in terms of obtaining a generic crack classifier regardless of the material. While ResNet networks scored similarly to other test cases, AlexNet scored around 0,88. The consistent scores of AlexNet for all test cases show that AlexNet is able to extract crack features which are adaptable to other cases. Yet, the relatively simple architecture of AlexNet is more susceptible to noise resulting in scores below 0,9.

Size of training dataset

When the scores for all tests are considered, it is observed that the accuracies of large training datasets are comparable with small training datasets per case. For Test 1, highest scores are obtained with the largest training datasets. As the training dataset and test dataset for Test1 are extracted from the same base dataset, all networks benefit from more image samples in the course of training. Therefore, it can be generalized as, in the event of having similar images for test cases, the size of training dataset positively influences the accuracy. On the other hand, when the test images show variance in terms of camera-surface orientation, camera distance to surface and/or illumination conditions, networks tend to achieve higher scores with smaller training datasets. 3,5K images are seen to be sufficient for learning crack features without subjecting to overfitting. Yet, the optimum size of training dataset is highly related with the test case, images and image capturing conditions and 3,5K training dataset size should not be perceived as optimum.

The same analysis is also valid for number of training epochs. If the training images and test images have high similarity, number of epoch for training increases the accuracy whereas for diverse cases, higher number of epochs increases risk of overfitting and networks tend to have bias towards training set.

Number of convolution layers and learnable parameters

When Test2, Test3 and Test4, which challenges networks in terms of illumination conditions, camera orientation and distance to surface, and subject material, are inspected VGG networks and GoogleNet networks have higher scores without being subjected to overfitting. As shown in Table 5, these networks have 16 to 22 convolution layers and no correlation between learnable parameters. Despite having different network architecture approaches, it can be inferred that an optimum window between 16 and 22 layers of convolution for obtaining a generic crack classifier. This inference is also supported with the decreasing performance of ResNet family with the increasing number of layers.

Computational time

Computational time required for training 28K training dataset for 1 epoch for all networks and their mean accuracies of highest scoring networks per case are shared in Table 9.

Table 9. computational time for 28K training set for 1 epoch

28K dataset | per Epoch	Training Time (s)	Mean CwA
AlexNet	133	0,89
VGG16	2827	0,98
VGG19	2943	0,96
GoogleNet	1227	0,96
ResNet50	1666	0,82
ResNet101	2447	0,82
ResNet152	3789	0,70

The computational cost of the networks are as expected, having correlation with the study of Canziani *et al.* which is shown in Figure 25. When the computational times analyzed in consideration with their performance, GoogleNet performs scores 0,95 in less than half computational time than VGG networks. However, VGG16 is the most successful in terms of CwA. On the other hand, AlexNet provides near 0,9 CwA with considerably less computational time which increases its preferability when experimenting on the datasets but not for actual test cases.

Within the scope of Chapter 4, a multidimensional analysis of variables effecting performance of pretrained networks on crack classification. The findings obtained in this analysis are aimed to provide foreknowledge to researchers working on crack detection task whether for building applications or not. It is believed that, the results and optimal ranges achieved in the course of the analysis are applicable to other cases which the discriminative visual features are limited. Furthermore, the analysis present herein provides a framework for the evaluation of pretrained networks and enable selection of the best performing network for future studies.

CHAPTER 5

SEMANTIC SEGMENTATION OF CRACK IMAGES QUADP (A NOVEL QUADTREE INTEGRATED DEEP LEARNING ALGORITHM)

> *"There's no sense in being precise when you don't even know what you're talking about."*
>
> *John von Neumann*

Semantic segmentation provides more information with respect to image classification methods as the goal is to output pixel wise prediction and labeling of images resulting in object regions with finely traced contours. By nature, as the outlines of the objects from different classes are obtained, it is possible to use the output to predict form, size and spatial relations between objects in the scene. In case for semantic segmentation of cracks, various features of detected cracks can be measured with respect to the crack boundaries such as orientation, width, number of cracks in an image. Furthermore, semantic segmentation applications provide significant information regarding the behavior of the networks utilized in the implementation. It is possible to trace which regions activate the network for classification of images and by this way it enables users to have an insight on the working principles of CNN’s.

There are several studies aiming semantic segmentation with the utilization of deep learning frameworks. These studies have differences in their methodology and approach towards semantic segmentation of images.

5.1. Semantic Segmentation Approaches

Three of the highly acknowledged studies; namely Fully Convolutional Networks, Deconvolutional Networks and SegNet, are explained below in consideration with the differences in their approaches while utilizing CNN framework in their studies. It should be noted that present studies are developed with the aim of achieving a generic framework and tested against generic datasets such as PASCAL VOC dataset.

Fully Convolutional Network (FCN)(Long, Shelhamer and Darrell, 2014)***:*** Studies of Long, et.al. can be given as an exemplary study which is commonly used and referenced. In their study, fully connected layers of pretrained networks (AlexNet, GoogleNet, and VGG) are converted to fully convolutional layers and the activation maps of the determined layers are upsampled to match input image dimensions. Then, upsampled activation maps are overlapped to obtain final output. Three versions of the fully convolutional network (FCN) is developed and their architecture is shown below:

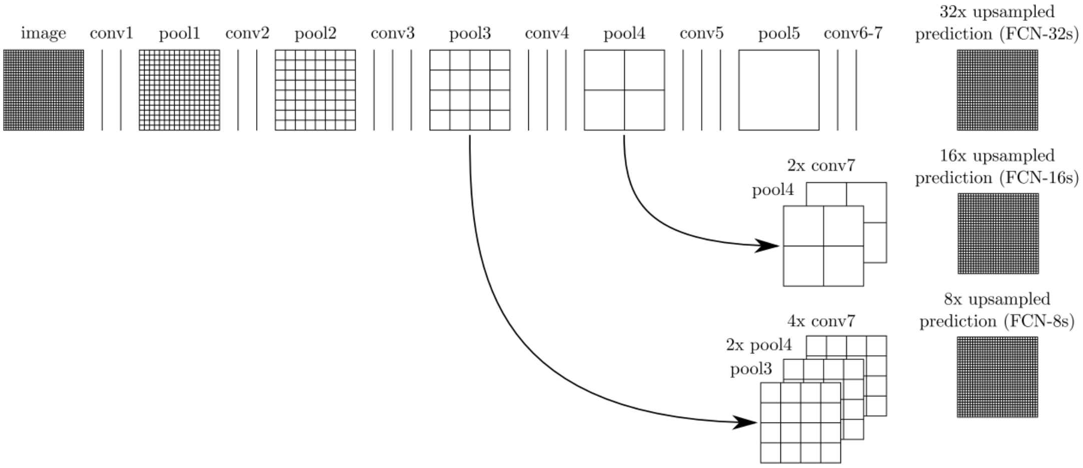


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

Figure 28. FCN32, FCN16, and FCN8 layer architecture (Long, Shelhamer and Darrell, 2014).

While FCN32s version only upsamples 7th convolution layer activation, FCN16s takes 4th pooling layer activation and 7th convolution layer activation into account and FCN8s uses 3rd pooling layer activations in addition to FCN16s. As the upsampling ratio decreases, the network is operated as a DAG network and combines information gathered from different layers. Among the pretrained networks used in this study, VGG shows the best performance against GoogleNet and AlexNet. It is reported that the consistent kernel size for convolution and hierarchical structure of VGG network makes it more suitable for reconstruction of images for segmentation tasks. Native input and output sizes of images are 500 to 500 pixels.

The upsampling mentioned in the study is not an image interpolation method but rather a learnable filter which is named as deconvolution layer. Deconvolution layer basically conducts the inverse of convolution layer. Deconvolution layer is introduced within the scope of this study and employed by various benchmarking studies in semantic segmentation.

DeconvNet (Noh, Hong and Han, 2015)***:*** The studies of Noh, et.al also utilizes deconvolution layers based on the VGG16 network. DeconvNet architecture follows a more hierarchical way from FCN by consistently downsampling the image until classification and upsampling to reach semantic segmentation result. The architecture of DeconvNet is shown in Figure 29.

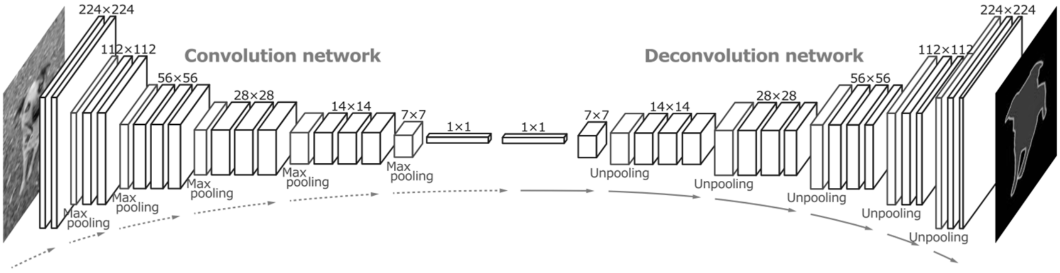


Figure 29. DeconvNet architecture (Badrinarayanan, Kendall and Cipolla, 2015)

In addition to deconvolution layer, unpooling layer is introduced in DeconvNet study. Unpooling layer effectively upsamples the data in consideration with respective pooling operation in the convolutional network. The symmetric architecture of convolutional and deconvolution network sections enables DeconvNet to keep track of which pixel is used for output of pooling and which pixels are discarded and reflects this information to unpooling operation. As DeconvNet is based on VGG16, native input-output sizes of images are 224 to 224 pixels.

SegNet: First version of SegNet has a similar architecture with DeconvNet as both architectures use VGG 16 as a base for convolution part (Badrinarayanan, Handa and Cipolla, 2015). In contrast with DeconvNet, SegNet reduces the number of parameters to be learned to increase computational capabilities resulting in better accuracy results in fewer iterations. The study is then extended to enable custom layer formations (Badrinarayanan, Handa and Cipolla, 2015). In this study, the network architecture is chosen to be flat, each layer having the depth of 64 and having fully connected structure among the layers while preserving convolution-deconvolution dual (named as encoder-decoder in the study). Both studies are conducted on 360 to 480 pixel images. 4 encoder and 4 decoder layers are present in the SegNet and are shown in Figure 30.

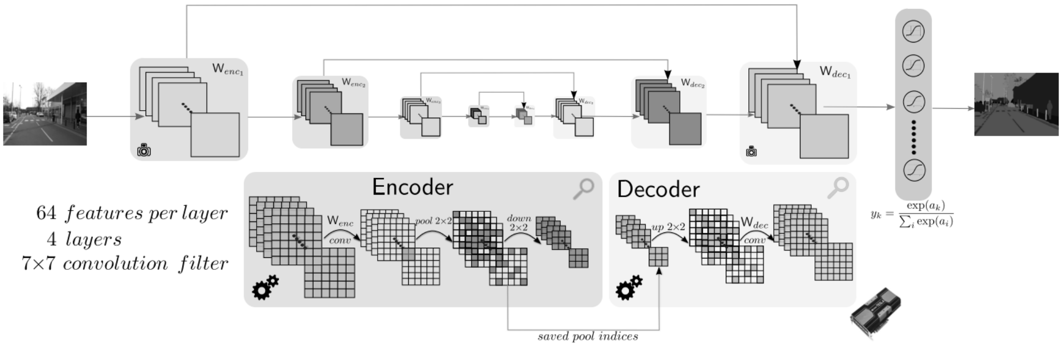


Figure 30. SegNet architecture (Badrinarayanan, Handa, et al., 2015)

In contrast to classification applications, semantic segmentation frameworks require ground truth images which each pixel in the input image are labeled rather than object classes per image. Hence, the required data for training such networks are proportional with the dimensions of the training images. On the other hand, for classification operation, only the class which the image belongs to is required. Despite having different approaches in their frameworks, all of the studies mentioned above are highly acknowledged and widely used semantic segmentation applications and have comparable accuracies in terms of semantic segmentation metrics.

5.2 Semantic Segmentation of Cracks

Semantic segmentation of crack images contains possibly vital information for assessment of cracks. It is possible to predict the number of cracks, their orientation and pixel based width by post processing the object outlines extracted by means of semantic segmentation. Furthermore, it is possible to conduct metric measurement if the camera – surface orientation, camera distance and extrinsic camera parameters are known for the image input. Hence, precise segmentation of cracks has utmost importance for fields including but not limited to autonomous inspection and structural health monitoring of buildings.

CNN’s are powerful frameworks for image processing tasks and have shown promising results on achieving a generic crack classification framework as discussed in Chapter 4. Being adaptable to various materials and cases while achieving high accuracy forms a basis for utilization of CNN’s in semantic segmentation of cracks.

However, the challenges caused by the nature of cracks, which are present in Chapter 2 for crack classification, pursue with increasing influence in semantic segmentation. These challenges are recapped and additional challenges in consideration with the semantic segmentation task as defined as below:

Challenges caused by the low level discriminative features of cracks:

Discriminative crack features are easily confused with noise in the background texture, foreign objects and/or irregularities in application such as exposure of jointing. For semantic segmentation, noises in a focused part of an image cause crack class activation even if the region is not a part of the crack. For crack classification task, softmax classifier compensate false activation and even if the confidence scores decreases, prediction results are less effected from local noises. Another challenge is the inhomogeneous illumination of the surface which causes occlusion of crack segments endangering the conservation of crack continuity. Shadowed part of the images reduces sensitivity to discriminative features of cracks. When the goal is determining the presence of crack, conserving the crack continuity is not crucial as long as the networks detects the presence of cracks. However, for semantic segmentation, loss of crack continuity results in error for making measurements. For example, a single crack, which is occluded due to inhomogeneous illumination, may result in semantic segmentation output having more than one crack segments.

Challenges caused by the amount of data:

Building inspection by means of visual data requires collection of vast amount of data. Even though CNN’s have capability of processing more information with respect to machine learning classifier, they are not necessarily lightweight frameworks either. As the number of layers and number of learnable parameters increase, memory concerns arise while processing high resolution images. In such cases, either images are subdivided into image patches and processed separately to combine the results later or images are downsampled if the fine details are not important. For crack detection case, downsampling high resolution images is not an option as even hairline cracks matter for classification. In the case of subdividing and combining back the image patches, the continuation of cracks is not guaranteed as image patches are processed independently.

In order to overcome the challenges mentioned above, a novel method which combines quadtree division algorithm and CNN’s is proposed and will be referred as QuadP hereafter.

5.3 QuadP

QuadP aims to conduct image segmentation by extracting and processing probability maps from either binary (presence) or weighted (confidence) classification results. Within the framework of QuadP, while the challenges based on the low level discriminative features of cracks are addressed by utilizing classification results instead of achieving activation maps, the challenges caused by the amount of data are addressed with the utilization of quadtree division method.

Quadtree is a highly acknowledged tree data structure which is widely utilized for data encoding in fields varying from image processing to state estimation in control theory. In general terms, quadtree division algorithms rearrange data in a tree data structure to enable fast and easy access among scattered data. For this purpose, a particular addressing method is utilized. This addressing is based on subdividing the dataspace into four quadrants in case of fulfilling a specified criterion. This criterion is also named decomposition condition. The process of subdividing continues until a termination condition or all of the data are indexed separately. As the decomposition occurs as long as the decomposition criterion is satisfied, the algorithm focused on regions where concerned data exists and does not use memory for unimportant parts of dataspace. Various implementation methods of this basic idea are proposed and widely utilized in different cases. Some of the region quadtree, which the data region is directly divided into four equal quadrants having data points reside in; point quadtree, which the region is divided in a way that the edges of regions correspond to data points; and matrix quadtree, which is operated like region quadtree but the division iteration is continued until the smallest cell is obtained. Among these methods, especially matrix quadtree is widely used in image encoding where pixels are treated as the smallest cells. Quadtree division algorithm is known to be more memory efficient with respect to methods storing all data in dataspace.

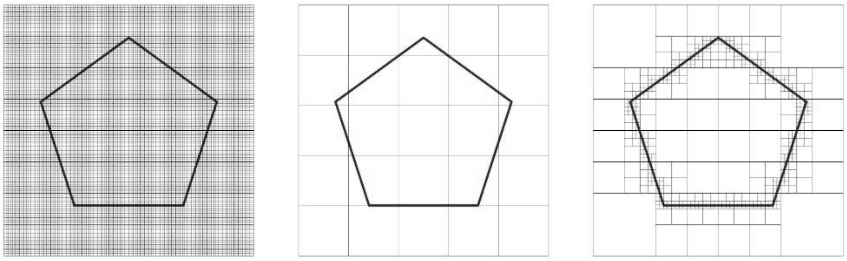


Figure 31. Pixel based data storage (raster images), subdivided image patches for CNN and quadtree division result (drawn by author)

In Figure 31, three images illustrating different data storing strategies are shared. First image represents a raster image where data for all pixels are stored individually. For high resolution images, memory consumption is the highest among the three images. Second image represents the conventional subdivision of images in cases which CNN operations have memory concerns. Even though it is more memory efficient, the divisions represent the data coarsely. Third image represents quadtree division. The image is iteratively subdivided into quadrants. It is both memory efficient and have fine details representing the data.

QuadP is constructed on the idea of utilizing CNN classification results as the decomposition condition of quadtree division algorithms to conduct semantic segmentation of cracks. Consequently, four main objectives are determined to be achieved. These objectives are explained as below:

1. *Precision:* As the output of the QuadP, precise crack outlines are expected to be able to conduct measurement. Precision referred here is not only how the obtained outlines corresponds to the crack region but also how the obtained outlines perform in measurements. For this purpose, the outputs are evaluated both with respect to pixel-wise accuracy and crack based metrics such as number of objects and mean orientation.
2. *Flexibility:* QuadP is desired to be flexible in terms of being operable in conjunction with any decision making framework. Even though CNN’s have proven themselves on crack detection task, QuadP is aimed to be adaptable to any classification method so that performance of QuadP can be improved in the presence of a better performing classification framework.
3. *Lightweight:* QuadP is aimed to be lightweight in terms of computational memory so that the method enables operation on high resolution images without losing the connectedness of juxtaposed multiple images.
4. *Robustness:* QuadP is aimed to be robust in terms of being insensitive to noise which is frequently encountered in building inspection applications while effectively focusing on crack regions and features.

Motivated with the objectives mentioned above, QuadP is developed as a hybrid method of CNN classification and quadtree division algorithm. How these methods are blended is shown in Figure 32.

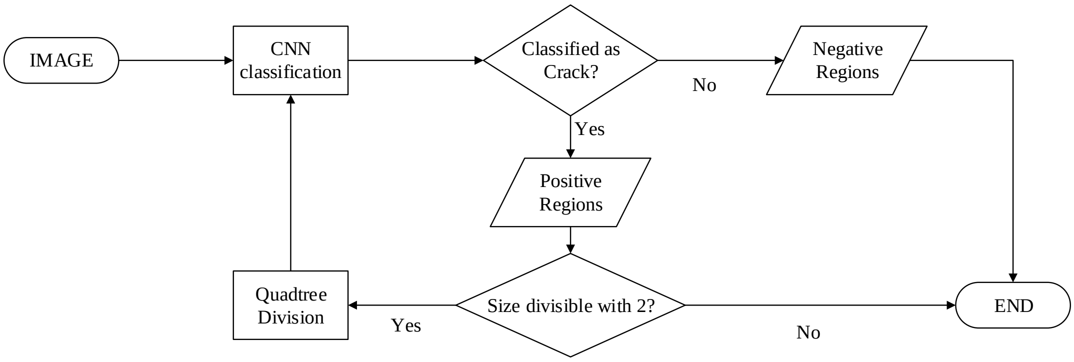


Figure 32. CNN and quadtree division in QuadP flowchart (drawn by author)

Workflow of QuadP can be explained recursive division of positively classified regions as cracks into quadrants and subjecting the quadrants to crack classification for next iteration. By nature, quadtree division algorithm focuses on where the relevant information is present. Due to the focusing, resulting quadrants contain less pixels at each iteration. When regions contain limited number of pixel failing to represent discriminative crack features, CNN algorithm classifies the region as a non-crack region even if the previous iteration results in positive classification. The process is iterated until the resulting image region is not divisible to quadrants or every region is classified as non-crack regions. In the course of subdivision, both indices of regions and confidence results of the respective classification is stored. As each image region is classified individually and results are merged after the classification results are obtained, memory requirement of QuadP is bounded with the memory requirement of CNN for classifying a single image region. On the other hand, QuadP algorithm is suitable for parallelization of the process and when the memory capacity allows parallel processing of multiple image regions, computational time can be reduced.

5.3.1 Issues caused by nature of cracks and semantic segmentation task

The challenges due to the nature of cracks which mentioned above are inspected in detail to be addressed while implementing QuadP algorithm. These challenges are illustrated and discussed as below:

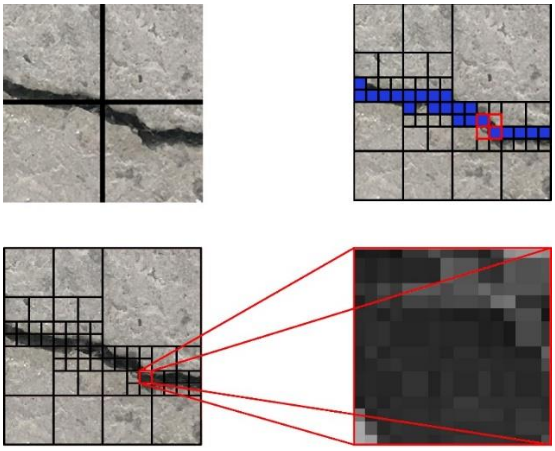


Figure 33.Issues caused by the nature of cracks in adaptation of quadtree division to CNN. Object corresponding to the region border (top left), losing object connectedness and jaggedness of object outlines (top right), losing features due to excessive focusing (bottom left and bottom right) (drawn by author)

Object corresponding to the region border: While processing high resolution image, the image is subdivided into image patches. Division borders play a crucial role for classification task as in the event of cracks corresponding to the subdivision/region border, it is not possible to detect discriminative crack features. In such cases, classification algorithm outputs false negative results, even though the region of interest contains parts of the whole. In Figure 33 top right image, bottom left quadrant is classified as false negative.

Losing object connectedness: Both quadtree division and subdivision for processing the high resolution images with CNN focuses on the content of subdivided region by nature. Hence, the relationship between neighboring regions are disregarded. When considered with objects corresponding to region borders, object connectedness may be lost which does not reflect the actual case. In such cases, the measurements conducted on semantic segmentation output gives erroneous results as multiple crack segments are obtained from single crack. In Figure 33 top right image, object connectedness is lost at regions marked with red.

Losing feature due to excessive focusing: As quadtree iteratively divides positively classified image patches into quadrants, the dimensions of region of interest gets smaller at each iteration. Before reaching pixel scale, focused region loses its ability to hold discriminative crack features. In such cases, classification algorithm fails to predict the region as a positive crack region even if the same region is classified positively in the previous iteration. In Figure 33 bottom image, the region marked with red loses features due to the excessive focusing and is classified as negative even though the region resides inside crack region.

Jaggedness of object outlines: As the method utilizes quadtree division for refinement of object boundary, the positively classified regions are subdivided into quadrants in horizontal and vertical axis. Hence, the object boundary is limited with only vertical and horizontal lines. As image classification requires feature detection of concerned object, obtained object outlines do not have pixel scale precision. In Figure 33 top right image, blue regions denote crack presence. As can be seen from the image, the outlines show jaggedness.

5.3.2 Resolutions for addressing the issues

In order to resolve these issues, three methods are developed as:

i) *defining two meta axes as quadtree and control axes for decomposition,*
ii) *region scoring,*
iii) *3D interpolation of region scores*

These methods also construct the bridge between quadtree division algorithm and CNN. Methods and corresponding addressed issues are shown in Table 10. Each of the method is described in detail below.

Table 10. Issues and corresponding approaches

Issue	Method
Object corresponding to the region border	2 meta axes
Losing object connectedness	2 meta axes
Losing feature due to excessive focusing	Region Scoring
Jaggedness of object outlines	3D interpolation of region scores

2 meta axes approach: In quadtree division implementations, the regions are indexed in a way to access the data in query. In image processing implementations, this indexing is constructed with respect to row and column values of the regions and iteration number. However, as mentioned before, strictly dividing the regions into quadrants result in losing the relation between neighboring regions. This approach causes losing crack continuity in crack detection and segmentation applications. In order to tackle this challenge and preserve crack continuity by taking neighborhood relations into account, quadtree division is conducted in two instances which one instance is shifted with half the region dimensions in vertical and horizontal axes. Two instances are then superposed to merge the information gathered from two axis system. While first instance conducts regular quadtree division, the second instance controls the neighborhood relations of regions examined in first instance. These instances are named as quadtree and control meta axes to refer spatial indexing which the quadtree division results are stored. The approach is illustrated in Figure 34.

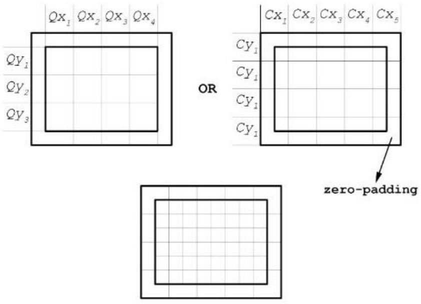


Figure 34. Quadtree and control meta axes decomposition and resulting decomposition by merging two (drawn by author)

For the first iteration only, the original image is zero-padded, i.e. outer boundary is populated with zero values, with half of the region dimension in order to take image borders into account for superposing. By nature, zero-padded regions don't have any relevant information for crack classification and does not contribute to the classification. Yet, the image area which the zero-padded regions share with and contain crack features are positively classified. For second and later iterations, the need of zero-padding is required only if the crack is extended to image border. Otherwise, the zero-padded region shown in Figure 34 corresponds to the neighboring regions. After the classification, the regions are labeled with respect to the prediction and upsampled with the factor of two for dimensional conformance between quadtree axis and control axis. The results are merged with OR logical gate. If a region is classified as crack by any of the instances, the result is stored as positive.

Idea of two shifted meta axes method effectively eliminates to problem of objects corresponding to the region boundaries and thus loss of object connectedness. Exemplary implementation is illustrated in Figure 35. Two regions residing at the left of top image are both classified as non-crack regions due to corresponding region border. Yet, these regions are positively classified as crack regions by the control meta axis and labeled as positive for later iterations.

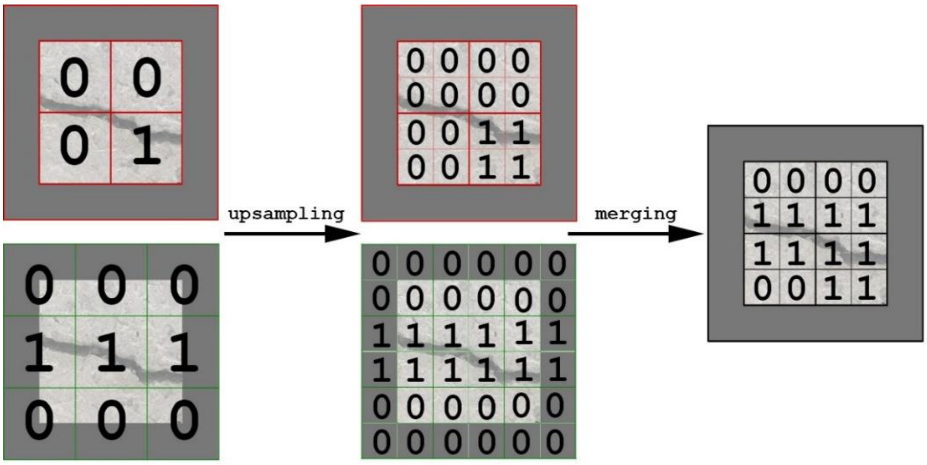


Figure 35. Two axes system method on an object corresponding to region boundary (drawn by author)

Region scoring: Region scoring is based on the idea of conservation of confidence scores as well as quadtree decomposition conditions. The aim of the region scoring method is to keep track of previous iterations. At each iteration, as the dimension of subdivided quadrant is reduced by the factor of two, the results are upsampled with the same factor to match with previous iteration. Then the scores are summed up to cumulatively store the how many times a region is positively classified as crack and if possible the confidence scores of these predictions are also stored. When the algorithm is terminated, the resultant score is divided by the total iteration number to normalize scores. As a result, region scores reflecting its decomposition history is obtained for each region. The calculation of final scores is as below:

$$R_{xy} = \frac{\sum_{i=1}^{k} P_{xy,i} * S_{xy,i}}{k}$$

Where *R* is region score, *S* is confidence score, *P* is binary prediction value, *i* is number of iteration, *xy* denotes the pixel/region location and *k* is the number of total iterations. *S* and *P* are provided by CNN and taken without subjecting any change. Confidence score resides between 0 and 1 while binary prediction value is either 0 or 1 denoting presence of crack. For multiclass problems, such as detection multiple defects as well as cracks, each class is treated separately as a binary problem of presence. Region scoring method aggressively penalizes regions which are not classified as cracks; thus quickly refines object boundaries. On the other hand, as the region scoring method preserves region decomposition history, problems caused by the excessive focusing resulting in objects losing their features are resolved and such regions still contribute to the probability distribution with high scores. As a result, region scoring outputs a probability map between 0 and 1.

Exemplary implementation of region scoring is illustrated in Figure 36. Top left image shows region scores overlaid on original image and used as an alpha map. Low scoring regions are more transparent than high scoring regions. Top right image shows the region scoring output with relative heat map denoting region scores. Bottom images are binary and heat map illustrations of region scoring by applying threshold with the value of 0,5 which the probability of presence is higher than the probability of absence.

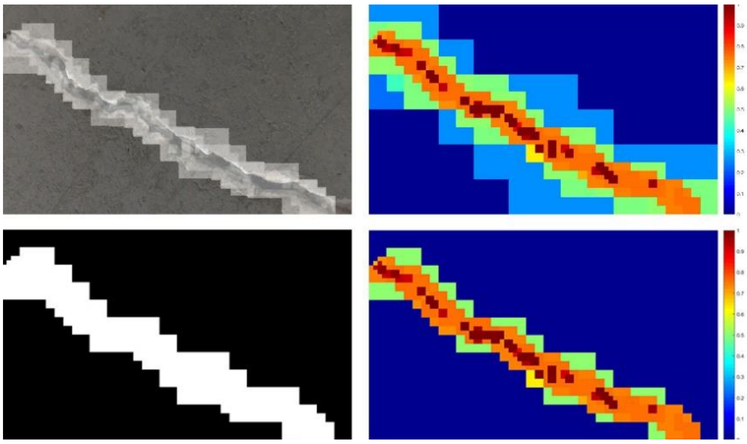


Figure 36. Region scores overlaid on crack image (top left), region scores (top right), Segmentation image based on quadtree division (bottom left), Region scores with 0.5 threshold (bottom right) (drawn by author)

3D Interpolation of region scores: In order to refine object boundaries, the problem is treated as a probability distribution in 3D space with width, height and region score dimensions. In other words, probability map with low resolution is transferred to 3D space to increase the resolution of probability map. The coordinates of image patch center points created by quadtree decomposition are extracted and their z-axis values are obtained from their respective probability obtained by region scoring. Then data points are connected to the neighboring points by linear interpolation to obtain a probability surface. Higher order interpolation techniques are not preferred due unnecessary smoothing of the surface resulting in loss of precision. An exemplary 3D interpolation of region scores to obtain probability distribution in shown in Figure 37.

Once the surface function is obtained, the probability values for each pixel is obtained by using the probability function. After obtaining per-pixel probability values, the problem is converted back to a 2D space to conduct semantic segmentation. The image is applied threshold of 0,5 where presence exceeds probability of absence. Figure 38 shows the stages of obtaining semantic segmentation results.

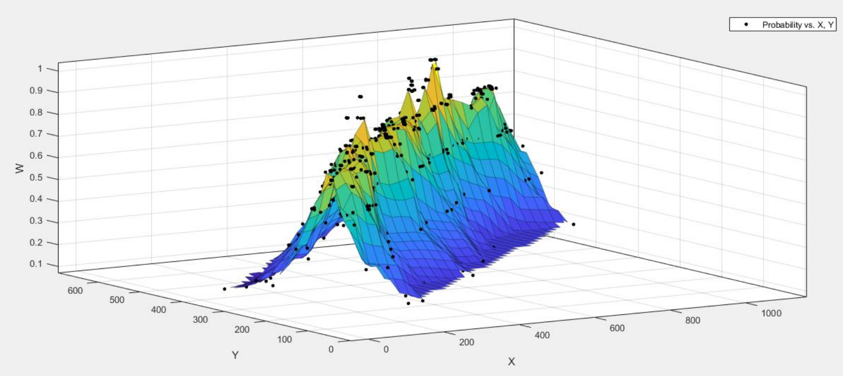


Figure 37. 3D Reconstruction and linear interpolation of pixel probabilities to obtain probability surface (drawn by author)

Apart from obtaining probability distribution of the image, 3D interpolation of region scores also serves for resolving the issue of jaggedness of outlines and provides semantic segmentation results with more precise boundaries.

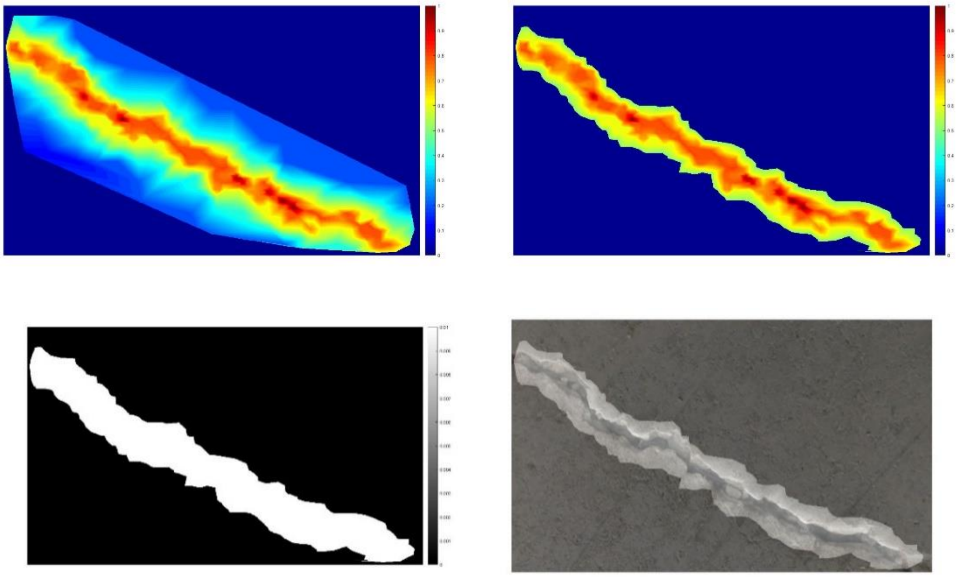


Figure 38. Probability map obtained by linear interpolation (top left), probability map obtained by 0,5 threshold (top right), semantic segmentation result (bottom left), image segmentation overlaid on original image (bottom right) (drawn by author)

To sum up, detailed QuadP flow chart showing how CNN, quadtree division algorithm, 2 meta axes, region scoring and 3D interpolation steps come together is shown in Figure 39.

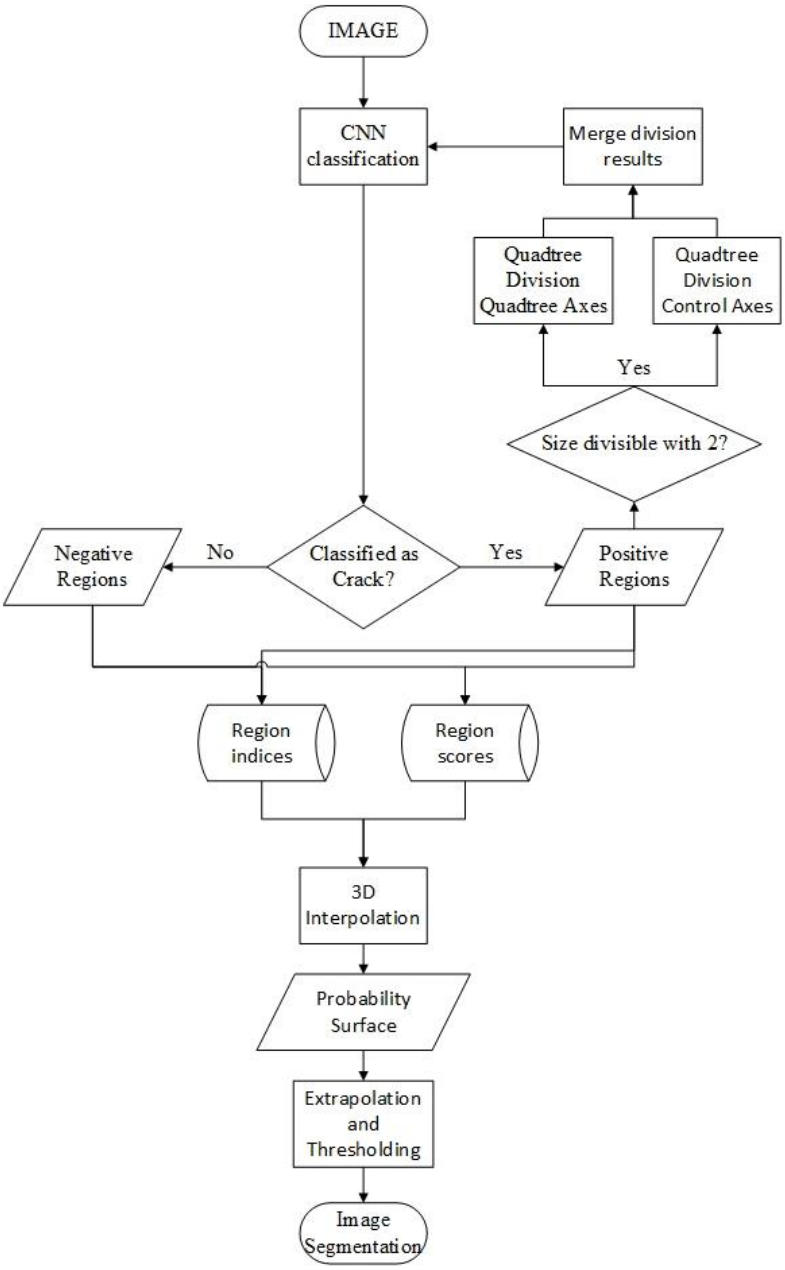


Figure 39. QuadP flowchart (drawn by author)

5.4 Performance inspection of QuadP

In order to test the performance of QuadP, semantic segmentation task is conducted and the performance of QuadP is compared with two highly acknowledged semantic segmentation methods, namely Fully Convolutional Networks (FCN) and SegNet. In order to avoid bias which may be caused because of the classification stage of the semantic segmentation, experiments are conducted with VGG16 pretrained network which is natively utilized in FCN and SegNet methods. Deconvolutional Networks are not included into comparison as VGG16 version of SegNet is the improved version of Deconvolutional Networks.

Both FCN and SegNet are end-to end trained with 6775 crack image samples with 500x500 pixel dimension together with the ground truth maps. The ground truth images are handcrafted by pixel scale painting. On the other hand, for experimentation with QuadP, VGG16 network trained with 1,75K image dataset, which contains 875 positive and 875 negative samples with 224x224 pixel dimension, is utilized. For validation, 376 images which are equally distributed to binary classes are used. Mentioned VGG16 network is also one of the best scoring network among the network compared in Chapter 4 and have a mean CwA score of %96,375 for binary classification of crack images. On the other hand, it should be noted that the performance of QuadP is bounded with the performance of VGG16 network, which are %99.2, %90.45, %97.79 and %81,75 accuracy scores and %99.8, %91.3, %98.1 and %96.3 CwA scores for test cases 1,2,3 and 4 respectively. Multiple VGG16 networks are intentionally not selected to observe the adaptability of a single network on multiple semantic segmentation tasks.

For testing the performances of three approaches, the approach utilized in Chapter 4 for performance comparison of pretrained network is chosen. Total of 491 full resolution images are used throughout the experiments. The test cases are explained in detail in Chapter 4. The distribution of test images per case is as shown below:

Table 11. Test cases and dataset sizes for semantic segmentation evaluation

Case	**Number of Images**
Building – concrete	458
Pavement – concrete	16
Building – concrete with varying conditions	9
Building - brickwork	8

5.4.1 Evaluation Metrics

The performances of three methods are evaluated with respect to two set of metrics, namely semantic segmentation metrics and crack based metrics. First set of metrics is based on semantic segmentation performance and consists five highly acknowledged metrics. These metrics are as below and calculated both globally and per class:

- *Global Accuracy:* Global accuracy measures the percentage of correctly classified pixels without consideration of classes. The formula of global accuracy is as below:

  $$Global\ Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

  Where TP, FP, TN and FN stand for respectively True Positive, False Positive, True Negative and False Negative.
- *Mean Accuracy (Accuracy):* Mean accuracy measures the average of percentages of correctly classified pixels per pixel. The formula is as below:

  $$Mean\ Accuracy = \frac{TP}{TP + FN}$$

  Accuracy metric is the calculation of the score per class.
- *Intersection over Union (IoU):* IoU measures the ratio of correctly classified pixels over sum of number of ground truth pixels and predicted pixels per class. Formula is as below:

  $$IoU = \frac{TP}{TP + FP + FN}$$

  While calculated IoU score per class, average of IoU score is taken for global calculation.

- *Weighted IoU:* Weighted IoU measures average of IoU weighted by the number of pixels of corresponding class
- *Mean Boundary F1 (BF) Score:* BF score is the extension of F-score for semantic segmentation evaluation. The formula is shared in Chapter 4. Mean BF score measures how well the predicted outline conforms with ground truth outline. For global evaluations average of BF scores are calculated.

Second set of metrics are proposed within the scope of this study with the aim of evaluating crack based on important features. These metrics are defined as below:

- *Number of Objects:* Number of objects calculates the number of individual regions for crack class. For connectedness 4-connected neighborhood of the regions are inspected. It is a measure of how well the object connectedness is preserved.
- *Mean Orientation:* Mean orientation measures the major axis orientation of individual crack regions and takes the average weighted with the area of the regions. It is a measure of influence of falsely classified regions on the output result.

A metric for pixel based crack width is not proposed as semantic segmentation metrics already cover how well the contours are matching with BF-Score.

5.4.2 Results and Discussions

The performances of methods are inspected with respect to two set of metrics on 4 test cases. The results of global and class based semantic segmentation metrics for 4 test cases are shared in Table 12 and Table 13.

Table 12. Global semantic segmentation results for 4 test cases

Case	**Method**	**Global Accuracy**	**Mean Accuracy**	**IoU**	**Weighted IoU**	**Mean BF**
Test1	**FCN**	**0.9915**	0.7410	0.7347	**0.9831**	**0.9605**
	SegNet	0.9904	0.8444	**0.7646**	**0.9829**	0.9104
	QuadP	0.9809	**0.9011**	0.6954	0.9714	0.8979
Test2	**FCN**	**0.9271**	0.6989	**0.6131**	**0.8793**	**0.4891**
	SegNet	0.8351	**0.8915**	0.5600	0.7844	0.2294
	QuadP	0.8914	0.7466	0.5828	0.8423	**0.4831**
Test3	**FCN**	**0.9855**	0.8007	**0.6816**	**0.9766**	**0.7660**
	SegNet	0.8948	**0.9115**	0.5031	0.8814	0.4716
	QuadP	0.9481	0.8536	0.5605	0.9364	0.5439
Test4	**FCN**	**0.9475**	0.7823	**0.5534**	**0.9341**	0.4436
	SegNet	0.4575	0.6941	0.2382	0.4421	0.1675
	QuadP	**0.9456**	**0.8334**	**0.5610**	**0.9324**	**0.5202**

When results of global semantic segmentation results are inspected, it is seen that FCN has the highest score for global accuracy for all cases while QuadP has comparable scores with FCN. For test cases 2, 3 and 4 where the illumination conditions, camera-surface orientations and materials change, performance of SegNet significantly drops, especially in test 4. On the other hand, for mean accuracy which represents the average of class accuracies, results of SegNet is higher than FCN and QuadP. High score is SegNet is due to the accuracy of detecting cracks but also being susceptible to noise resulting in falsely classified background as cracks. IoU and Weighted IoU score of FCN and QuadP are comparable and observed to be higher than SegNet for test cases 2, 3 and 4. For test case 1, even though QuadP scored the lowest, the result is comparable to counterparts. For BF score which evaluates how well the object contours match, FCN has the highest score for test 1 and test 3 while QuadP scored higher than FCN in test 4. Both have similar results in test case 2. To sum up, QuadP achieved comparable scores with highly acknowledge methods, even surpassing in challenging test cases. In order to obtain more information about the performances of methods, class based metrics are inspected.

Table 13. Class based semantic segmentation results for 4 test cases

Case	**Method**	**Class**	**Accuracy**	**IoU**	**Mean BF**
Test 1	**FCN**	**Crack**	0.4803	0.4779	**0.9519**
		Background	**0.9999**	**0.9914**	**0.9691**
	SegNet	**Crack**	0.6936	**0.5389**	0.8960
		Background	0.9953	0.9903	0.9248
	QuadP	**Crack**	**0.8187**	0.4101	0.8740
		Background	0.9836	0.9806	0.9219
Test 2	**FCN**	**Crack**	0.4318	**0.3014**	**0.4395**
		Background	0.9660	**0.9247**	0.5388
	SegNet	**Crack**	**0.9557**	0.2973	0.201
		Background	0.8254	0.8227	0.2578
	QuadP	**Crack**	0.5771	0.279	0.3763
		Background	**0.9161**	0.8866	**0.5898**
Test 3	**FCN**	**Crack**	0.6104	**0.3778**	**0.7090**
		Background	**0.9910**	**0.9854**	**0.8230**
	SegNet	**Crack**	**0.9286**	0.1128	0.4241
		Background	0.8943	0.8934	0.5191
	QuadP	**Crack**	0.7563	0.1734	0.3818
		Background	0.9509	0.9475	0.7061
Test4	**FCN**	**Crack**	0.6116	0.1597	**0.3637**
		Background	**0.9531**	**0.9470**	0.5234
	SegNet	**Crack**	**0.9388**	0.0275	0.1046
		Background	0.4495	0.449	0.2304
	QuadP	**Crack**	0.7174	**0.177**	0.3574
		Background	0.9494	**0.9449**	**0.683**

For class based semantic segmentation evaluation, QuadP achieved promising results by achieving the highest accuracy for crack class in test 1, background class in test2. In general, FCN scored the highest background accuracy for all test cases while SegNet achieved high scores for crack accuracy for test cases 2, 3 and 4. However, class based accuracy can be misleading as only correctly classified pixels over ground truth pixels are calculated. Hence, if all the image is classified as crack by falsely classifying background, the crack accuracy would be 1. For this purpose, class based accuracies are evaluated together with IoU and Mean BF scores. For IoU scores, FCN has the highest scores for test case 1, 2 and 3 while for the most challenging task which deals with segmentation of brickwork surfaces, QuadP achieved the highest score. For other cases, QuadP also scored comparable to counterparts. When mean BF scores are inspected, a similar result is observed. To sum up, FCN achieved to better than SegNet and QuadP for semantic segmentation metrics with close margin with QuadP. QuadP managed to score close scores to FCN, even surpassing SegNet and FCN in some of the challenging test cases.

However, the outputs of these methods are only valuable if the results are adequate for crack measurements. For this purpose, second set of metrics are analyzed and summary of the analysis is shared in Table 14.

Table 14. Number of objects and mean orientaion scores for 4 test cases

		Number Objects (percentage error)	Number of Objects (count)	Mean Orientation (percentage error)
Test 1	**FCN**	22.852	18.6x	0.076
	SegNet	727.002	536.8x	0.108
	QuadP	**4.685**	**4.3x**	**0.053**
Test 2	**FCN**	252.261	187.2x	**0.177**
	SegNet	6216.913	4906.9x	0.180
	QuadP	**7.914**	**6.2x**	0.293
Test 3	**FCN**	186.751	49.1x	0.131
	SegNet	5510.248	2751.1x	0.149
	QuadP	**9.974**	**4x**	**0.127**
Test 4	**FCN**	257.335	69.9x	0.328
	SegNet	10351.102	3470.8x	0.597
	QuadP	**12.234**	**3.45x**	**0.177**

When number of objects and mean orientation scores are inspected, it is seen that FCN and SegNet fails to preserve object connectedness in cases which the object classes have low features which can be easily confused with background noise. While QuadP also suffers from losing object connectedness and prediction of more cracks than the truth, the results of FCN and SegNet are considerably higher than QuadP and ground truth which risks conduction of measurement with respect to semantic segmentation results. For mean orientation scores, QuadP scored the best by getting the lowest percentage error in three of four test cases with considerable gap between counterparts. For test case 2, FCN scored better than QuadP. When the performances of networks for classification task are inspected, VGG16 network scored its considerably low for the test case 2 with %90,45 accuracy and %91.3 CwA for the utilized network for the semantic segmentation tests. Hence, the performance of QuadP is also bounded by the %90,45 performance of VGG16 network utilized.

When the results are observed visually, the rationale of performance scores both for semantic segmentation metrics and crack based metrics are evident. A sample comparison of original image, ground truth and results of QuadP, FCN and SegNet is shared in Figure 40Figure 39.

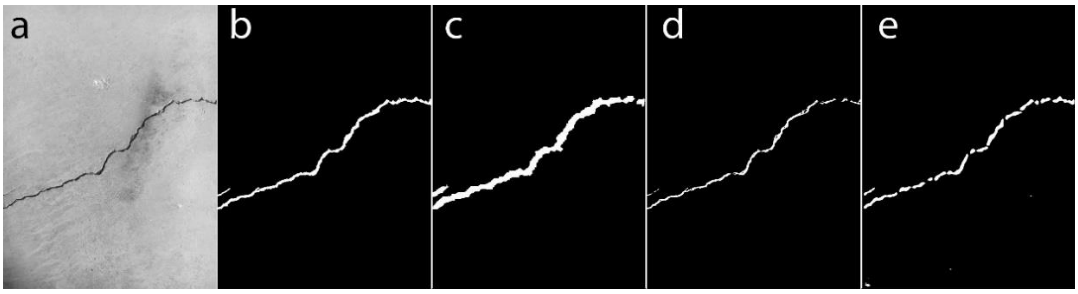


Figure 40.Comparison of network outputs versus original image and ground truth. a) Input image, b) manually drawn ground truth, c) QuadP result, d) FCN result, e) SegNet result (drawn by author)

As can be seen from the Figure 40, FCN and SegNet have finer outlines with respect to QuadP but crack connectedness is lost in many regions. While QuadP have tendency to overestimate crack region, FCN underestimates the crack width. On the other hand, SegNet has multiple segmentations and false classified pixels across the image canvas.

The proposed method within the scope of this study is believed to fulfill the objectives mentioned previously as being precise, flexible, lightweight and robust. QuadP achieved considerable well in crack based metrics while having comparable results in semantic segmentation metrics. The scores of QuadP is doubtlessly effected with the performance of VGG16 network. However, it is believed that the performance can be improved when QuadP method is utilized in conjunction with better performing networks or classification algorithms due its flexible nature.

As a result, Chapter 5 explains semantic segmentation process with the utilization of CNN algorithms. Thus, it is aimed to provide in depth perception of how CNN's operate by means of visualization of regions effecting the classification process. In addition, Chapter 5 constitutes examples of how the precision can be enhanced by means of utilizing various algorithms in conjunction and how case specific evaluation metrics can be devised and is crucial in determining the performance of CNN algorithms.

CHAPTER 6

CONCLUSION

Machine learning approaches and deep learning algorithms gained importance in the last two decades due to the availability of accumulated vast amount of data. When the contemporary technology and the means of data collection are taken into account, it is possible to foresee that the Big Data will continue to grow apace in terms of quantity and complexity. Correspondingly, the machine learning approaches and raw data processing algorithms, either deep learning or any other algorithm to be developed in future, will be prominent in the future. In order to survive in the data deluge and appraise useful information embedded in the mass of data, one needs to be able to have an insight on what this mass embodies, which part is relevant to the task and capability of formulating the task as a problem of learning from data. When the applicable fields of DL algorithms are considered, available data comprises more potentials than what can be achieved by only focusing on training phase, especially in cases which the subjective interpretation and personal taste are necessitated. In this context, data design term is introduced within the scope of this thesis as an instrument to control, collaborate with and evaluate the data and the algorithms to reach desired outcome.

6.1 General Discussions

Data design, which is defined and explained in depth in chapter 3, covers data related choices and their impacts in the course of utilizing deep learning or any other raw data processing framework. These choices include but not limited to data selection, reformulation of the task in terms of selected data, establishing the compatibility between algorithm and data, and devising task specific evaluation/assessment metrics.

Correspondingly, this study is constructed on the hypothesis of data design is one the most important act effecting the performance of raw data processing (i.e. deep learning) algorithms, and explores the questions of “how does data design influence output and evaluation of DL algorithm?”, “how can metrics for the evaluation of the results be determined?”, “is it possible to decide on optimal values for number and quality of data to guide data designers?” and “what is the relationship between data design and the structure of DL framework?”. The outline of the thesis in relation with the research questions, objectives and contributions are highlighted in Table 15. Accordingly, the present study aims to achieve the research objectives in four chapters.

Chapter 2 inspects how the data is incorporated for problem solving in traditional machine learning and deep learning algorithms. Traditional ML and DL frameworks pose considerable differences even though the mathematics behind the algorithms are almost the same. The ways of handling problems with these two approaches necessitate different strategies due to the nature of data provided to each framework. While traditional ML algorithms necessitate explicit definition of the features relevant the task, DL algorithms conduct feature extraction within the framework. Hence, users are obliged to redefine the problem regarding the available and relevant data instead of an accustomed way of problem definition through constraints and logical presumptions. Breaking the routine way of problem definition for machine learning implementations provides an opportunity for working on problems which the relevant features cannot be decoded such as problems requiring intuitive decisions. On the other hand, data remains as the only instrument to define the problem and desired outcome, putting emphasis on both data selection and evaluation of the results. Both of these actions involve subjective interpretation to an extent, regardless of the complexity and nature of the problem. Yet, as the subjectivity involved in the data increases to reflect personal preferences, results can only be evaluated by the user who provides the data.

Table 15. Research questions, objectives and contributions of the study

Research Questions	Chapter 2	Chapter 3	Chapter 4	Chapter 5
How data design influences output and evaluation	Comparison of ML and DL frameworks		Through crack detection as an example of image classification	
How can metrics for the evaluation of the results be determined			Confidence weighted accuracy	Crack metrics (orientation, number of cracks)
Is it possible to achieve optimal values for number and quality of data to guide data designers			Analysis of CNN parameters vs performance	
What is the relationship between data design and the structure of framework		Explanation of CNN architecture in relation with data design	Through crack detection as an example of image classification	Through semantic segmentation of crack images
Objectives				
To exemplify data design process by highlighting the constituents of the action			Through crack detection as an example of image classification	
To demonstrate how deep learning algorithms can be utilized in architecture	Previous examples of crack detection with CNN		Through crack detection as an example of image classification	Through semantic segmentation of crack images
To explore evaluation metrics of the output of the frameworks			Accuracy, F-score, Confidence weighted accuracy	Semantic segmentation metrics, Crack metrics
To identify case specific limitations and potentials of raw data processing frameworks	Challenges of crack detection from visual data		Through crack detection as an example of image classification	Through semantic segmentation of crack images
To explicate the working principles of DL to provide AI literacy				
To demonstrate how DL frameworks can be used in conjunction with other algorithms		Explanation of CNN architecture		QuadP
To provide a holistic understanding of data design	Each chapter contributes to the holistic understanding of data design			
Contributions		Data design term is introduced and explained in detail	Confidence weighted accuracy	Crack metrics
			Optimal value search	QuadP

Being able to operate on raw data without explicitly defining the relevant features to the task has significant potentials, especially for disciplines such as architecture which comprises problems with varying complexities, scales and requirement of subjective interpretation. Especially, convolutional neural networks become prominent among DL algorithms for implementations in architecture due to the fact that CNN’s are capable of processing multidimensional raw data while conserving the spatial relations which is crucial for architecture.

For elaboration of the discussion, crack detection in buildings is chosen as the case study with the aim of exemplifying the DL implementations. As crack detection is a straightforward task for human perception, objective evaluation of the results is possible. Yet, it is a challenging problem for DL algorithms at the same time, as cracks have limited visual features to be discriminated from non-crack objects such as shadows, foreign objects and irregularities on the surface. A comprehensive literature survey is conducted on machine learning implementations for crack detection including deep learning examples. As studies on crack detection in buildings are limited, the survey is extended to include infrastructures artifacts. Reported performances are compared in relation with how the data is handled in each study.

As a result of the literature survey, it is observed that performance of CNN applications surpasses the traditional ML applications and number of studies employing CNN has considerably increased since 2015. It should be noted that the inspected studies focus on different application areas such as sewer pipes or buildings and the data employed in these studies are exclusive to the studies. Hence, the performance comparison is based on the reported results and does not reflect the superiority of any algorithm or approach but instead is perceived as a benchmark for the implementation phase of the study. One of the most significant findings of the study is that the data selection in terms of site of acquisition, quality and representativeness of data is implicitly referred in the applications and reasons for providing that particular dataset remains obscure. For this reason, it is not possible to trace the influence of data selection to the performance of the implementations.

Data design is introduced and discussed in depth as the major contribution of this thesis in Chapter 3. Data design aims to construct a frame for series of actions taken in the course of employing DL algorithms for custom problems. In that sense, data design is neither solely selection of data nor the task specific evaluation of the outcome but instead an end to end approach to process. For this purpose, the process is divided into three phases as pre DL, DL and post DL phases and decisions taken at each phase are determined together with the interrelations respectively. Study attaches utmost importance to pre and post DL phases as the relationship between DL algorithms and task is constructed in these phases by means of reformulating the task as a problem of learning from data and getting sensible inferences from the results DL algorithm outputs by means of using or devising task specific evaluation methods.

Pre DL phase is composed of four tasks which are reformulation of the problem, determining the data representing the case and the solution, determining sutiable DL framework and crafting the data without altering the information with respect to the selected DL framework. The decisions taken at this phase directly effects DL and post DL phases in terms of configuring the DL structure, determining the DL parameters, determining evaluation method and case specific metrics. As illustrated and discussed in Chapter 3, the decisions taken at each phase are not in a linear order. Instead they are interrelated effecting each other and require holistic understanding of the data design process, field expertise and DL literacy. Chapter 3 also contributes to DL literacy by explaining the working principles of convolutional neural networks as an exemplary DL framework apt for applications in architecture. While scrutinizing mathematics and structure of CNN, the relevance of data design is also investigated. As data design is context sensitive for which the data and decisions are task specific, the discussion is pursued in Chapters 4 and 5 with the case of crack detection in buildings.

Chapter 4 and Chapter 5 exemplifies data design process and demonstrates how DL algorithms can be utilized in architecture through crack detection in buildings. In addition, case study is used as a facilitator to investigate and identify the task specificity of data design in relation with its potentials and limitations. While Chapter

4 handles crack detection in image level, i.e. presence of crack in an image; Chapter 5 focuses on pixel level predictions for the determination of crack region in an image.

One of the contributions of this study is the comprehensive analysis of parameters effecting the performance of convolutional neural networks in crack detection task. For this purpose, 7 highly acknowledged pretrained networks are employed and a multidimensional analysis regarding training dataset size, number of training epochs, number of convolution layers and number of learnable parameters is conducted in Chapter 4. By this means, it is aimed to inspect how each of these parameters effect the training and testing performance so that further studies can employ this foreknowledge to start with wise guess for construction and/or implementation of CNN networks on crack detection. It is observed even though it is not possible to obtain a generic recipe for the data design process, it is possible to achieve optimal ranges specific to tasks.

In the course of obtaining the most suitable network and parameters for crack detection, it is observed that pretrained networks converge rapidly and obtain high accuracy results. On the other hand, for crack detection task, confidence of the prediction is as valuable as the prediction itself revealing valuable information on how the algorithm behaves in the course of making predictions. For this purpose, confidence weighted accuracy (CwA) is proposed to differentiate between multiple networks having similar accuracies with respect to their confidences. Thus, introduction of CwA is regarded as a contribution to the field and an example of devising evaluation metrics for concerned task.

Chapter 4 also demonstrates the power of CNN’s in the course of obtaining generic solutions. Even though the networks are only trained with images of crack on concrete surfaces, the resulting framework performed considerably successfully on brickwork and pavement images. Hence, it is possible to conclude that the trained networks obtained generic features defining the cracks and are applicable to other materials and cases even though they are not trained with the data of these materials.

Similarly, Chapter 5 focuses on pixel level prediction of crack regions by means of semantic segmentation method. In that sense, the thesis constitutes the first study which conducts the semantic segmentation of cracks on buildings as application area to be evaluated against semantic segmentation metrics. Even though semantic segmentation metrics provide valuable information based on pixel predictions, it is believed that each case require additional metrics in order to evaluate the compatibility of results with the desired outcome. For semantic segmentation of cracks, the outputs are expected to reveal crack properties for further analysis and inference on the severity of cracks. In that sense, conservation of crack connectedness and making predictions in correct orientation provide valuable information for the assessment of structural performance of buildings. For this purpose, case specific metrics (i.e. crack metrics) are proposed to evaluate the segmentation results besides semantic segmentation metrics. Similar to confidence weighted accuracy in Chapter 4, crack metrics are also contributions to the field and exemplifies of case specific metrics for the evaluation of the DL framework.

In order to conduct crack segmentation, two highly acknowledged frameworks; namely Fully Convolutional Networks and SegNet, are utilized as well as QuadP framework which is proposed within the scope of this thesis. It is observed that both FCN and SegNet are prone to lose object connectivity even though obtaining high scores in segmentation metrics.

One of the biggest contribution of the study can be evaluated as the introduction of a novel semantic segmentation method which combines quadtree algorithm and CNN. Proposed method is novel in terms of linking separate algorithms to benefit from their strong properties and by this way to develop a lightweight, flexible and yet precise and robust method for crack detection task. In that sense, QuadP embodies novel approaches to semantic segmentation. These approaches can be summarized as utilization of two meta axes for conservation of neighboring regions for quadtree division algorithms and handling probability distribution in 3D space to obtain a probability surface obtained from binary classification results. Achieving comparable results in semantic segmentation metrics and having better performance in crack based metrics without the requirement of ground truth images is a considerable success. Even though the study is proposed in consideration with crack detection, it is believed that QuadP will have similar performance for semantic segmentation of subject which have only low level discriminative features such as blood veins or tree branches.

In the light of the findings discussed above, it is confirmed that how the problem is reflected to deep learning algorithms and how the data is designated have an undeniable role in the success of the implementation. In order to get the best out of DL algorithms, the process of implementation must be perceived with a holistic view which is the core of data design.

6.2 Recommendations for Future Work

Data design, which is introduced within the scope of this thesis, is exemplified with a well-defined task in order to validate the postulate through quantifiable analysis. Yet, the importance of data design becomes more significant as the level of subjectivity involved in the task. For prospective studies, it is recommended to investigate the methods for evaluation of the results, especially for the tasks involving personal preferences. In that sense, documentation and publication of case specific DL evaluation metrics have utmost importance for guiding new researchers and for establishing a common ground for case specific implementations. Similarly, studies on case specific optimal ranges for parameters influencing the performance of DL algorithms are encouraged in order to provide foreknowledge to researchers.

This study scratches the surface of possible implementations in architecture. Although handling highly complex and multidimensional architecture problems is not possible due to hardware limitations and lack of data, it is believed that each study focusing on partial implementations will provide priceless know-how and experience for future researchers. It is evident that hardware constraints will be resolved in near future. Yet, data acquisition for problems of architecture may not be possible for individual researcher which puts emphasis on publicly available data.

In that sense, one of the challenges encountered in the process of conducting this study is the lack of standardized data. Absence of such data not only causes incompatibility of the studies, but also obliges researchers to spend invaluable time and effort for collection of data. It is evident that establishing standardized datasets targeting custom tasks will accelerate the developments in deep learning and also increase effective use of DL algorithms in various fields. In addition, it will be possible to compare DL algorithms without the influence of data selection for specific cases.

Within the scope of this thesis, operability of several algorithms in conjunction for precision enhancements is demonstrated with QuadP method. Yet, QuadP resides as an exemplary study among many possible algorithm combinations. As each case brings its own potentials and challenges, deliberate analysis is advised for searching for algorithms to fuse with ML or DL algorithms. In this respect, it would be possible to overcome the limitations of DL algorithms and achieve more precise or preferable results depending on the task.

REFERENCES

Alpaydin, E. (2010) *Introduction to Machine Learning*. MIT Press. doi: 10.1016/j.neuroimage.2010.11.004.

Alsheikh, M. A., Niyato, D., Lin, S., Tan, H.-P. and Han, Z. (2016) ‘Mobile Big Data Analytics Using Deep Learning and Apache Spark’, *IEEE Network*, 30(3), pp. 22–29. doi: 10.1109/MNET.2016.7474340.

Badrinarayanan, V., Handa, A. and Cipolla, R. (2015) ‘SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling’. doi: 10.1103/PhysRevX.5.041024.

Badrinarayanan, V., Kendall, A. and Cipolla, R. (2015) ‘SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation’, pp. 1–14. doi: 10.1109/TPAMI.2016.2644615.

Boser, B. E., Guyon, I. M. and Vapnik, V. N. (1992) ‘A Training Algorithm for Optimal Margin Classiiers’, *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152. doi: 10.1.1.21.3818.

Canny, J. (1986) ‘A Computational Approach to Edge Detection’, in *IEEE Tranaction on Pattern Analysis and Machine Intelligence*. IEEE, pp. 679–697. doi: 10.1109/TPAMI.1986.4767851.

Canziani, A., Paszke, A. and Culurciello, E. (2016) ‘An Analysis of Deep Neural Network Models for Practical Applications’, pp. 1–7. Available at: http://arxiv.org/abs/1605.07678.

Cha, Y.-J., Choi, W. and Büyüköztürk, O. (2017) ‘Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks’, *Computer-Aided Civil and Infrastructure Engineering*, 32(5), pp. 361–378. doi: 10.1111/mice.12263.

Chatfield, K., Simonyan, K., Vedaldi, A. and Zisserman, A. (2014) ‘Return of the Devil in the Details: Delving Deep into Convolutional Nets’. doi: 10.5244/C.28.6.

Chen, C., Seff, A., A., K. and Xiao, J. (2015) ‘DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving’, in *ICCV ’15 Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV),* pp. 2722–2730. Available at: http://deepdriving.cs.princeton.edu/.

Collobert, R. and Weston, J. (2008) ‘A unified architecture for natural language processing: Deep neural networks with multitask learning’, *Proceedings of the 25th international conference on Machine learning,* pp. 160–167. doi: 10.1145/1390156.1390177.

Cover, T. and Hart, P. (1967) ‘Nearest neighbor pattern classification’, *IEEE Transactions on Information Theory,* 13(1), pp. 21–27. doi: 10.1109/TIT.1967.1053964.

Dozat, T. (2016) ‘Incorporating Nesterov Momentum into Adam’, *ICLR Workshop,* (1), pp. 2013–2016.

Duchi, J., Hazan, E. and Singer, Y. (2011) ‘Adaptive Subgradient Methods for Online Learning and Stochastic Optimization’, *Journal of Machine Learning Research,* 12, pp. 2121–2159. doi: 10.1109/CDC.2012.6426698.

Eisenbach, M., Stricker, R. and Debes, K. (2017) ‘Crack Detection with an Interactive and Adaptive Video Inspection System’, in *Arbeitsgruppentagung Infrastrukturmanagement*, pp. 94–103.

Entezari-Maleki, R., Rezaei, A. and Minaei-Bidgoli, B. (2009) ‘Comparison of Classification Methods Based on the Type of Attributes and Sample Size’, *Journal of Convergence Information Technology,* 4(3), pp. 94–102. doi: 10.4156/jcit.vol4.issue3.14.

Ersoz, A. B., Pekcan, O. and Teke, T. (2017) ‘Crack identification for rigid pavements using unmanned aerial vehicles’, *IOP Conference Series: Materials Science and Engineering*, 236(1). doi: 10.1088/1757-899X/236/1/012101.

Ghanghau, I. (2017) *Activation Functions in Neural Networks*. Available at: https://isaacchanghau.github.io/post/activation_functions/ (Accessed: 22 April 2018).

Gibert, X., Patel, V. M. and Chellappa, R. (2015) ‘Semantic Segmentation of Railway Track Images with Deep Convolutional Neural Networks’, pp. 621–625.

Gill, J. K. (2017) *Log Analytics With Deep Learning And Machine Learning, Xenonstack*. Available at: https://www.xenonstack.com/blog/data-science/log-analytics-with-deep-learning-and-machine-learning (Accessed: 22 April 2018).

Goertzel, B. (2015) ‘Are there deep reasons underlying the pathologies of today’s deep learning algorithms?’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9205, pp. 70–79. doi: 10.1007/978-3-319-21365-1_8.

Gonzalez, R. C., Woods, R. E. and Eddins, S. L. (2009) *Digital image processing*. 2nd edn. Gatesmark Publishing. doi: 10.1016/0734-189X(90)90171-Q.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014) ‘Generative Adversarial Networks’, pp. 1–9. doi: 10.1001/jamainternmed.2016.8245.

Gopalakrishnan, K., Khaitan, S. K., Choudhary, A. and Agrawal, A. (2017) ‘Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection’, *Construction and Building Materials*. Elsevier Ltd, 157(September), pp. 322–330. doi: 10.1016/j.conbuildmat.2017.09.110.

Haque, U. (2007) ‘The Architectural Relevance of Gordon Pask’, *Architectural Design*, 77(4), pp. 54–61. doi: 10.1002/ad.487.

He, K., Zhang, X., Ren, S. and Sun, J. (2016) ‘Deep Residual Learning for Image Recognition’, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. doi: 10.1109/CVPR.2016.90.

Huang, J., Lu, J. and Ling, C. X. (2003) ‘Comparing naive Bayes, decision trees, and SVM with AUC and accuracy’, *Third IEEE International Conference on Data Mining*, pp. 553–556. doi: 10.1109/ICDM.2003.1250975.

Hubel, D. and Wiesel, T. (1959) ‘Receptive fields of single neurones in the cat’s striate cortex’, *The Journal of physiology*, 148(3), pp. 574–591.

Kabir, S., Rivard, P. and Ballivy, G. (2008) ‘Neural-network-based damage classification of bridge infrastructure using texture analysis’, *Canadian Journal of Civil Engineering*, 35(3), pp. 258–267. doi: 10.1139/L07-105.

Karpathy, A. (2018) *CS231n Convolutional Neural Networks for Visual Recognition*. Available at: http://cs231n.github.io/convolutional-networks/ (Accessed: 27 April 2018).

Kingma, D. P. and Ba, J. (2014) ‘Adam: A Method for Stochastic Optimization’, pp. 1–15. doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503.

Koch, C., Georgieva, K., Kasireddy, V., Akinci, B. and Fieguth, P. (2015) ‘A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure’, *Advanced Engineering Informatics*. Elsevier Ltd, 29(2), pp. 196–210. doi: 10.1016/j.aei.2015.01.008.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012) ‘ImageNet Classification with Deep Convolutional Neural Networks’, *Advances In Neural Information Processing Systems*, pp. 1–9. doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007.

Küçüksubaşı, F. (2017) *An Integrated System Design for Building Inspection by Autonomous UAVs*. Middle East Technical Unviersity.

Lattanzi, D. and Miller, G. R. (2014) ‘Robust Automated Concrete Damage Detection Algorithms for Field Applications’, *Journal of Computing in Civil Engineering*, 28(April), p. 120917010504009. doi: 10.1061/(ASCE)CP.1943-5487.0000257.

Lecun, Y., Bengio, Y. and Hinton, G. (2015) ‘Deep learning’, *Nature*, 521(7553), pp. 436–444. doi: 10.1038/nature14539.

Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) ‘Gradient-Based Learning Applied to Document Recognition’, *Proceedings of the IEEE*, 86(11), pp. 2278–2323. doi: 10.1109/5.726791.

Lee, A. (2016) ‘The meaning of AlphaGo , the AI program that beat a Go champ’, *Maclean’s*, pp. 1–7. Available at: http://www.macleans.ca/society/science/the-meaning-of-alphago-the-ai-program-that-beat-a-go-champ/.

Li, G., Zhao, X., Du, K., Ru, F. and Zhang, Y. (2017) ‘Recognition and evaluation of bridge cracks with modified active contour model and greedy search-based support vector machine’, *Automation in Construction*. Elsevier B.V., 78, pp. 51–61.

Liu, L., Yan, R. J., Maruvanchery, V., Kayacan, E., Chen, I. M. and Tiong, L. K. (2017) ‘Transfer learning on convolutional activation feature as applied to a building quality assessment robot’, *International Journal of Advanced Robotic Systems*, 14(3), pp. 1–12. doi: 10.1177/1729881417712620.

Liu, Z., Suandi, S. A., Ohashi, T. and Ejima, T. (2002) ‘Tunnel crack detection and classification system based on image processing’, *Electronic Imaging 2002*, 4664, pp. 145–152.

Long, J., Shelhamer, E. and Darrell, T. (2014) ‘Fully Convolutional Networks for Semantic Segmentation’. doi: 10.1109/TPAMI.2016.2572683.

MatConvNet Team (2018) *MatConvNet*. Available at: http://www.vlfeat.org/matconvnet/ (Accessed: 29 April 2018).

Mathworks (2018) *MATLAB - Mathworks - MATLAB & Simulink*. Available at: https://www.mathworks.com/products/matlab.html (Accessed: 29 April 2018).

Mitchell, T. (1997) *Machine learning*. McGraw-Hill. doi: 10.1007/978-3-540-75488-6_2.

Moon, H. and Kim, J. (2011) ‘Inteligent Crack Detecting Algorithm On The Concrete Crack Image Using Neural network’, *Proceedings of the 28th ISARC, Seoul, Korea*, pp. 1461–1467. Available at: http://www.iaarc.org/publications/fulltext/P2-20.pdf.

Nesterov, Y. (1983) ‘A Method of Solving A Convex Programming Problem With Convergence rate O(1/k^2)’, *Soviet Mathematics Doklady*, pp. 372–376. Available at: http://www.core.ucl.ac.be/~nesterov/Research/Papers/DAN83.pdf.

NIST Big Data Public Working Group (2015) *NIST Special Publication 1500-1 - NIST*

Big Data Interoperability Framework: Volume 1, Definitions, NIST Special Publication. doi: http://dx.doi.org/10.6028/NIST.SP.1500-1.

Noh, H., Hong, S. and Han, B. (2015) ‘Learning deconvolution network for semantic segmentation’, *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter, pp. 1520–1528. doi: 10.1109/ICCV.2015.178.

Oxford Robotics Institute (2018) *The PASCAL Visual Object Classes Homepage*. Available at: http://host.robots.ox.ac.uk/pascal/VOC/ (Accessed: 29 April 2018).

Özgenel, Ç. F. (2018) ‘Concrete Crack Images for Classification’. Mendeley Data. doi: 10.17632/5y9wdsg2zt.1.

Pauly, L., Peel, H., Luo, S., Hogg, D. and Fuentes, R. (2017) ‘Deeper Networks for Pavement Crack Detection’, in *Proceedings of the 34th ISARC. 34th International Symposium in Automation and Robotics in Construction*. Taipei, pp. 479–485. doi: https://doi.org/10.22260/ISARC2017/0066.

Powers, D. M. W. (2015) *What the F-measure doesn’t measure: Features, Flaws, Fallacies and Fixes*. doi: KIT-14-001.

Ruder, S. (2016) ‘An overview of gradient descent optimization algorithms’, pp. 1–14. doi: 10.1111/j.0006-341X.1999.00591.x.

Samuel, A. L. (1959) ‘Some Studies in Machine Learning Using the Game of Checkers’, *IBM Journal of Research and Development*, 3(3), pp. 210–229.

Santur, Y., Karaköse, M. and Akın, E. (2016) ‘Random Forest Based Diagnosis Approach for Rail Fault Inspection in Railways’, in *National Conference on Electrical, Electronics and Biomedical Engineering (ELECO), 2016*. Bursa: IEEE, pp. 745–750.

Schmidhuber, J. (2015) ‘Deep Learning in neural networks: An overview’, *Neural Networks*. Elsevier Ltd, 61, pp. 85–117. doi: 10.1016/j.neunet.2014.09.003.

Schmugge, S. J., Nguyen, N. R., Thao, C., Lindberg, J., Grizzi, R., Joffe, C. and Shin, M. C. (2015) ‘Automatic detection of cracks during power plant inspection’, *Proceedings of the 3rd International Conference on Applied Robotics for the Power Industry, CARPI 2014*, (Figure 1). doi: 10.1109/CARPI.2014.7030042.

Shoham, Y., Perrault, R., Brynjolfsson, E., Clark, J. and LeGassick, C. (2017) ‘Artificial Intelligence Index, November 2017’, (November), p. 101. Available at: https://aiindex.org/.

Simonyan, K. and Zisserman, A. (2015) ‘Very Deep Convolutional Networks for Large-Scale Image Recognition’, in *International Conference on Learning Representations (ICRL)*, pp. 1–14. doi: 10.1016/j.infsof.2008.09.005.

Sinha, S. K. and Fieguth, P. W. (2006) ‘Neuro-fuzzy network for the classification of buried pipe defects’, *Automation in Construction*, 15(1), pp. 73–83. doi: 10.1016/j.autcon.2005.02.005.

Stanford Vision Lab (2018) *ImageNet*. Available at: http://www.image-net.org (Accessed: 29 April 2018).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) ‘Going deeper with convolutions’, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1–9. doi: 10.1109/CVPR.2015.7298594.

Turing, A. M. (1950) ‘Computing machinery and intelligence’, *Mind*, 49, pp. 433–460. doi: 10.1007/978-1-4020-6710-5_3.

Vapnik, V. and Lerner, A. (1963) ‘Pattern recognition using generalized portrait method’, *Automation and remote control*, 24(6), pp. 774–780. doi: citeulike-article-id:619639.

Wang, K. C. P., Zhang, A., Li, J. Q., Fei, Y., Chen, C. and Li, B. (2017) ‘Deep Learning for Asphalt Pavement Cracking Recognition Using Convolutional Neural Network’, in *Conference: International Conference on Highway Pavements and Airfield Technology 2017*, pp. 166–177.

Wilamowski, B. M., Wu, B. and Korniak, J. (2016) ‘Big Data an Deep Learning’, in *International Conference on Intelligent Engineering Systems (INES)*. Budapest: IEEE, pp. 11–16. doi: 10.1109/INES.2016.7555103.

Wu, S., Zhong, S. and Liu, Y. (2017) ‘Deep residual learning for image steganalysis’, *Multimedia Tools and Applications*, pp. 1–17. doi: 10.1007/s11042-017-4440-4.

Wu, W., Liu, Z. and He, Y. (2015) ‘Classification of defects with ensemble methods in the automated visual inspection of sewer pipes’, *Pattern Analysis and Applications*, 18(2), pp. 263–276. doi: 10.1007/s10044-013-0355-5.

Yakopcic, C., Alom, M. Z. and Taha, T. M. (2016) ‘Memristor crossbar deep network implementation based on a Convolutional neural network’, *Proceedings of the International Joint Conference on Neural Networks*, 2016–Octob, pp. 963–970. doi: 10.1109/IJCNN.2016.7727302.

Yang, M. D. and Su, T. C. (2008) ‘Automated diagnosis of sewer pipe defects based on machine learning approaches’, *Expert Systems with Applications*, 35(3), pp. 1327–1337. doi: 10.1016/j.eswa.2007.08.013.

Zeiler, M. D. (2012) ‘ADADELTA: An Adaptive Learning Rate Method’. Available at: http://arxiv.org/abs/1212.5701.

Zhang, L., Yang, F., Zhang, Y. D. and Zhu, Y. J. (2016) ‘Road Crack Detection Using Deep Convolutional Neural Network’, in *2016 IEEE International Conference on Image Processing (ICIP)*. doi: 10.1109/ICIP.2016.7533052.

Zhang, Q., Yang, L. T., Chen, Z. and Li, P. (2018) ‘A survey on deep learning for big data’, *Information Fusion*. Elsevier, 42(October 2017), pp. 146–157. doi: 10.1016/j.inffus.2017.10.006.

Zhang, W., Zhang, Z., Qi, D. and Liu, Y. (2014) ‘Automatic crack detection and classification method for subway tunnel safety monitoring’, *Sensors (Switzerland)*, 14(10), pp. 19307–19328. doi: 10.3390/s141019307.

Zou, Q., Cao, Y., Li, Q., Mao, Q. and Wang, S. (2012) ‘CrackTree: Automatic crack detection from pavement images’, *Pattern Recognition Letters*. Elsevier B.V., 33(3), pp. 227–238. doi: 10.1016/j.patrec.2011.11.004.

APPENDIX I

PRETRAINED NETWORK ARCHITECTURES

Table 16. AlexNet Architecture

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
conv1	Convolution	x0	x1	11x11x3x96	96
relu1	ReLu	x1	x2		
norm1	Local Response Normalization	x2	x3		
pool1	Max Pooling	x3	x4		
conv2	Convolution	x4	x5	5x5x48x255	256
relu2	ReLu	x5	x6		
norm2	Local Response Normalization	x6	x7		
pool2	Max Pooling	x7	x8		
conv3	Convolution	x8	x9	3x3x256x384	384
relu3	ReLu	x9	x10		
conv4	Convolution	x10	x11	3x3x192x384	384
relu4	ReLu	x11	x12		
conv5	Convolution	x12	x13	3x3x192x256	256
relu5	ReLu	x13	x14		
pool5	Max Pooling	x14	x15		
fc6	Convolution	x15	x16	6x6x256x4096	4096
relu6	ReLu	x16	x17		
fc7	Convolution	x17	x18	1x1x4096x4096	4096
relu7	ReLu	x18	x19		
fc8	Convolution	x19	x20	1x1x4096x1000	1000
prob	Softmax	x20	x21		

Table 17. VGG16 Architecture

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
conv1_1	Convolution	x0	x1	3x3x3x64	64
relu1_1	ReLu	x1	x2		
conv1_2	Convolution	x2	x3	3x3x64x64	64
relu1_2	ReLu	x3	x4		
pool1	Max Pooling	x4	x5		
conv2_1	Convolution	x5	x6	3x3x64x128	128
relu2_1	ReLu	x6	x7		
conv2_2	Convolution	x7	x8	3x3x128x128	128
relu2_2	ReLu	x8	x9		
pool2	Max Pooling	x9	x10		
conv3_1	Convolution	x10	x11	3x3x128x256	256
relu3_1	ReLu	x11	x12		
conv3_2	Convolution	x12	x13	3x3x256x256	256
relu3_2	ReLu	x13	x14		
conv3_3	Convolution	x14	x15	3x3x256x256	256
relu3_3	ReLu	x15	x16		
pool3	Max Pooling	x16	x17		
conv4_1	Convolution	x17	x18	3x3x256x512	512
relu4_1	ReLu	x18	x19		
conv4_2	Convolution	x19	x20	3x3x512x512	512
relu4_2	ReLu	x20	x21		
conv4_3	Convolution	x21	x22	3x3x512x512	512
relu4_3	ReLu	x22	x23		
pool4	Max Pooling	x23	x24		
conv5_1	Convolution	x24	x25	3x3x512x512	512
relu5_1	ReLu	x25	x26		
conv5_2	Convolution	x26	x27	3x3x512x512	512
relu5_2	ReLu	x27	x28		
conv5_3	Convolution	x28	x29	3x3x512x512	512
relu5_3	ReLu	x29	x30		
pool5	Max Pooling	x30	x31		
fc6	Convolution	x31	x32	7x7x512x4096	4096
relu6	ReLu	x32	x33		
fc7	Convolution	x33	x34	1x1x4096x4096	4096
relu7	ReLu	x34	x35		
fc8	Convolution	x35	x36	1x1x4096x1000	1000
prob	Softmax	x36	x37		

Table 18.VGG19 Architecture

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
conv1_1	Convolution	x0	x1	3x3x3x64	64
relu1_1	ReLu	x1	x2		
conv1_2	Convolution	x2	x3	3x3x64x64	64
relu1_2	ReLu	x3	x4		
pool1	Max Pooling	x4	x5		
conv2_1	Convolution	x5	x6	3x3x64x128	128
relu2_1	ReLu	x6	x7		
conv2_2	Convolution	x7	x8	3x3x128x128	128
relu2_2	ReLu	x8	x9		
pool2	Max Pooling	x9	x10		
conv3_1	Convolution	x10	x11	3x3x128x256	256
relu3_1	ReLu	x11	x12		
conv3_2	Convolution	x12	x13	3x3x256x256	256
relu3_2	ReLu	x13	x14		
conv3_3	Convolution	x14	x15	3x3x256x256	256
relu3_3	ReLu	x15	x16		
conv3_4	Convolution	x16	x17	3x3x256x256	256
relu3_4	ReLu	x17	x18		
pool3	Max Pooling	x18	x19		
conv4_1	Convolution	x19	x20	3x3x256x512	512
relu4_1	ReLu	x20	x21		
conv4_2	Convolution	x21	x22	3x3x512x512	512
relu4_2	ReLu	x22	x23		
conv4_3	Convolution	x23	x24	3x3x512x512	512
relu4_3	ReLu	x24	x25		
conv4_4	Convolution	x25	x26	3x3x512x512	512
relu4_4	ReLu	x26	x27		
pool4	Max Pooling	x27	x28		
conv5_1	Convolution	x28	x29	3x3x512x512	512
relu5_1	ReLu	x29	x30		
conv5_2	Convolution	x30	x31	3x3x512x512	512
relu5_2	ReLu	x31	x32		
conv5_3	Convolution	x32	x33	3x3x512x512	512
relu5_3	ReLu	x33	x34		
conv5_4	Convolution	x34	x35	3x3x512x512	512
relu5_4	ReLu	x35	x36		
pool5	Max Pooling	x36	x37		
fc6	Convolution	x37	x38	7x7x512x4096	4096
relu6	ReLu	x38	x39		
fc7	Convolution	x39	x40	1x1x4096x4096	4096
relu7	ReLu	x40	x41		
fc8	Convolution	x41	x42	1x1x4096x1000	1000
prob	Softmax	x42	x43		

Table 19. GoogleNet Architecture

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
conv1	Convolution	data	conv1	7x7x3x64	64
relu1	ReLu	conv1	conv1x		
pool1	Max Pooling	conv1x	pool1		
norm1	Local Response Normalization	pool1	norm1		
reduction2	Convolution	norm1	reduction2	1x1x64x64	64
relu_reduction2	ReLu	reduction2	reduction2x		
conv2	Convolution	reduction2x	conv2	3x3x64x192	192
relu2	ReLu	conv2	conv2x		
norm2	Local Response Normalization	conv2x	norm2		
pool2	Max Pooling	norm2	pool2		
icp1_reduction1	Convolution	pool2	icp1_reduction1	1x1x192x96	96
relu_icp1_reduction1	ReLu	icp1_reduction1	icp1_reduction1x		
icp1_reduction2	Convolution	pool2	icp1_reduction2	1x1x192x16	16
relu_icp1_reduction2	ReLu	icp1_reduction2	icp1_reduction2x		
icp1_pool	Max Pooling	pool2	icp1_pool		
icp1_out0	Convolution	pool2	icp1_out0	1x1x192x64	64
relu_icp1_out0	ReLu	icp1_out0	icp1_out0x		
icp1_out1	Convolution	icp1_reduction1x	icp1_out1	3x3x96x128	128
relu_icp1_out1	ReLu	icp1_out1	icp1_out1x		
icp1_out2	Convolution	icp1_reduction2x	icp1_out2	5x5x16x32	32
relu_icp1_out2	ReLu	icp1_out2	icp1_out2x		
icp1_out3	Convolution	icp1_pool	icp1_out3	1x1x192x32	32
relu_icp1_out3	ReLu	icp1_out3	icp1_out3x		
icp2_in	Concatenate	icp1_out0x, icp1_out1x, icp1_out2x, icp1_out3x	icp2_in		
icp2_reduction1	Convolution	icp2_in	icp2_reduction1	1x1x256x128	128
relu_icp2_reduction1	ReLu	icp2_reduction1	icp2_reduction1x		
icp2_reduction2	Convolution	icp2_in	icp2_reduction2	1x1x256x32	32
relu_icp2_reduction2	ReLu	icp2_reduction2	icp2_reduction2x		
icp2_pool	Max Pooling	icp2_in	icp2_pool		
icp2_out0	Convolution	icp2_in	icp2_out0	1x1x256x128	128
relu_icp2_out0	ReLu	icp2_out0	icp2_out0x		
icp2_out1	Convolution	icp2_reduction1x	icp2_out1	3x3x128x192	192
relu_icp2_out1	ReLu	icp2_out1	icp2_out1x		
icp2_out2	Convolution	icp2_reduction2x	icp2_out2	5x5x32x96	96
relu_icp2_out2	ReLu	icp2_out2	icp2_out2x		
icp2_out3	Convolution	icp2_pool	icp2_out3	1x1x256x64	64
relu_icp2_out3	ReLu	icp2_out3	icp2_out3x		
icp2_out	Concatenate	icp2_out0x, icp2_out1x, icp2_out2x, icp2_out3x	icp2_out		
icp3_in	Max Pooling	icp2_out	icp3_in		
icp3_reduction1	Convolution	icp3_in	icp3_reduction1	1x1x480x96	96
relu_icp3_reduction1	ReLu	icp3_reduction1	icp3_reduction1x		
icp3_reduction2	Convolution	icp3_in	icp3_reduction2	1x1x480x16	16
relu_icp3_reduction2	ReLu	icp3_reduction2	icp3_reduction2x		
icp3_pool	Max Pooling	icp3_in	icp3_pool		
icp3_out0	Convolution	icp3_in	icp3_out0	1x1x480x192	192
relu_icp3_out0	ReLu	icp3_out0	icp3_out0x		
icp3_out1	Convolution	icp3_reduction1x	icp3_out1	3x3x96x208	208
relu_icp3_out1	ReLu	icp3_out1	icp3_out1x		
icp3_out2	Convolution	icp3_reduction2x	icp3_out2	5x5x16x48	48
relu_icp3_out2	ReLu	icp3_out2	icp3_out2x		
icp3_out3	Convolution	icp3_pool	icp3_out3	1x1x480x64	64
relu_icp3_out3	ReLu	icp3_out3	icp3_out3x		

Table 19. GoogleNet Architecture (cont'd)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
icp3_out	Concatenate	icp3_out0x, icp3_out1x, icp3_out2x, icp3_out3x	icp3_out		
cls1_pool	Max Pooling	icp3_out	cls1_pool		
cls1_reduction	Convolution	cls1_pool	cls1_reduction	1x1x512x128	128
relu_cls1_reduction	ReLu	cls1_reduction	cls1_reductionx		
cls1_fc1	Convolution	cls1_reductionx	cls1_fc1	4x4x128x1024	1024
relu_cls1_fc1	ReLu	cls1_fc1	cls1_fc1x		
cls1_fc2	Convolution	cls1_fc1x	cls1_fc2	1x1x1024x1000	1000
icp4_reduction1	Convolution	icp3_out	icp4_reduction1	1x1x512x112	112
relu_icp4_reduction1	ReLu	icp4_reduction1	icp4_reduction1x		
icp4_reduction2	Convolution	icp3_out	icp4_reduction2	1x1x512x24	24
relu_icp4_reduction2	ReLu	icp4_reduction2	icp4_reduction2x		
icp4_pool	Max Pooling	icp3_out	icp4_pool		
icp4_out0	Convolution	icp3_out	icp4_out0	1x1x512x160	160
relu_icp4_out0	ReLu	icp4_out0	icp4_out0x		
icp4_out1	Convolution	icp4_reduction1x	icp4_out1	3x3x112x224	224
relu_icp4_out1	ReLu	icp4_out1	icp4_out1x		
icp4_out2	Convolution	icp4_reduction2x	icp4_out2	5x5x24x64	64
relu_icp4_out2	ReLu	icp4_out2	icp4_out2x		
icp4_out3	Convolution	icp4_pool	icp4_out3	1x1x512x64	64
relu_icp4_out3	ReLu	icp4_out3	icp4_out3x		
icp4_out	Concatenation	icp4_out0x, icp4_out1x, icp4_out2x, icp4_out3x	icp4_out		
icp5_reduction1	Convolution	icp4_out	icp5_reduction1	1x1x512x128	128
relu_icp5_reduction1	ReLu	icp5_reduction1	icp5_reduction1x		
icp5_reduction2	Convolution	icp4_out	icp5_reduction2	1x1x512x24	24
relu_icp5_reduction2	ReLu	icp5_reduction2	icp5_reduction2x		
icp5_pool	Max Pooling	icp4_out	icp5_pool		
icp5_out0	Convolution	icp4_out	icp5_out0	1x1x512x128	128
relu_icp5_out0	ReLu	icp5_out0	icp5_out0x		
icp5_out1	Convolution	icp5_reduction1x	icp5_out1	3x3x128x256	256
relu_icp5_out1	ReLu	icp5_out1	icp5_out1x		
icp5_out2	Convolution	icp5_reduction2x	icp5_out2	5x5x24x64	64
relu_icp5_out2	ReLu	icp5_out2	icp5_out2x		
icp5_out3	Convolution	icp5_pool	icp5_out3	1x1x512x64	64
relu_icp5_out3	ReLu	icp5_out3	icp5_out3x		
icp5_out	Concatenation	icp5_out0x, icp5_out1x, icp5_out2x, icp5_out3x	icp5_out		
icp6_reduction1	Convolution	icp5_out	icp6_reduction1	1x1x512x144	144
relu_icp6_reduction1	ReLu	icp6_reduction1	icp6_reduction1x		
icp6_reduction2	Convolution	icp5_out	icp6_reduction2	1x1x512x32	32
relu_icp6_reduction2	ReLu	icp6_reduction2	icp6_reduction2x		
icp6_pool	Max Pooling	icp5_out	icp6_pool		
icp6_out0	Convolution	icp5_out	icp6_out0	1x1x512x112	112
relu_icp6_out0	ReLu	icp6_out0	icp6_out0x		
icp6_out1	Convolution	icp6_reduction1x	icp6_out1	3x3x144x288	288
relu_icp6_out1	ReLu	icp6_out1	icp6_out1x		
icp6_out2	Convolution	icp6_reduction2x	icp6_out2	5x5x32x64	64
relu_icp6_out2	ReLu	icp6_out2	icp6_out2x		
icp6_out3	Convolution	icp6_pool	icp6_out3	1x1x512x64	64
relu_icp6_out3	ReLu	icp6_out3	icp6_out3x		
icp6_out	Concatenation	icp6_out0x, icp6_out1x, icp6_out2x, icp6_out3x	icp6_out		
cls2_pool	Max Pooling	icp6_out	cls2_pool		

Table 19. GoogleNet Architecture (cont'd)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
cls2_reduction	Convolution	cls2_pool	cls2_reduction	1x1x528x128	128
relu_cls2_reduction	ReLu	cls2_reduction	cls2_reductionx		
cls2_fc1	Convolution	cls2_reductionx	cls2_fc1	4x4x128x1024	1024
relu_cls2_fc1	ReLu	cls2_fc1	cls2_fc1x		
cls2_fc2	Convolution	cls2_fc1x	cls2_fc2	1x1x1024x1000	1000
icp7_reduction1	Convolution	icp6_out	icp7_reduction1	1x1x528x160	160
relu_icp7_reduction1	ReLu	icp7_reduction1	icp7_reduction1x		
icp7_reduction2	Convolution	icp6_out	icp7_reduction2	1x1x528x32	32
relu_icp7_reduction2	ReLu	icp7_reduction2	icp7_reduction2x		
icp7_pool	Max Pooling	icp6_out	icp7_pool		
icp7_out0	Convolution	icp6_out	icp7_out0	1x1x528x256	256
relu_icp7_out0	ReLu	icp7_out0	icp7_out0x		
icp7_out1	Convolution	icp7_reduction1x	icp7_out1	3x3x160x320	320
relu_icp7_out1	ReLu	icp7_out1	icp7_out1x		
icp7_out2	Convolution	icp7_reduction2x	icp7_out2	5x5x32x128	128
relu_icp7_out2	ReLu	icp7_out2	icp7_out2x		
icp7_out3	Convolution	icp7_pool	icp7_out3	1x1x528x128	128
relu_icp7_out3	ReLu	icp7_out3	icp7_out3x		
icp7_out	Concatenation	icp7_out0x, icp7_out1x, icp7_out2x, icp7_out3x	icp7_out		
icp8_in	Max Pooling	icp7_out	icp8_in		
icp8_reduction1	Convolution	icp8_in	icp8_reduction1	1x1x832x160	160
relu_icp8_reduction1	ReLu	icp8_reduction1	icp8_reduction1x		
icp8_reduction2	Convolution	icp8_in	icp8_reduction2	1x1x832x32	32
relu_icp8_reduction2	ReLu	icp8_reduction2	icp8_reduction2x		
icp8_pool	Max Pooling	icp8_in	icp8_pool		
icp8_out0	Convolution	icp8_in	icp8_out0	1x1x832x256	256
relu_icp8_out0	ReLu	icp8_out0	icp8_out0x		
icp8_out1	Convolution	icp8_reduction1x	icp8_out1	3x3x160x320	320
relu_icp8_out1	ReLu	icp8_out1	icp8_out1x		
icp8_out2	Convolution	icp8_reduction2x	icp8_out2	5x5x32x128	128
relu_icp8_out2	ReLu	icp8_out2	icp8_out2x		
icp8_out3	Convolution	icp8_pool	icp8_out3	1x1x832x128	128
relu_icp8_out3	ReLu	icp8_out3	icp8_out3x		
icp8_out	Concatenation	icp8_out0x, icp8_out1x, icp8_out2x, icp8_out3x	icp8_out		
icp9_reduction1	Convolution	icp8_out	icp9_reduction1	1x1x832x192	192
relu_icp9_reduction1	ReLu	icp9_reduction1	icp9_reduction1x		
icp9_reduction2	Convolution	icp8_out	icp9_reduction2	1x1x832x48	48
relu_icp9_reduction2	ReLu	icp9_reduction2	icp9_reduction2x		
icp9_pool	Max Pooling	icp8_out	icp9_pool		
icp9_out0	Convolution	icp8_out	icp9_out0	1x1x832x384	384
relu_icp9_out0	ReLu	icp9_out0	icp9_out0x		
icp9_out1	Convolution	icp9_reduction1x	icp9_out1	3x3x192x384	384
relu_icp9_out1	ReLu	icp9_out1	icp9_out1x		
icp9_out2	Convolution	icp9_reduction2x	icp9_out2	5x5x48x128	128
relu_icp9_out2	ReLu	icp9_out2	icp9_out2x		
icp9_out3	Convolution	icp9_pool	icp9_out3	1x1x832x128	128
relu_icp9_out3	ReLu	icp9_out3	icp9_out3x		
icp9_out	Concatenation	icp9_out0x, icp9_out1x, icp9_out2x, icp9_out3x	icp9_out		
cls3_pool	Max Pooling	icp9_out	cls3_pool		
cls3_fc	Convolution	cls3_pool	cls3_fc	1x1x1024x1000	1000
softmax	Softmax	cls3_fc	prob		

Table 20.ResNet50 Architecture

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
conv1	Convolution	data	conv1	7x7x3x64	64
bn_conv1	Batch Normalization	conv1	conv1x		
conv1_relu	ReLu	conv1x	conv1xxx		
pool1	Max Pooling	conv1xxx	pool1		
res2a_branch1	Convolution	pool1	res2a_branch1	1x1x64x256	256
bn2a_branch1	Batch Normalization	res2a_branch1	res2a_branch1x		
res2a_branch2a	Convolution	pool1	res2a_branch2a	1x1x64x64	64
bn2a_branch2a	Batch Normalization	res2a_branch2a	res2a_branch2ax		
res2a_branch2a_relu	ReLu	res2a_branch2ax	res2a_branch2axxx		
res2a_branch2b	Convolution	res2a_branch2axxx	res2a_branch2b	3x3x64x64	64
bn2a_branch2b	Batch Normalization	res2a_branch2b	res2a_branch2bx		
res2a_branch2b_relu	ReLu	res2a_branch2bx	res2a_branch2bxxx		
res2a_branch2c	Convolution	res2a_branch2bxxx	res2a_branch2c	1x1x64x256	256
bn2a_branch2c	Batch Normalization	res2a_branch2c	res2a_branch2cx		
res2a	Summation	res2a_branch1x, res2a_branch2cx	res2a		
res2a_relu	ReLu	res2a	res2ax		
res2b_branch2a	Convolution	res2ax	res2b_branch2a	1x1x256x64	64
bn2b_branch2a	Batch Normalization	res2b_branch2a	res2b_branch2ax		
res2b_branch2a_relu	ReLu	res2b_branch2ax	res2b_branch2axxx		
res2b_branch2b	Convolution	res2b_branch2axxx	res2b_branch2b	3x3x64x64	64
bn2b_branch2b	Batch Normalization	res2b_branch2b	res2b_branch2bx		
res2b_branch2b_relu	ReLu	res2b_branch2bx	res2b_branch2bxxx		
res2b_branch2c	Convolution	res2b_branch2bxxx	res2b_branch2c	1x1x64x256	256
bn2b_branch2c	Batch Normalization	res2b_branch2c	res2b_branch2cx		
res2b	Summation	res2ax, res2b_branch2cx	res2b		
res2b_relu	ReLu	res2b	res2bx		
res2c_branch2a	Convolution	res2bx	res2c_branch2a	1x1x256x64	64
bn2c_branch2a	Batch Normalization	res2c_branch2a	res2c_branch2ax		
res2c_branch2a_relu	ReLu	res2c_branch2ax	res2c_branch2axxx		
res2c_branch2b	Convolution	res2c_branch2axxx	res2c_branch2b	3x3x64x64	64
bn2c_branch2b	Batch Normalization	res2c_branch2b	res2c_branch2bx		
res2c_branch2b_relu	ReLu	res2c_branch2bx	res2c_branch2bxxx		
res2c_branch2c	Convolution	res2c_branch2bxxx	res2c_branch2c	1x1x64x256	256
bn2c_branch2c	Batch Normalization	res2c_branch2c	res2c_branch2cx		
res2c	Summation	res2bx, res2c_branch2cx	res2c		
res2c_relu	ReLu	res2c	res2cx		
res3a_branch1	Convolution	res2cx	res3a_branch1	1x1x256x512	512
bn3a_branch1	Batch Normalization	res3a_branch1	res3a_branch1x		
res3a_branch2a	Convolution	res2cx	res3a_branch2a	1x1x256x128	128
bn3a_branch2a	Batch Normalization	res3a_branch2a	res3a_branch2ax		
res3a_branch2a_relu	ReLu	res3a_branch2ax	res3a_branch2axxx		
res3a_branch2b	Convolution	res3a_branch2axxx	res3a_branch2b	3x3x128x128	128
bn3a_branch2b	Batch Normalization	res3a_branch2b	res3a_branch2bx		
res3a_branch2b_relu	ReLu	res3a_branch2bx	res3a_branch2bxxx		
res3a_branch2c	Convolution	res3a_branch2bxxx	res3a_branch2c	1x1x128x512	512

Table 20. ResNet50 Architecture (cont’d)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
bn3a_branch2c	Batch Normalization	res3a_branch2c	res3a_branch2cx		
res3a	Summation	res3a_branch1x, res3a_branch2cx	res3a		
res3a_relu	ReLu	res3a	res3ax		
res3b_branch2a	Convolution	res3ax	res3b_branch2a	1x1x512x128	128
bn3b_branch2a	Batch Normalization	res3b_branch2a	res3b_branch2ax		
res3b_branch2a_relu	ReLu	res3b_branch2ax	res3b_branch2axxx		
res3b_branch2b	Convolution	res3b_branch2axxx	res3b_branch2b	3x3x128x128	128
bn3b_branch2b	Batch Normalization	res3b_branch2b	res3b_branch2bx		
res3b_branch2b_relu	ReLu	res3b_branch2bx	res3b_branch2bxxx		
res3b_branch2c	Convolution	res3b_branch2bxxx	res3b_branch2c	1x1x128x512	512
bn3b_branch2c	Batch Normalization	res3b_branch2c	res3b_branch2cx		
res3b	Summation	res3ax, res3b_branch2cx	res3b		
res3b_relu	ReLu	res3b	res3bx		
res3c_branch2a	Convolution	res3bx	res3c_branch2a	1x1x512x128	128
bn3c_branch2a	Batch Normalization	res3c_branch2a	res3c_branch2ax		
res3c_branch2a_relu	ReLu	res3c_branch2ax	res3c_branch2axxx		
res3c_branch2b	Convolution	res3c_branch2axxx	res3c_branch2b	3x3x128x128	128
bn3c_branch2b	Batch Normalization	res3c_branch2b	res3c_branch2bx		
res3c_branch2b_relu	ReLu	res3c_branch2bx	res3c_branch2bxxx		
res3c_branch2c	Convolution	res3c_branch2bxxx	res3c_branch2c	1x1x128x512	512
bn3c_branch2c	Batch Normalization	res3c_branch2c	res3c_branch2cx		
res3c	Summation	res3bx, res3c_branch2cx	res3c		
res3c_relu	ReLu	res3c	res3cx		
res3d_branch2a	Convolution	res3cx	res3d_branch2a	1x1x512x128	128
bn3d_branch2a	Batch Normalization	res3d_branch2a	res3d_branch2ax		
res3d_branch2a_relu	ReLu	res3d_branch2ax	res3d_branch2axxx		
res3d_branch2b	Convolution	res3d_branch2axxx	res3d_branch2b	3x3x128x128	128
bn3d_branch2b	Batch Normalization	res3d_branch2b	res3d_branch2bx		
res3d_branch2b_relu	ReLu	res3d_branch2bx	res3d_branch2bxxx		
res3d_branch2c	Convolution	res3d_branch2bxxx	res3d_branch2c	1x1x128x512	512
bn3d_branch2c	Batch Normalization	res3d_branch2c	res3d_branch2cx		
res3d	Summation	res3cx, res3d_branch2cx	res3d		
res3d_relu	ReLu	res3d	res3dx		
res4a_branch1	Convolution	res3dx	res4a_branch1	1x1x512x1024	1024
bn4a_branch1	Batch Normalization	res4a_branch1	res4a_branch1x		
res4a_branch2a	Convolution	res3dx	res4a_branch2a	1x1x512x256	256
bn4a_branch2a	Batch Normalization	res4a_branch2a	res4a_branch2ax		
res4a_branch2a_relu	ReLu	res4a_branch2ax	res4a_branch2axxx		
res4a_branch2b	Convolution	res4a_branch2axxx	res4a_branch2b	3x3x256x256	256
bn4a_branch2b	Batch Normalization	res4a_branch2b	res4a_branch2bx		
res4a_branch2b_relu	ReLu	res4a_branch2bx	res4a_branch2bxxx		
res4a_branch2c	Convolution	res4a_branch2bxxx	res4a_branch2c	1x1x256x1024	1024
bn4a_branch2c	Batch Normalization	res4a_branch2c	res4a_branch2cx		

Table 20. ResNet50 Architecture (cont'd)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
res4a	Summation	res4a_branch1x, res4a_branch2cx	res4a		
res4a_relu	ReLu	res4a	res4ax		
res4b_branch2a	Convolution	res4ax	res4b_branch2a	1x1x1024x256	256
bn4b_branch2a	Batch Normalization	res4b_branch2a	res4b_branch2ax		
res4b_branch2a_relu	ReLu	res4b_branch2ax	res4b_branch2axxx		
res4b_branch2b	Convolution	res4b_branch2axxx	res4b_branch2b	3x3x256x256	256
bn4b_branch2b	Batch Normalization	res4b_branch2b	res4b_branch2bx		
res4b_branch2b_relu	ReLu	res4b_branch2bx	res4b_branch2bxxx		
res4b_branch2c	Convolution	res4b_branch2bxxx	res4b_branch2c	1x1x256x1024	1024
bn4b_branch2c	Batch Normalization	res4b_branch2c	res4b_branch2cx		
res4b	Summation	res4ax, res4b_branch2cx	res4b		
res4b_relu	ReLu	res4b	res4bx		
res4c_branch2a	Convolution	res4bx	res4c_branch2a	1x1x1024x256	256
bn4c_branch2a	Batch Normalization	res4c_branch2a	res4c_branch2ax		
res4c_branch2a_relu	ReLu	res4c_branch2ax	res4c_branch2axxx		
res4c_branch2b	Convolution	res4c_branch2axxx	res4c_branch2b	3x3x256x256	256
bn4c_branch2b	Batch Normalization	res4c_branch2b	res4c_branch2bx		
res4c_branch2b_relu	ReLu	res4c_branch2bx	res4c_branch2bxxx		
res4c_branch2c	Convolution	res4c_branch2bxxx	res4c_branch2c	1x1x256x1024	1024
bn4c_branch2c	Batch Normalization	res4c_branch2c	res4c_branch2cx		
res4c	Summation	res4bx, res4c_branch2cx	res4c		
res4c_relu	ReLu	res4c	res4cx		
res4d_branch2a	Convolution	res4cx	res4d_branch2a	1x1x1024x256	256
bn4d_branch2a	Batch Normalization	res4d_branch2a	res4d_branch2ax		
res4d_branch2a_relu	ReLu	res4d_branch2ax	res4d_branch2axxx		
res4d_branch2b	Convolution	res4d_branch2axxx	res4d_branch2b	3x3x256x256	256
bn4d_branch2b	Batch Normalization	res4d_branch2b	res4d_branch2bx		
res4d_branch2b_relu	ReLu	res4d_branch2bx	res4d_branch2bxxx		
res4d_branch2c	Convolution	res4d_branch2bxxx	res4d_branch2c	1x1x256x1024	1024
bn4d_branch2c	Batch Normalization	res4d_branch2c	res4d_branch2cx		
res4d	Summation	res4cx, res4d_branch2cx	res4d		
res4d_relu	ReLu	res4d	res4dx		
res4e_branch2a	Convolution	res4dx	res4e_branch2a	1x1x1024x256	256
bn4e_branch2a	Batch Normalization	res4e_branch2a	res4e_branch2ax		
res4e_branch2a_relu	ReLu	res4e_branch2ax	res4e_branch2axxx		
res4e_branch2b	Convolution	res4e_branch2axxx	res4e_branch2b	3x3x256x256	256
bn4e_branch2b	Batch Normalization	res4e_branch2b	res4e_branch2bx		
res4e_branch2b_relu	ReLu	res4e_branch2bx	res4e_branch2bxxx		
res4e_branch2c	Convolution	res4e_branch2bxxx	res4e_branch2c	1x1x256x1024	1024
bn4e_branch2c	Batch Normalization	res4e_branch2c	res4e_branch2cx		
res4e	Summation	res4dx, res4e_branch2cx	res4e		
res4e_relu	ReLu	res4e	res4ex		
res4f_branch2a	Convolution	res4ex	res4f_branch2a	1x1x1024x256	256
bn4f_branch2a	Batch Normalization	res4f_branch2a	res4f_branch2ax		

Table 20. ResNet50 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
res4f_branch2a_relu	ReLu	res4f_branch2ax	res4f_branch2axxx		
res4f_branch2b	Convolution	res4f_branch2axxx	res4f_branch2b	3x3x256x256	256
bn4f_branch2b	Batch Normalization	res4f_branch2b	res4f_branch2bx		
res4f_branch2b_relu	ReLu	res4f_branch2bx	res4f_branch2bxxx		
res4f_branch2c	Convolution	res4f_branch2bxxx	res4f_branch2c	1x1x256x1024	1024
bn4f_branch2c	Batch Normalization	res4f_branch2c	res4f_branch2cx		
res4f	Summation	res4ex, res4f_branch2cx	res4f		
res4f_relu	ReLu	res4f	res4fx		
res5a_branch1	Convolution	res4fx	res5a_branch1	1x1x1024x2048	2048
bn5a_branch1	Batch Normalization	res5a_branch1	res5a_branch1x		
res5a_branch2a	Convolution	res4fx	res5a_branch2a	1x1x1024x512	512
bn5a_branch2a	Batch Normalization	res5a_branch2a	res5a_branch2ax		
res5a_branch2a_relu	ReLu	res5a_branch2ax	res5a_branch2axxx		
res5a_branch2b	Convolution	res5a_branch2axxx	res5a_branch2b	3x3x512x512	512
bn5a_branch2b	Batch Normalization	res5a_branch2b	res5a_branch2bx		
res5a_branch2b_relu	ReLu	res5a_branch2bx	res5a_branch2bxxx		
res5a_branch2c	Convolution	res5a_branch2bxxx	res5a_branch2c	1x1x512x2048	2048
bn5a_branch2c	Batch Normalization	res5a_branch2c	res5a_branch2cx		
res5a	Summation	res5a_branch1x, res5a_branch2cx	res5a		
res5a_relu	ReLu	res5a	res5ax		
res5b_branch2a	Convolution	res5ax	res5b_branch2a	1x1x2048x512	512
bn5b_branch2a	Batch Normalization	res5b_branch2a	res5b_branch2ax		
res5b_branch2a_relu	ReLu	res5b_branch2ax	res5b_branch2axxx		
res5b_branch2b	Convolution	res5b_branch2axxx	res5b_branch2b	3x3x512x512	512
bn5b_branch2b	Batch Normalization	res5b_branch2b	res5b_branch2bx		
res5b_branch2b_relu	ReLu	res5b_branch2bx	res5b_branch2bxxx		
res5b_branch2c	Convolution	res5b_branch2bxxx	res5b_branch2c	1x1x512x2048	2048
bn5b_branch2c	Batch Normalization	res5b_branch2c	res5b_branch2cx		
res5b	Summation	res5ax, res5b_branch2cx	res5b		
res5b_relu	ReLu	res5b	res5bx		
res5c_branch2a	Convolution	res5bx	res5c_branch2a	1x1x2048x512	512
bn5c_branch2a	Batch Normalization	res5c_branch2a	res5c_branch2ax		
res5c_branch2a_relu	ReLu	res5c_branch2ax	res5c_branch2axxx		
res5c_branch2b	Convolution	res5c_branch2axxx	res5c_branch2b	3x3x512x512	512
bn5c_branch2b	Batch Normalization	res5c_branch2b	res5c_branch2bx		
res5c_branch2b_relu	ReLu	res5c_branch2bx	res5c_branch2bxxx		
res5c_branch2c	Convolution	res5c_branch2bxxx	res5c_branch2c	1x1x512x2048	2048
bn5c_branch2c	Batch Normalization	res5c_branch2c	res5c_branch2cx		
res5c	Summation	res5bx, res5c_branch2cx	res5c		
res5c_relu	ReLu	res5c	res5cx		
pool5	Average Pooling	res5cx	pool5		
fc1000	Convolution	pool5	fc1000	1x1x2048x1000	1000
prob	SoftMax	fc1000	prob		

Table 21. ResNet101 Architecture

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
conv1	Convolutional	data	conv1	7x7x3x64	64
bn_conv1	Batch Normalization	conv1	conv1x		
conv1_relu	ReLu	conv1x	conv1xxx		
pool1	Max Pooling	conv1xxx	pool1		
res2a_branch1	Convolutional	pool1	res2a_branch1	1x1x64x256	256
bn2a_branch1	Batch Normalization	res2a_branch1	res2a_branch1x		
res2a_branch2a	Convolutional	pool1	res2a_branch2a	1x1x64x64	64
bn2a_branch2a	Batch Normalization	res2a_branch2a	res2a_branch2ax		
res2a_branch2a_relu	ReLu	res2a_branch2ax	res2a_branch2axxx		
res2a_branch2b	Convolutional	res2a_branch2axxx	res2a_branch2b	3x3x64x64	64
bn2a_branch2b	Batch Normalization	res2a_branch2b	res2a_branch2bx		
res2a_branch2b_relu	ReLu	res2a_branch2bx	res2a_branch2bxxx		
res2a_branch2c	Convolutional	res2a_branch2bxxx	res2a_branch2c	1x1x64x256	256
bn2a_branch2c	Batch Normalization	res2a_branch2c	res2a_branch2cx		
res2a	Summation	res2a_branch1x, res2a_branch2cx	res2a		
res2a_relu	ReLu	res2a	res2ax		
res2b_branch2a	Convolutional	res2ax	res2b_branch2a	1x1x256x64	64
bn2b_branch2a	Batch Normalization	res2b_branch2a	res2b_branch2ax		
res2b_branch2a_relu	ReLu	res2b_branch2ax	res2b_branch2axxx		
res2b_branch2b	Convolutional	res2b_branch2axxx	res2b_branch2b	3x3x64x64	64
bn2b_branch2b	Batch Normalization	res2b_branch2b	res2b_branch2bx		
res2b_branch2b_relu	ReLu	res2b_branch2bx	res2b_branch2bxxx		
res2b_branch2c	Convolutional	res2b_branch2bxxx	res2b_branch2c	1x1x64x256	256
bn2b_branch2c	Batch Normalization	res2b_branch2c	res2b_branch2cx		
res2b	Summation	res2ax, res2b_branch2cx	res2b		
res2b_relu	ReLu	res2b	res2bx		
res2c_branch2a	Convolutional	res2bx	res2c_branch2a	1x1x256x64	64
bn2c_branch2a	Batch Normalization	res2c_branch2a	res2c_branch2ax		
res2c_branch2a_relu	ReLu	res2c_branch2ax	res2c_branch2axxx		
res2c_branch2b	Convolutional	res2c_branch2axxx	res2c_branch2b	3x3x64x64	64
bn2c_branch2b	Batch Normalization	res2c_branch2b	res2c_branch2bx		
res2c_branch2b_relu	ReLu	res2c_branch2bx	res2c_branch2bxxx		
res2c_branch2c	Convolutional	res2c_branch2bxxx	res2c_branch2c	1x1x64x256	256
bn2c_branch2c	Batch Normalization	res2c_branch2c	res2c_branch2cx		
res2c	Summation	res2bx, res2c_branch2cx	res2c		
res2c_relu	ReLu	res2c	res2cx		
res3a_branch1	Convolutional	res2cx	res3a_branch1	1x1x256x512	512
bn3a_branch1	Batch Normalization	res3a_branch1	res3a_branch1x		
res3a_branch2a	Convolutional	res2cx	res3a_branch2a	1x1x256x128	128
bn3a_branch2a	Batch Normalization	res3a_branch2a	res3a_branch2ax		
res3a_branch2a_relu	ReLu	res3a_branch2ax	res3a_branch2axxx		
res3a_branch2b	Convolutional	res3a_branch2axxx	res3a_branch2b	3x3x128x128	128
bn3a_branch2b	Batch Normalization	res3a_branch2b	res3a_branch2bx		
res3a_branch2b_relu	ReLu	res3a_branch2bx	res3a_branch2bxxx		
res3a_branch2c	Convolutional	res3a_branch2bxxx	res3a_branch2c	1x1x128x512	512

Table 21. ResNet101 Architecture (cont'd)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
bn3a_branch2c	Batch Normalization	res3a_branch2c	res3a_branch2cx		
res3a	Summation	res3a_branch1x, res3a_branch2cx	res3a		
res3a_relu	ReLu	res3a	res3ax		
res3b1_branch2a	Convolutional	res3ax	res3b1_branch2a	1x1x512x128	128
bn3b1_branch2a	Batch Normalization	res3b1_branch2a	res3b1_branch2ax		
res3b1_branch2a_relu	ReLu	res3b1_branch2ax	res3b1_branch2axxx		
res3b1_branch2b	Convolutional	res3b1_branch2axxx	res3b1_branch2b	3x3x128x128	128
bn3b1_branch2b	Batch Normalization	res3b1_branch2b	res3b1_branch2bx		
res3b1_branch2b_relu	ReLu	res3b1_branch2bx	res3b1_branch2bxxx		
res3b1_branch2c	Convolutional	res3b1_branch2bxxx	res3b1_branch2c	1x1x128x512	512
bn3b1_branch2c	Batch Normalization	res3b1_branch2c	res3b1_branch2cx		
res3b1	Summation	res3ax, res3b1_branch2cx	res3b1		
res3b1_relu	ReLu	res3b1	res3b1x		
res3b2_branch2a	Convolutional	res3b1x	res3b2_branch2a	1x1x512x128	128
bn3b2_branch2a	Batch Normalization	res3b2_branch2a	res3b2_branch2ax		
res3b2_branch2a_relu	ReLu	res3b2_branch2ax	res3b2_branch2axxx		
res3b2_branch2b	Convolutional	res3b2_branch2axxx	res3b2_branch2b	3x3x128x128	128
bn3b2_branch2b	Batch Normalization	res3b2_branch2b	res3b2_branch2bx		
res3b2_branch2b_relu	ReLu	res3b2_branch2bx	res3b2_branch2bxxx		
res3b2_branch2c	Convolutional	res3b2_branch2bxxx	res3b2_branch2c	1x1x128x512	512
bn3b2_branch2c	Batch Normalization	res3b2_branch2c	res3b2_branch2cx		
res3b2	Summation	res3b1x, res3b2_branch2cx	res3b2		
res3b2_relu	ReLu	res3b2	res3b2x		
res3b3_branch2a	Convolutional	res3b2x	res3b3_branch2a	1x1x512x128	128
bn3b3_branch2a	Batch Normalization	res3b3_branch2a	res3b3_branch2ax		
res3b3_branch2a_relu	ReLu	res3b3_branch2ax	res3b3_branch2axxx		
res3b3_branch2b	Convolutional	res3b3_branch2axxx	res3b3_branch2b	3x3x128x128	128
bn3b3_branch2b	Batch Normalization	res3b3_branch2b	res3b3_branch2bx		
res3b3_branch2b_relu	ReLu	res3b3_branch2bx	res3b3_branch2bxxx		
res3b3_branch2c	Convolutional	res3b3_branch2bxxx	res3b3_branch2c	1x1x128x512	512
bn3b3_branch2c	Batch Normalization	res3b3_branch2c	res3b3_branch2cx		
res3b3	Summation	res3b2x, res3b3_branch2cx	res3b3		
res3b3_relu	ReLu	res3b3	res3b3x		
res4a_branch1	Convolutional	res3b3x	res4a_branch1	1x1x512x1024	1024
bn4a_branch1	Batch Normalization	res4a_branch1	res4a_branch1x		
res4a_branch2a	Convolutional	res3b3x	res4a_branch2a	1x1x512x256	256
bn4a_branch2a	Batch Normalization	res4a_branch2a	res4a_branch2ax		
res4a_branch2a_relu	ReLu	res4a_branch2ax	res4a_branch2axxx		
res4a_branch2b	Convolutional	res4a_branch2axxx	res4a_branch2b	3x3x256x256	256
bn4a_branch2b	Batch Normalization	res4a_branch2b	res4a_branch2bx		
res4a_branch2b_relu	ReLu	res4a_branch2bx	res4a_branch2bxxx		
res4a_branch2c	Convolutional	res4a_branch2bxxx	res4a_branch2c	1x1x256x1024	1024
bn4a_branch2c	Batch Normalization	res4a_branch2c	res4a_branch2cx		

Table 21. ResNet101 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
res4a	Summation	res4a_branch1x, res4a_branch2cx	res4a		
res4a_relu	ReLu	res4a	res4ax		
res4b1_branch2a	Convolutional	res4ax	res4b1_branch2a	1x1x1024x256	256
bn4b1_branch2a	Batch Normalization	res4b1_branch2a	res4b1_branch2ax		
res4b1_branch2a_relu	ReLu	res4b1_branch2ax	res4b1_branch2axxx		
res4b1_branch2b	Convolutional	res4b1_branch2axxx	res4b1_branch2b	3x3x256x256	256
bn4b1_branch2b	Batch Normalization	res4b1_branch2b	res4b1_branch2bx		
res4b1_branch2b_relu	ReLu	res4b1_branch2bx	res4b1_branch2bxxx		
res4b1_branch2c	Convolutional	res4b1_branch2bxxx	res4b1_branch2c	1x1x256x1024	1024
bn4b1_branch2c	Batch Normalization	res4b1_branch2c	res4b1_branch2cx		
res4b1	Summation	res4ax, res4b1_branch2cx	res4b1		
res4b1_relu	ReLu	res4b1	res4b1x		
res4b2_branch2a	Convolutional	res4b1x	res4b2_branch2a	1x1x1024x256	256
bn4b2_branch2a	Batch Normalization	res4b2_branch2a	res4b2_branch2ax		
res4b2_branch2a_relu	ReLu	res4b2_branch2ax	res4b2_branch2axxx		
res4b2_branch2b	Convolutional	res4b2_branch2axxx	res4b2_branch2b	3x3x256x256	256
bn4b2_branch2b	Batch Normalization	res4b2_branch2b	res4b2_branch2bx		
res4b2_branch2b_relu	ReLu	res4b2_branch2bx	res4b2_branch2bxxx		
res4b2_branch2c	Convolutional	res4b2_branch2bxxx	res4b2_branch2c	1x1x256x1024	1024
bn4b2_branch2c	Batch Normalization	res4b2_branch2c	res4b2_branch2cx		
res4b2	Summation	res4b1x, res4b2_branch2cx	res4b2		
res4b2_relu	ReLu	res4b2	res4b2x		
res4b3_branch2a	Convolutional	res4b2x	res4b3_branch2a	1x1x1024x256	256
bn4b3_branch2a	Batch Normalization	res4b3_branch2a	res4b3_branch2ax		
res4b3_branch2a_relu	ReLu	res4b3_branch2ax	res4b3_branch2axxx		
res4b3_branch2b	Convolutional	res4b3_branch2axxx	res4b3_branch2b	3x3x256x256	256
bn4b3_branch2b	Batch Normalization	res4b3_branch2b	res4b3_branch2bx		
res4b3_branch2b_relu	ReLu	res4b3_branch2bx	res4b3_branch2bxxx		
res4b3_branch2c	Convolutional	res4b3_branch2bxxx	res4b3_branch2c	1x1x256x1024	1024
bn4b3_branch2c	Batch Normalization	res4b3_branch2c	res4b3_branch2cx		
res4b3	Summation	res4b2x, res4b3_branch2cx	res4b3		
res4b3_relu	ReLu	res4b3	res4b3x		
res4b4_branch2a	Convolutional	res4b3x	res4b4_branch2a	1x1x1024x256	256
bn4b4_branch2a	Batch Normalization	res4b4_branch2a	res4b4_branch2ax		
res4b4_branch2a_relu	ReLu	res4b4_branch2ax	res4b4_branch2axxx		
res4b4_branch2b	Convolutional	res4b4_branch2axxx	res4b4_branch2b	3x3x256x256	256
bn4b4_branch2b	Batch Normalization	res4b4_branch2b	res4b4_branch2bx		
res4b4_branch2b_relu	ReLu	res4b4_branch2bx	res4b4_branch2bxxx		
res4b4_branch2c	Convolutional	res4b4_branch2bxxx	res4b4_branch2c	1x1x256x1024	1024
bn4b4_branch2c	Batch Normalization	res4b4_branch2c	res4b4_branch2cx		
res4b4	Summation	res4b3x, res4b4_branch2cx	res4b4		
res4b4_relu	ReLu	res4b4	res4b4x		
res4b5_branch2a	Convolutional	res4b4x	res4b5_branch2a	1x1x1024x256	256
bn4b5_branch2a	Batch Normalization	res4b5_branch2a	res4b5_branch2ax		

Table 21. ResNet101 Architecture (cont'd)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
res4b5_branch2a_relu	ReLu	res4b5_branch2ax	res4b5_branch2axxx		
res4b5_branch2b	Convolutional	res4b5_branch2axxx	res4b5_branch2b	3x3x256x256	256
bn4b5_branch2b	Batch Normalization	res4b5_branch2b	res4b5_branch2bx		
res4b5_branch2b_relu	ReLu	res4b5_branch2bx	res4b5_branch2bxxx		
res4b5_branch2c	Convolutional	res4b5_branch2bxxx	res4b5_branch2c	1x1x256x1024	1024
bn4b5_branch2c	Batch Normalization	res4b5_branch2c	res4b5_branch2cx		
res4b5	Summation	res4b4x, res4b5_branch2cx	res4b5		
res4b5_relu	ReLu	res4b5	res4b5x		
res4b6_branch2a	Convolutional	res4b5x	res4b6_branch2a	1x1x1024x256	256
bn4b6_branch2a	Batch Normalization	res4b6_branch2a	res4b6_branch2ax		
res4b6_branch2a_relu	ReLu	res4b6_branch2ax	res4b6_branch2axxx		
res4b6_branch2b	Convolutional	res4b6_branch2axxx	res4b6_branch2b	3x3x256x256	256
bn4b6_branch2b	Batch Normalization	res4b6_branch2b	res4b6_branch2bx		
res4b6_branch2b_relu	ReLu	res4b6_branch2bx	res4b6_branch2bxxx		
res4b6_branch2c	Convolutional	res4b6_branch2bxxx	res4b6_branch2c	1x1x256x1024	1024
bn4b6_branch2c	Batch Normalization	res4b6_branch2c	res4b6_branch2cx		
res4b6	Summation	res4b5x, res4b6_branch2cx	res4b6		
res4b6_relu	ReLu	res4b6	res4b6x		
res4b7_branch2a	Convolutional	res4b6x	res4b7_branch2a	1x1x1024x256	256
bn4b7_branch2a	Batch Normalization	res4b7_branch2a	res4b7_branch2ax		
res4b7_branch2a_relu	ReLu	res4b7_branch2ax	res4b7_branch2axxx		
res4b7_branch2b	Convolutional	res4b7_branch2axxx	res4b7_branch2b	3x3x256x256	256
bn4b7_branch2b	Batch Normalization	res4b7_branch2b	res4b7_branch2bx		
res4b7_branch2b_relu	ReLu	res4b7_branch2bx	res4b7_branch2bxxx		
res4b7_branch2c	Convolutional	res4b7_branch2bxxx	res4b7_branch2c	1x1x256x1024	1024
bn4b7_branch2c	Batch Normalization	res4b7_branch2c	res4b7_branch2cx		
res4b7	Summation	res4b6x, res4b7_branch2cx	res4b7		
res4b7_relu	ReLu	res4b7	res4b7x		
res4b8_branch2a	Convolutional	res4b7x	res4b8_branch2a	1x1x1024x256	256
bn4b8_branch2a	Batch Normalization	res4b8_branch2a	res4b8_branch2ax		
res4b8_branch2a_relu	ReLu	res4b8_branch2ax	res4b8_branch2axxx		
res4b8_branch2b	Convolutional	res4b8_branch2axxx	res4b8_branch2b	3x3x256x256	256
bn4b8_branch2b	Batch Normalization	res4b8_branch2b	res4b8_branch2bx		
res4b8_branch2b_relu	ReLu	res4b8_branch2bx	res4b8_branch2bxxx		
res4b8_branch2c	Convolutional	res4b8_branch2bxxx	res4b8_branch2c	1x1x256x1024	1024
bn4b8_branch2c	Batch Normalization	res4b8_branch2c	res4b8_branch2cx		
res4b8	Summation	res4b7x, res4b8_branch2cx	res4b8		
res4b8_relu	ReLu	res4b8	res4b8x		
res4b9_branch2a	Convolutional	res4b8x	res4b9_branch2a	1x1x1024x256	256
bn4b9_branch2a	Batch Normalization	res4b9_branch2a	res4b9_branch2ax		
res4b9_branch2a_relu	ReLu	res4b9_branch2ax	res4b9_branch2axxx		
res4b9_branch2b	Convolutional	res4b9_branch2axxx	res4b9_branch2b	3x3x256x256	256
bn4b9_branch2b	Batch Normalization	res4b9_branch2b	res4b9_branch2bx		
res4b9_branch2b_relu	ReLu	res4b9_branch2bx	res4b9_branch2bxxx		
res4b9_branch2c	Convolutional	res4b9_branch2bxxx	res4b9_branch2c	1x1x256x1024	1024

Table 21. ResNet101 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
bn4b9_branch2c	Batch Normalization	res4b9_branch2c	res4b9_branch2cx		
res4b9	Summation	res4b8x, res4b9_branch2cx	res4b9		
res4b9_relu	ReLu	res4b9	res4b9x		
res4b10_branch2a	Convolutional	res4b9x	res4b10_branch2a	1x1x1024x256	256
bn4b10_branch2a	Batch Normalization	res4b10_branch2a	res4b10_branch2ax		
res4b10_branch2a_relu	ReLu	res4b10_branch2ax	res4b10_branch2axxx		
res4b10_branch2b	Convolutional	res4b10_branch2axxx	res4b10_branch2b	3x3x256x256	256
bn4b10_branch2b	Batch Normalization	res4b10_branch2b	res4b10_branch2bx		
res4b10_branch2b_relu	ReLu	res4b10_branch2bx	res4b10_branch2bxxx		
res4b10_branch2c	Convolutional	res4b10_branch2bxxx	res4b10_branch2c	1x1x256x1024	1024
bn4b10_branch2c	Batch Normalization	res4b10_branch2c	res4b10_branch2cx		
res4b10	Summation	res4b9x, res4b10_branch2cx	res4b10		
res4b10_relu	ReLu	res4b10	res4b10x		
res4b11_branch2a	Convolutional	res4b10x	res4b11_branch2a	1x1x1024x256	256
bn4b11_branch2a	Batch Normalization	res4b11_branch2a	res4b11_branch2ax		
res4b11_branch2a_relu	ReLu	res4b11_branch2ax	res4b11_branch2axxx		
res4b11_branch2b	Convolutional	res4b11_branch2axxx	res4b11_branch2b	3x3x256x256	256
bn4b11_branch2b	Batch Normalization	res4b11_branch2b	res4b11_branch2bx		
res4b11_branch2b_relu	ReLu	res4b11_branch2bx	res4b11_branch2bxxx		
res4b11_branch2c	Convolutional	res4b11_branch2bxxx	res4b11_branch2c	1x1x256x1024	1024
bn4b11_branch2c	Batch Normalization	res4b11_branch2c	res4b11_branch2cx		
res4b11	Summation	res4b10x, res4b11_branch2cx	res4b11		
res4b11_relu	ReLu	res4b11	res4b11x		
res4b12_branch2a	Convolutional	res4b11x	res4b12_branch2a	1x1x1024x256	256
bn4b12_branch2a	Batch Normalization	res4b12_branch2a	res4b12_branch2ax		
res4b12_branch2a_relu	ReLu	res4b12_branch2ax	res4b12_branch2axxx		
res4b12_branch2b	Convolutional	res4b12_branch2axxx	res4b12_branch2b	3x3x256x256	256
bn4b12_branch2b	Batch Normalization	res4b12_branch2b	res4b12_branch2bx		
res4b12_branch2b_relu	ReLu	res4b12_branch2bx	res4b12_branch2bxxx		
res4b12_branch2c	Convolutional	res4b12_branch2bxxx	res4b12_branch2c	1x1x256x1024	1024
bn4b12_branch2c	Batch Normalization	res4b12_branch2c	res4b12_branch2cx		
res4b12	Summation	res4b11x, res4b12_branch2cx	res4b12		
res4b12_relu	ReLu	res4b12	res4b12x		
res4b13_branch2a	Convolutional	res4b12x	res4b13_branch2a	1x1x1024x256	256
bn4b13_branch2a	Batch Normalization	res4b13_branch2a	res4b13_branch2ax		
res4b13_branch2a_relu	ReLu	res4b13_branch2ax	res4b13_branch2axxx		
res4b13_branch2b	Convolutional	res4b13_branch2axxx	res4b13_branch2b	3x3x256x256	256
bn4b13_branch2b	Batch Normalization	res4b13_branch2b	res4b13_branch2bx		
res4b13_branch2b_relu	ReLu	res4b13_branch2bx	res4b13_branch2bxxx		
res4b13_branch2c	Convolutional	res4b13_branch2bxxx	res4b13_branch2c	1x1x256x1024	1024
bn4b13_branch2c	Batch Normalization	res4b13_branch2c	res4b13_branch2cx		
res4b13	Summation	res4b12x, res4b13_branch2cx	res4b13		
res4b13_relu	ReLu	res4b13	res4b13x		
res4b14_branch2a	Convolutional	res4b13x	res4b14_branch2a	1x1x1024x256	256

Table 21. ResNet101 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
bn4b14_branch2a	Batch Normalization	res4b14_branch2a	res4b14_branch2ax		
res4b14_branch2a_relu	ReLu	res4b14_branch2ax	res4b14_branch2axxx		
res4b14_branch2b	Convolutional	res4b14_branch2axxx	res4b14_branch2b	3x3x256x256	256
bn4b14_branch2b	Batch Normalization	res4b14_branch2b	res4b14_branch2bx		
res4b14_branch2b_relu	ReLu	res4b14_branch2bx	res4b14_branch2bxxx		
res4b14_branch2c	Convolutional	res4b14_branch2bxxx	res4b14_branch2c	1x1x256x1024	1024
bn4b14_branch2c	Batch Normalization	res4b14_branch2c	res4b14_branch2cx		
res4b14	Summation	res4b13x, res4b14_branch2cx	res4b14		
res4b14_relu	ReLu	res4b14	res4b14x		
res4b15_branch2a	Convolutional	res4b14x	res4b15_branch2a	1x1x1024x256	256
bn4b15_branch2a	Batch Normalization	res4b15_branch2a	res4b15_branch2ax		
res4b15_branch2a_relu	ReLu	res4b15_branch2ax	res4b15_branch2axxx		
res4b15_branch2b	Convolutional	res4b15_branch2axxx	res4b15_branch2b	3x3x256x256	256
bn4b15_branch2b	Batch Normalization	res4b15_branch2b	res4b15_branch2bx		
res4b15_branch2b_relu	ReLu	res4b15_branch2bx	res4b15_branch2bxxx		
res4b15_branch2c	Convolutional	res4b15_branch2bxxx	res4b15_branch2c	1x1x256x1024	1024
bn4b15_branch2c	Batch Normalization	res4b15_branch2c	res4b15_branch2cx		
res4b15	Summation	res4b14x, res4b15_branch2cx	res4b15		
res4b15_relu	ReLu	res4b15	res4b15x		
res4b16_branch2a	Convolutional	res4b15x	res4b16_branch2a	1x1x1024x256	256
bn4b16_branch2a	Batch Normalization	res4b16_branch2a	res4b16_branch2ax		
res4b16_branch2a_relu	ReLu	res4b16_branch2ax	res4b16_branch2axxx		
res4b16_branch2b	Convolutional	res4b16_branch2axxx	res4b16_branch2b	3x3x256x256	256
bn4b16_branch2b	Batch Normalization	res4b16_branch2b	res4b16_branch2bx		
res4b16_branch2b_relu	ReLu	res4b16_branch2bx	res4b16_branch2bxxx		
res4b16_branch2c	Convolutional	res4b16_branch2bxxx	res4b16_branch2c	1x1x256x1024	1024
bn4b16_branch2c	Batch Normalization	res4b16_branch2c	res4b16_branch2cx		
res4b16	Summation	res4b15x, res4b16_branch2cx	res4b16		
res4b16_relu	ReLu	res4b16	res4b16x		
res4b17_branch2a	Convolutional	res4b16x	res4b17_branch2a	1x1x1024x256	256
bn4b17_branch2a	Batch Normalization	res4b17_branch2a	res4b17_branch2ax		
res4b17_branch2a_relu	ReLu	res4b17_branch2ax	res4b17_branch2axxx		
res4b17_branch2b	Convolutional	res4b17_branch2axxx	res4b17_branch2b	3x3x256x256	256
bn4b17_branch2b	Batch Normalization	res4b17_branch2b	res4b17_branch2bx		
res4b17_branch2b_relu	ReLu	res4b17_branch2bx	res4b17_branch2bxxx		
res4b17_branch2c	Convolutional	res4b17_branch2bxxx	res4b17_branch2c	1x1x256x1024	1024
bn4b17_branch2c	Batch Normalization	res4b17_branch2c	res4b17_branch2cx		
res4b17	Summation	res4b16x, res4b17_branch2cx	res4b17		
res4b17_relu	ReLu	res4b17	res4b17x		
res4b18_branch2a	Convolutional	res4b17x	res4b18_branch2a	1x1x1024x256	256
bn4b18_branch2a	Batch Normalization	res4b18_branch2a	res4b18_branch2ax		
res4b18_branch2a_relu	ReLu	res4b18_branch2ax	res4b18_branch2axxx		
res4b18_branch2b	Convolutional	res4b18_branch2axxx	res4b18_branch2b	3x3x256x256	256
bn4b18_branch2b	Batch Normalization	res4b18_branch2b	res4b18_branch2bx		

Table 21. ResNet101 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
res4b18_branch2b_relu	ReLu	res4b18_branch2bx	res4b18_branch2bxxx		
res4b18_branch2c	Convolutional	res4b18_branch2bxxx	res4b18_branch2c	1x1x256x1024	1024
bn4b18_branch2c	Batch Normalization	res4b18_branch2c	res4b18_branch2cx		
res4b18	Summation	res4b17x, res4b18_branch2cx	res4b18		
res4b18_relu	ReLu	res4b18	res4b18x		
res4b19_branch2a	Convolutional	res4b18x	res4b19_branch2a	1x1x1024x256	256
bn4b19_branch2a	Batch Normalization	res4b19_branch2a	res4b19_branch2ax		
res4b19_branch2a_relu	ReLu	res4b19_branch2ax	res4b19_branch2axxx		
res4b19_branch2b	Convolutional	res4b19_branch2axxx	res4b19_branch2b	3x3x256x256	256
bn4b19_branch2b	Batch Normalization	res4b19_branch2b	res4b19_branch2bx		
res4b19_branch2b_relu	ReLu	res4b19_branch2bx	res4b19_branch2bxxx		
res4b19_branch2c	Convolutional	res4b19_branch2bxxx	res4b19_branch2c	1x1x256x1024	1024
bn4b19_branch2c	Batch Normalization	res4b19_branch2c	res4b19_branch2cx		
res4b19	Summation	res4b18x, res4b19_branch2cx	res4b19		
res4b19_relu	ReLu	res4b19	res4b19x		
res4b20_branch2a	Convolutional	res4b19x	res4b20_branch2a	1x1x1024x256	256
bn4b20_branch2a	Batch Normalization	res4b20_branch2a	res4b20_branch2ax		
res4b20_branch2a_relu	ReLu	res4b20_branch2ax	res4b20_branch2axxx		
res4b20_branch2b	Convolutional	res4b20_branch2axxx	res4b20_branch2b	3x3x256x256	256
bn4b20_branch2b	Batch Normalization	res4b20_branch2b	res4b20_branch2bx		
res4b20_branch2b_relu	ReLu	res4b20_branch2bx	res4b20_branch2bxxx		
res4b20_branch2c	Convolutional	res4b20_branch2bxxx	res4b20_branch2c	1x1x256x1024	1024
bn4b20_branch2c	Batch Normalization	res4b20_branch2c	res4b20_branch2cx		
res4b20	Summation	res4b19x, res4b20_branch2cx	res4b20		
res4b20_relu	ReLu	res4b20	res4b20x		
res4b21_branch2a	Convolutional	res4b20x	res4b21_branch2a	1x1x1024x256	256
bn4b21_branch2a	Batch Normalization	res4b21_branch2a	res4b21_branch2ax		
res4b21_branch2a_relu	ReLu	res4b21_branch2ax	res4b21_branch2axxx		
res4b21_branch2b	Convolutional	res4b21_branch2axxx	res4b21_branch2b	3x3x256x256	256
bn4b21_branch2b	Batch Normalization	res4b21_branch2b	res4b21_branch2bx		
res4b21_branch2b_relu	ReLu	res4b21_branch2bx	res4b21_branch2bxxx		
res4b21_branch2c	Convolutional	res4b21_branch2bxxx	res4b21_branch2c	1x1x256x1024	1024
bn4b21_branch2c	Batch Normalization	res4b21_branch2c	res4b21_branch2cx		
res4b21	Summation	res4b20x, res4b21_branch2cx	res4b21		
res4b21_relu	ReLu	res4b21	res4b21x		
res4b22_branch2a	Convolutional	res4b21x	res4b22_branch2a	1x1x1024x256	256
bn4b22_branch2a	Batch Normalization	res4b22_branch2a	res4b22_branch2ax		
res4b22_branch2a_relu	ReLu	res4b22_branch2ax	res4b22_branch2axxx		
res4b22_branch2b	Convolutional	res4b22_branch2axxx	res4b22_branch2b	3x3x256x256	256
bn4b22_branch2b	Batch Normalization	res4b22_branch2b	res4b22_branch2bx		
res4b22_branch2b_relu	ReLu	res4b22_branch2bx	res4b22_branch2bxxx		
res4b22_branch2c	Convolutional	res4b22_branch2bxxx	res4b22_branch2c	1x1x256x1024	1024
bn4b22_branch2c	Batch Normalization	res4b22_branch2c	res4b22_branch2cx		
res4b22	Summation	res4b21x, res4b22_branch2cx	res4b22		

Table 21. ResNet101 Architecture (cont'd)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
res4b22_relu	ReLu	res4b22	res4b22x		
res5a_branch1	Convolutional	res4b22x	res5a_branch1	1x1x1024x2048	2048
bn5a_branch1	Batch Normalization	res5a_branch1	res5a_branch1x		
res5a_branch2a	Convolutional	res4b22x	res5a_branch2a	1x1x1024x512	512
bn5a_branch2a	Batch Normalization	res5a_branch2a	res5a_branch2ax		
res5a_branch2a_relu	ReLu	res5a_branch2ax	res5a_branch2axxx		
res5a_branch2b	Convolutional	res5a_branch2axxx	res5a_branch2b	3x3x512x512	512
bn5a_branch2b	Batch Normalization	res5a_branch2b	res5a_branch2bx		
res5a_branch2b_relu	ReLu	res5a_branch2bx	res5a_branch2bxxx		
res5a_branch2c	Convolutional	res5a_branch2bxxx	res5a_branch2c	1x1x512x2048	2048
bn5a_branch2c	Batch Normalization	res5a_branch2c	res5a_branch2cx		
res5a	Summation	res5a_branch1x, res5a_branch2cx	res5a		
res5a_relu	ReLu	res5a	res5ax		
res5b_branch2a	Convolutional	res5ax	res5b_branch2a	1x1x2048x512	512
bn5b_branch2a	Batch Normalization	res5b_branch2a	res5b_branch2ax		
res5b_branch2a_relu	ReLu	res5b_branch2ax	res5b_branch2axxx		
res5b_branch2b	Convolutional	res5b_branch2axxx	res5b_branch2b	3x3x512x512	512
bn5b_branch2b	Batch Normalization	res5b_branch2b	res5b_branch2bx		
res5b_branch2b_relu	ReLu	res5b_branch2bx	res5b_branch2bxxx		
res5b_branch2c	Convolutional	res5b_branch2bxxx	res5b_branch2c	1x1x512x2048	2048
bn5b_branch2c	Batch Normalization	res5b_branch2c	res5b_branch2cx		
res5b	Summation	res5ax, res5b_branch2cx	res5b		
res5b_relu	ReLu	res5b	res5bx		
res5c_branch2a	Convolutional	res5bx	res5c_branch2a	1x1x2048x512	512
bn5c_branch2a	Batch Normalization	res5c_branch2a	res5c_branch2ax		
res5c_branch2a_relu	ReLu	res5c_branch2ax	res5c_branch2axxx		
res5c_branch2b	Convolutional	res5c_branch2axxx	res5c_branch2b	3x3x512x512	512
bn5c_branch2b	Batch Normalization	res5c_branch2b	res5c_branch2bx		
res5c_branch2b_relu	ReLu	res5c_branch2bx	res5c_branch2bxxx		
res5c_branch2c	Convolutional	res5c_branch2bxxx	res5c_branch2c	1x1x512x2048	2048
bn5c_branch2c	Batch Normalization	res5c_branch2c	res5c_branch2cx		
res5c	Summation	res5bx, res5c_branch2cx	res5c		
res5c_relu	ReLu	res5c	res5cx		
pool5	Average Pooling	res5cx	pool5		
fc1000	Convolutional	pool5	fc1000	1x1x2048x1000	1000
prob	SoftMax	fc1000	prob		

Table 22. ResNet152 Architecture

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
conv1	Convolution	data	conv1	7x7x3x64	64
bn_conv1	Batch Normalization	conv1	conv1x		
conv1_relu	ReLu	conv1x	conv1xxx		
pool1	Max Pooling	conv1xxx	pool1		
res2a_branch1	Convolution	pool1	res2a_branch1	1x1x64x256	256
bn2a_branch1	Batch Normalization	res2a_branch1	res2a_branch1x		
res2a_branch2a	Convolution	pool1	res2a_branch2a	1x1x64x64	64
bn2a_branch2a	Batch Normalization	res2a_branch2a	res2a_branch2ax		
res2a_branch2a_relu	ReLu	res2a_branch2ax	res2a_branch2axxx		
res2a_branch2b	Convolution	res2a_branch2axxx	res2a_branch2b	3x3x64x64	64
bn2a_branch2b	Batch Normalization	res2a_branch2b	res2a_branch2bx		
res2a_branch2b_relu	ReLu	res2a_branch2bx	res2a_branch2bxxx		
res2a_branch2c	Convolution	res2a_branch2bxxx	res2a_branch2c	1x1x64x256	256
bn2a_branch2c	Batch Normalization	res2a_branch2c	res2a_branch2cx		
res2a	Summation	res2a_branch1x, res2a_branch2cx	res2a		
res2a_relu	ReLu	res2a	res2ax		
res2b_branch2a	Convolution	res2ax	res2b_branch2a	1x1x256x64	64
bn2b_branch2a	Batch Normalization	res2b_branch2a	res2b_branch2ax		
res2b_branch2a_relu	ReLu	res2b_branch2ax	res2b_branch2axxx		
res2b_branch2b	Convolution	res2b_branch2axxx	res2b_branch2b	3x3x64x64	64
bn2b_branch2b	Batch Normalization	res2b_branch2b	res2b_branch2bx		
res2b_branch2b_relu	ReLu	res2b_branch2bx	res2b_branch2bxxx		
res2b_branch2c	Convolution	res2b_branch2bxxx	res2b_branch2c	1x1x64x256	256
bn2b_branch2c	Batch Normalization	res2b_branch2c	res2b_branch2cx		
res2b	Summation	res2ax, res2b_branch2cx	res2b		
res2b_relu	ReLu	res2b	res2bx		
res2c_branch2a	Convolution	res2bx	res2c_branch2a	1x1x256x64	64
bn2c_branch2a	Batch Normalization	res2c_branch2a	res2c_branch2ax		
res2c_branch2a_relu	ReLu	res2c_branch2ax	res2c_branch2axxx		
res2c_branch2b	Convolution	res2c_branch2axxx	res2c_branch2b	3x3x64x64	64
bn2c_branch2b	Batch Normalization	res2c_branch2b	res2c_branch2bx		
res2c_branch2b_relu	ReLu	res2c_branch2bx	res2c_branch2bxxx		
res2c_branch2c	Convolution	res2c_branch2bxxx	res2c_branch2c	1x1x64x256	256
bn2c_branch2c	Batch Normalization	res2c_branch2c	res2c_branch2cx		
res2c	Summation	res2bx, res2c_branch2cx	res2c		
res2c_relu	ReLu	res2c	res2cx		
res3a_branch1	Convolution	res2cx	res3a_branch1	1x1x256x512	512
bn3a_branch1	Batch Normalization	res3a_branch1	res3a_branch1x		
res3a_branch2a	Convolution	res2cx	res3a_branch2a	1x1x256x128	128
bn3a_branch2a	Batch Normalization	res3a_branch2a	res3a_branch2ax		
res3a_branch2a_relu	ReLu	res3a_branch2ax	res3a_branch2axxx		
res3a_branch2b	Convolution	res3a_branch2axxx	res3a_branch2b	3x3x128x128	128
bn3a_branch2b	Batch Normalization	res3a_branch2b	res3a_branch2bx		
res3a_branch2b_relu	ReLu	res3a_branch2bx	res3a_branch2bxxx		
res3a_branch2c	Convolution	res3a_branch2bxxx	res3a_branch2c	1x1x128x512	512

Table 22. ResNet152 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
bn3a_branch2c	Batch Normalization	res3a_branch2c	res3a_branch2cx		
res3a	Summation	res3a_branch1x, res3a_branch2cx	res3a		
res3a_relu	ReLu	res3a	res3ax		
res3b1_branch2a	Convolution	res3ax	res3b1_branch2a	1x1x512x128	128
bn3b1_branch2a	Batch Normalization	res3b1_branch2a	res3b1_branch2ax		
res3b1_branch2a_relu	ReLu	res3b1_branch2ax	res3b1_branch2axxx		
res3b1_branch2b	Convolution	res3b1_branch2axxx	res3b1_branch2b	3x3x128x128	128
bn3b1_branch2b	Batch Normalization	res3b1_branch2b	res3b1_branch2bx		
res3b1_branch2b_relu	ReLu	res3b1_branch2bx	res3b1_branch2bxxx		
res3b1_branch2c	Convolution	res3b1_branch2bxxx	res3b1_branch2c	1x1x128x512	512
bn3b1_branch2c	Batch Normalization	res3b1_branch2c	res3b1_branch2cx		
res3b1	Summation	res3ax, res3b1_branch2cx	res3b1		
res3b1_relu	ReLu	res3b1	res3b1x		
res3b2_branch2a	Convolution	res3b1x	res3b2_branch2a	1x1x512x128	128
bn3b2_branch2a	Batch Normalization	res3b2_branch2a	res3b2_branch2ax		
res3b2_branch2a_relu	ReLu	res3b2_branch2ax	res3b2_branch2axxx		
res3b2_branch2b	Convolution	res3b2_branch2axxx	res3b2_branch2b	3x3x128x128	128
bn3b2_branch2b	Batch Normalization	res3b2_branch2b	res3b2_branch2bx		
res3b2_branch2b_relu	ReLu	res3b2_branch2bx	res3b2_branch2bxxx		
res3b2_branch2c	Convolution	res3b2_branch2bxxx	res3b2_branch2c	1x1x128x512	512
bn3b2_branch2c	Batch Normalization	res3b2_branch2c	res3b2_branch2cx		
res3b2	Summation	res3b1x, res3b2_branch2cx	res3b2		
res3b2_relu	ReLu	res3b2	res3b2x		
res3b3_branch2a	Convolution	res3b2x	res3b3_branch2a	1x1x512x128	128
bn3b3_branch2a	Batch Normalization	res3b3_branch2a	res3b3_branch2ax		
res3b3_branch2a_relu	ReLu	res3b3_branch2ax	res3b3_branch2axxx		
res3b3_branch2b	Convolution	res3b3_branch2axxx	res3b3_branch2b	3x3x128x128	128
bn3b3_branch2b	Batch Normalization	res3b3_branch2b	res3b3_branch2bx		
res3b3_branch2b_relu	ReLu	res3b3_branch2bx	res3b3_branch2bxxx		
res3b3_branch2c	Convolution	res3b3_branch2bxxx	res3b3_branch2c	1x1x128x512	512
bn3b3_branch2c	Batch Normalization	res3b3_branch2c	res3b3_branch2cx		
res3b3	Summation	res3b2x, res3b3_branch2cx	res3b3		
res3b3_relu	ReLu	res3b3	res3b3x		
res3b4_branch2a	Convolution	res3b3x	res3b4_branch2a	1x1x512x128	128
bn3b4_branch2a	Batch Normalization	res3b4_branch2a	res3b4_branch2ax		
res3b4_branch2a_relu	ReLu	res3b4_branch2ax	res3b4_branch2axxx		
res3b4_branch2b	Convolution	res3b4_branch2axxx	res3b4_branch2b	3x3x128x128	128
bn3b4_branch2b	Batch Normalization	res3b4_branch2b	res3b4_branch2bx		
res3b4_branch2b_relu	ReLu	res3b4_branch2bx	res3b4_branch2bxxx		
res3b4_branch2c	Convolution	res3b4_branch2bxxx	res3b4_branch2c	1x1x128x512	512
bn3b4_branch2c	Batch Normalization	res3b4_branch2c	res3b4_branch2cx		
res3b4	Summation	res3b3x, res3b4_branch2cx	res3b4		
res3b4_relu	ReLu	res3b4	res3b4x		
res3b5_branch2a	Convolution	res3b4x	res3b5_branch2a	1x1x512x128	128

Table 22. ResNet152 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
bn3b5_branch2a	Batch Normalization	res3b5_branch2a	res3b5_branch2ax		
res3b5_branch2a_relu	ReLu	res3b5_branch2ax	res3b5_branch2axxx		
res3b5_branch2b	Convolution	res3b5_branch2axxx	res3b5_branch2b	3x3x128x128	128
bn3b5_branch2b	Batch Normalization	res3b5_branch2b	res3b5_branch2bx		
res3b5_branch2b_relu	ReLu	res3b5_branch2bx	res3b5_branch2bxxx		
res3b5_branch2c	Convolution	res3b5_branch2bxxx	res3b5_branch2c	1x1x128x512	512
bn3b5_branch2c	Batch Normalization	res3b5_branch2c	res3b5_branch2cx		
res3b5	Summation	res3b4x, res3b5_branch2cx	res3b5		
res3b5_relu	ReLu	res3b5	res3b5x		
res3b6_branch2a	Convolution	res3b5x	res3b6_branch2a	1x1x512x128	128
bn3b6_branch2a	Batch Normalization	res3b6_branch2a	res3b6_branch2ax		
res3b6_branch2a_relu	ReLu	res3b6_branch2ax	res3b6_branch2axxx		
res3b6_branch2b	Convolution	res3b6_branch2axxx	res3b6_branch2b	3x3x128x128	128
bn3b6_branch2b	Batch Normalization	res3b6_branch2b	res3b6_branch2bx		
res3b6_branch2b_relu	ReLu	res3b6_branch2bx	res3b6_branch2bxxx		
res3b6_branch2c	Convolution	res3b6_branch2bxxx	res3b6_branch2c	1x1x128x512	512
bn3b6_branch2c	Batch Normalization	res3b6_branch2c	res3b6_branch2cx		
res3b6	Summation	res3b5x, res3b6_branch2cx	res3b6		
res3b6_relu	ReLu	res3b6	res3b6x		
res3b7_branch2a	Convolution	res3b6x	res3b7_branch2a	1x1x512x128	128
bn3b7_branch2a	Batch Normalization	res3b7_branch2a	res3b7_branch2ax		
res3b7_branch2a_relu	ReLu	res3b7_branch2ax	res3b7_branch2axxx		
res3b7_branch2b	Convolution	res3b7_branch2axxx	res3b7_branch2b	3x3x128x128	128
bn3b7_branch2b	Batch Normalization	res3b7_branch2b	res3b7_branch2bx		
res3b7_branch2b_relu	ReLu	res3b7_branch2bx	res3b7_branch2bxxx		
res3b7_branch2c	Convolution	res3b7_branch2bxxx	res3b7_branch2c	1x1x128x512	512
bn3b7_branch2c	Batch Normalization	res3b7_branch2c	res3b7_branch2cx		
res3b7	Summation	res3b6x, res3b7_branch2cx	res3b7		
res3b7_relu	ReLu	res3b7	res3b7x		
res4a_branch1	Convolution	res3b7x	res4a_branch1	1x1x512x1024	1024
bn4a_branch1	Batch Normalization	res4a_branch1	res4a_branch1x		
res4a_branch2a	Convolution	res3b7x	res4a_branch2a	1x1x512x256	256
bn4a_branch2a	Batch Normalization	res4a_branch2a	res4a_branch2ax		
res4a_branch2a_relu	ReLu	res4a_branch2ax	res4a_branch2axxx		
res4a_branch2b	Convolution	res4a_branch2axxx	res4a_branch2b	3x3x256x256	256
bn4a_branch2b	Batch Normalization	res4a_branch2b	res4a_branch2bx		
res4a_branch2b_relu	ReLu	res4a_branch2bx	res4a_branch2bxxx		
res4a_branch2c	Convolution	res4a_branch2bxxx	res4a_branch2c	1x1x256x1024	1024
bn4a_branch2c	Batch Normalization	res4a_branch2c	res4a_branch2cx		
res4a	Summation	res4a_branch1x, res4a_branch2cx	res4a		
res4a_relu	ReLu	res4a	res4ax		
res4b1_branch2a	Convolution	res4ax	res4b1_branch2a	1x1x1024x256	256
bn4b1_branch2a	Batch Normalization	res4b1_branch2a	res4b1_branch2ax		
res4b1_branch2a_relu	ReLu	res4b1_branch2ax	res4b1_branch2axxx		

Table 22. ResNet152 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
res4b1_branch2b	Convolution	res4b1_branch2axxx	res4b1_branch2b	3x3x256x256	256
bn4b1_branch2b	Batch Normalization	res4b1_branch2b	res4b1_branch2bx		
res4b1_branch2b_relu	ReLu	res4b1_branch2bx	res4b1_branch2bxxx		
res4b1_branch2c	Convolution	res4b1_branch2bxxx	res4b1_branch2c	1x1x256x1024	1024
bn4b1_branch2c	Batch Normalization	res4b1_branch2c	res4b1_branch2cx		
res4b1	Summation	res4ax, res4b1_branch2cx	res4b1		
res4b1_relu	ReLu	res4b1	res4b1x		
res4b2_branch2a	Convolution	res4b1x	res4b2_branch2a	1x1x1024x256	256
bn4b2_branch2a	Batch Normalization	res4b2_branch2a	res4b2_branch2ax		
res4b2_branch2a_relu	ReLu	res4b2_branch2ax	res4b2_branch2axxx		
res4b2_branch2b	Convolution	res4b2_branch2axxx	res4b2_branch2b	3x3x256x256	256
bn4b2_branch2b	Batch Normalization	res4b2_branch2b	res4b2_branch2bx		
res4b2_branch2b_relu	ReLu	res4b2_branch2bx	res4b2_branch2bxxx		
res4b2_branch2c	Convolution	res4b2_branch2bxxx	res4b2_branch2c	1x1x256x1024	1024
bn4b2_branch2c	Batch Normalization	res4b2_branch2c	res4b2_branch2cx		
res4b2	Summation	res4b1x, res4b2_branch2cx	res4b2		
res4b2_relu	ReLu	res4b2	res4b2x		
res4b3_branch2a	Convolution	res4b2x	res4b3_branch2a	1x1x1024x256	256
bn4b3_branch2a	Batch Normalization	res4b3_branch2a	res4b3_branch2ax		
res4b3_branch2a_relu	ReLu	res4b3_branch2ax	res4b3_branch2axxx		
res4b3_branch2b	Convolution	res4b3_branch2axxx	res4b3_branch2b	3x3x256x256	256
bn4b3_branch2b	Batch Normalization	res4b3_branch2b	res4b3_branch2bx		
res4b3_branch2b_relu	ReLu	res4b3_branch2bx	res4b3_branch2bxxx		
res4b3_branch2c	Convolution	res4b3_branch2bxxx	res4b3_branch2c	1x1x256x1024	1024
bn4b3_branch2c	Batch Normalization	res4b3_branch2c	res4b3_branch2cx		
res4b3	Summation	res4b2x, res4b3_branch2cx	res4b3		
res4b3_relu	ReLu	res4b3	res4b3x		
res4b4_branch2a	Convolution	res4b3x	res4b4_branch2a	1x1x1024x256	256
bn4b4_branch2a	Batch Normalization	res4b4_branch2a	res4b4_branch2ax		
res4b4_branch2a_relu	ReLu	res4b4_branch2ax	res4b4_branch2axxx		
res4b4_branch2b	Convolution	res4b4_branch2axxx	res4b4_branch2b	3x3x256x256	256
bn4b4_branch2b	Batch Normalization	res4b4_branch2b	res4b4_branch2bx		
res4b4_branch2b_relu	ReLu	res4b4_branch2bx	res4b4_branch2bxxx		
res4b4_branch2c	Convolution	res4b4_branch2bxxx	res4b4_branch2c	1x1x256x1024	1024
bn4b4_branch2c	Batch Normalization	res4b4_branch2c	res4b4_branch2cx		
res4b4	Summation	res4b3x, res4b4_branch2cx	res4b4		
res4b4_relu	ReLu	res4b4	res4b4x		
res4b5_branch2a	Convolution	res4b4x	res4b5_branch2a	1x1x1024x256	256
bn4b5_branch2a	Batch Normalization	res4b5_branch2a	res4b5_branch2ax		
res4b5_branch2a_relu	ReLu	res4b5_branch2ax	res4b5_branch2axxx		
res4b5_branch2b	Convolution	res4b5_branch2axxx	res4b5_branch2b	3x3x256x256	256
bn4b5_branch2b	Batch Normalization	res4b5_branch2b	res4b5_branch2bx		
res4b5_branch2b_relu	ReLu	res4b5_branch2bx	res4b5_branch2bxxx		
res4b5_branch2c	Convolution	res4b5_branch2bxxx	res4b5_branch2c	1x1x256x1024	1024

Table 22. ResNet152 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
bn4b5_branch2c	Batch Normalization	res4b5_branch2c	res4b5_branch2cx		
res4b5	Summation	res4b4x, res4b5_branch2cx	res4b5		
res4b5_relu	ReLu	res4b5	res4b5x		
res4b6_branch2a	Convolution	res4b5x	res4b6_branch2a	1x1x1024x256	256
bn4b6_branch2a	Batch Normalization	res4b6_branch2a	res4b6_branch2ax		
res4b6_branch2a_relu	ReLu	res4b6_branch2ax	res4b6_branch2axxx		
res4b6_branch2b	Convolution	res4b6_branch2axxx	res4b6_branch2b	3x3x256x256	256
bn4b6_branch2b	Batch Normalization	res4b6_branch2b	res4b6_branch2bx		
res4b6_branch2b_relu	ReLu	res4b6_branch2bx	res4b6_branch2bxxx		
res4b6_branch2c	Convolution	res4b6_branch2bxxx	res4b6_branch2c	1x1x256x1024	1024
bn4b6_branch2c	Batch Normalization	res4b6_branch2c	res4b6_branch2cx		
res4b6	Summation	res4b5x, res4b6_branch2cx	res4b6		
res4b6_relu	ReLu	res4b6	res4b6x		
res4b7_branch2a	Convolution	res4b6x	res4b7_branch2a	1x1x1024x256	256
bn4b7_branch2a	Batch Normalization	res4b7_branch2a	res4b7_branch2ax		
res4b7_branch2a_relu	ReLu	res4b7_branch2ax	res4b7_branch2axxx		
res4b7_branch2b	Convolution	res4b7_branch2axxx	res4b7_branch2b	3x3x256x256	256
bn4b7_branch2b	Batch Normalization	res4b7_branch2b	res4b7_branch2bx		
res4b7_branch2b_relu	ReLu	res4b7_branch2bx	res4b7_branch2bxxx		
res4b7_branch2c	Convolution	res4b7_branch2bxxx	res4b7_branch2c	1x1x256x1024	1024
bn4b7_branch2c	Batch Normalization	res4b7_branch2c	res4b7_branch2cx		
res4b7	Summation	res4b6x, res4b7_branch2cx	res4b7		
res4b7_relu	ReLu	res4b7	res4b7x		
res4b8_branch2a	Convolution	res4b7x	res4b8_branch2a	1x1x1024x256	256
bn4b8_branch2a	Batch Normalization	res4b8_branch2a	res4b8_branch2ax		
res4b8_branch2a_relu	ReLu	res4b8_branch2ax	res4b8_branch2axxx		
res4b8_branch2b	Convolution	res4b8_branch2axxx	res4b8_branch2b	3x3x256x256	256
bn4b8_branch2b	Batch Normalization	res4b8_branch2b	res4b8_branch2bx		
res4b8_branch2b_relu	ReLu	res4b8_branch2bx	res4b8_branch2bxxx		
res4b8_branch2c	Convolution	res4b8_branch2bxxx	res4b8_branch2c	1x1x256x1024	1024
bn4b8_branch2c	Batch Normalization	res4b8_branch2c	res4b8_branch2cx		
res4b8	Summation	res4b7x, res4b8_branch2cx	res4b8		
res4b8_relu	ReLu	res4b8	res4b8x		
res4b9_branch2a	Convolution	res4b8x	res4b9_branch2a	1x1x1024x256	256
bn4b9_branch2a	Batch Normalization	res4b9_branch2a	res4b9_branch2ax		
res4b9_branch2a_relu	ReLu	res4b9_branch2ax	res4b9_branch2axxx		
res4b9_branch2b	Convolution	res4b9_branch2axxx	res4b9_branch2b	3x3x256x256	256
bn4b9_branch2b	Batch Normalization	res4b9_branch2b	res4b9_branch2bx		
res4b9_branch2b_relu	ReLu	res4b9_branch2bx	res4b9_branch2bxxx		
res4b9_branch2c	Convolution	res4b9_branch2bxxx	res4b9_branch2c	1x1x256x1024	1024
bn4b9_branch2c	Batch Normalization	res4b9_branch2c	res4b9_branch2cx		
res4b9	Summation	res4b8x, res4b9_branch2cx	res4b9		
res4b9_relu	ReLu	res4b9	res4b9x		
res4b10_branch2a	Convolution	res4b9x	res4b10_branch2a	1x1x1024x256	256

Table 22. ResNet152 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
bn4b10_branch2a	Batch Normalization	res4b10_branch2a	res4b10_branch2ax		
res4b10_branch2a_relu	ReLu	res4b10_branch2ax	res4b10_branch2axxx		
res4b10_branch2b	Convolution	res4b10_branch2axxx	res4b10_branch2b	3x3x256x256	256
bn4b10_branch2b	Batch Normalization	res4b10_branch2b	res4b10_branch2bx		
res4b10_branch2b_relu	ReLu	res4b10_branch2bx	res4b10_branch2bxxx		
res4b10_branch2c	Convolution	res4b10_branch2bxxx	res4b10_branch2c	1x1x256x1024	1024
bn4b10_branch2c	Batch Normalization	res4b10_branch2c	res4b10_branch2cx		
res4b10	Summation	res4b9x, res4b10_branch2cx	res4b10		
res4b10_relu	ReLu	res4b10	res4b10x		
res4b11_branch2a	Convolution	res4b10x	res4b11_branch2a	1x1x1024x256	256
bn4b11_branch2a	Batch Normalization	res4b11_branch2a	res4b11_branch2ax		
res4b11_branch2a_relu	ReLu	res4b11_branch2ax	res4b11_branch2axxx		
res4b11_branch2b	Convolution	res4b11_branch2axxx	res4b11_branch2b	3x3x256x256	256
bn4b11_branch2b	Batch Normalization	res4b11_branch2b	res4b11_branch2bx		
res4b11_branch2b_relu	ReLu	res4b11_branch2bx	res4b11_branch2bxxx		
res4b11_branch2c	Convolution	res4b11_branch2bxxx	res4b11_branch2c	1x1x256x1024	1024
bn4b11_branch2c	Batch Normalization	res4b11_branch2c	res4b11_branch2cx		
res4b11	Summation	res4b10x, res4b11_branch2cx	res4b11		
res4b11_relu	ReLu	res4b11	res4b11x		
res4b12_branch2a	Convolution	res4b11x	res4b12_branch2a	1x1x1024x256	256
bn4b12_branch2a	Batch Normalization	res4b12_branch2a	res4b12_branch2ax		
res4b12_branch2a_relu	ReLu	res4b12_branch2ax	res4b12_branch2axxx		
res4b12_branch2b	Convolution	res4b12_branch2axxx	res4b12_branch2b	3x3x256x256	256
bn4b12_branch2b	Batch Normalization	res4b12_branch2b	res4b12_branch2bx		
res4b12_branch2b_relu	ReLu	res4b12_branch2bx	res4b12_branch2bxxx		
res4b12_branch2c	Convolution	res4b12_branch2bxxx	res4b12_branch2c	1x1x256x1024	1024
bn4b12_branch2c	Batch Normalization	res4b12_branch2c	res4b12_branch2cx		
res4b12	Summation	res4b11x, res4b12_branch2cx	res4b12		
res4b12_relu	ReLu	res4b12	res4b12x		
res4b13_branch2a	Convolution	res4b12x	res4b13_branch2a	1x1x1024x256	256
bn4b13_branch2a	Batch Normalization	res4b13_branch2a	res4b13_branch2ax		
res4b13_branch2a_relu	ReLu	res4b13_branch2ax	res4b13_branch2axxx		
res4b13_branch2b	Convolution	res4b13_branch2axxx	res4b13_branch2b	3x3x256x256	256
bn4b13_branch2b	Batch Normalization	res4b13_branch2b	res4b13_branch2bx		
res4b13_branch2b_relu	ReLu	res4b13_branch2bx	res4b13_branch2bxxx		
res4b13_branch2c	Convolution	res4b13_branch2bxxx	res4b13_branch2c	1x1x256x1024	1024
bn4b13_branch2c	Batch Normalization	res4b13_branch2c	res4b13_branch2cx		
res4b13	Summation	res4b12x, res4b13_branch2cx	res4b13		
res4b13_relu	ReLu	res4b13	res4b13x		
res4b14_branch2a	Convolution	res4b13x	res4b14_branch2a	1x1x1024x256	256
bn4b14_branch2a	Batch Normalization	res4b14_branch2a	res4b14_branch2ax		
res4b14_branch2a_relu	ReLu	res4b14_branch2ax	res4b14_branch2axxx		
res4b14_branch2b	Convolution	res4b14_branch2axxx	res4b14_branch2b	3x3x256x256	256
bn4b14_branch2b	Batch Normalization	res4b14_branch2b	res4b14_branch2bx		

Table 22. ResNet152 Architecture (cont'd)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
res4b14_branch2b_relu	ReLu	res4b14_branch2bx	res4b14_branch2bxxx		
res4b14_branch2c	Convolution	res4b14_branch2bxxx	res4b14_branch2c	1x1x256x1024	1024
bn4b14_branch2c	Batch Normalization	res4b14_branch2c	res4b14_branch2cx		
res4b14	Summation	res4b13x, res4b14_branch2cx	res4b14		
res4b14_relu	ReLu	res4b14	res4b14x		
res4b15_branch2a	Convolution	res4b14x	res4b15_branch2a	1x1x1024x256	256
bn4b15_branch2a	Batch Normalization	res4b15_branch2a	res4b15_branch2ax		
res4b15_branch2a_relu	ReLu	res4b15_branch2ax	res4b15_branch2axxx		
res4b15_branch2b	Convolution	res4b15_branch2axxx	res4b15_branch2b	3x3x256x256	256
bn4b15_branch2b	Batch Normalization	res4b15_branch2b	res4b15_branch2bx		
res4b15_branch2b_relu	ReLu	res4b15_branch2bx	res4b15_branch2bxxx		
res4b15_branch2c	Convolution	res4b15_branch2bxxx	res4b15_branch2c	1x1x256x1024	1024
bn4b15_branch2c	Batch Normalization	res4b15_branch2c	res4b15_branch2cx		
res4b15	Summation	res4b14x, res4b15_branch2cx	res4b15		
res4b15_relu	ReLu	res4b15	res4b15x		
res4b16_branch2a	Convolution	res4b15x	res4b16_branch2a	1x1x1024x256	256
bn4b16_branch2a	Batch Normalization	res4b16_branch2a	res4b16_branch2ax		
res4b16_branch2a_relu	ReLu	res4b16_branch2ax	res4b16_branch2axxx		
res4b16_branch2b	Convolution	res4b16_branch2axxx	res4b16_branch2b	3x3x256x256	256
bn4b16_branch2b	Batch Normalization	res4b16_branch2b	res4b16_branch2bx		
res4b16_branch2b_relu	ReLu	res4b16_branch2bx	res4b16_branch2bxxx		
res4b16_branch2c	Convolution	res4b16_branch2bxxx	res4b16_branch2c	1x1x256x1024	1024
bn4b16_branch2c	Batch Normalization	res4b16_branch2c	res4b16_branch2cx		
res4b16	Summation	res4b15x, res4b16_branch2cx	res4b16		
res4b16_relu	ReLu	res4b16	res4b16x		
res4b17_branch2a	Convolution	res4b16x	res4b17_branch2a	1x1x1024x256	256
bn4b17_branch2a	Batch Normalization	res4b17_branch2a	res4b17_branch2ax		
res4b17_branch2a_relu	ReLu	res4b17_branch2ax	res4b17_branch2axxx		
res4b17_branch2b	Convolution	res4b17_branch2axxx	res4b17_branch2b	3x3x256x256	256
bn4b17_branch2b	Batch Normalization	res4b17_branch2b	res4b17_branch2bx		
res4b17_branch2b_relu	ReLu	res4b17_branch2bx	res4b17_branch2bxxx		
res4b17_branch2c	Convolution	res4b17_branch2bxxx	res4b17_branch2c	1x1x256x1024	1024
bn4b17_branch2c	Batch Normalization	res4b17_branch2c	res4b17_branch2cx		
res4b17	Summation	res4b16x, res4b17_branch2cx	res4b17		
res4b17_relu	ReLu	res4b17	res4b17x		
res4b18_branch2a	Convolution	res4b17x	res4b18_branch2a	1x1x1024x256	256
bn4b18_branch2a	Batch Normalization	res4b18_branch2a	res4b18_branch2ax		
res4b18_branch2a_relu	ReLu	res4b18_branch2ax	res4b18_branch2axxx		
res4b18_branch2b	Convolution	res4b18_branch2axxx	res4b18_branch2b	3x3x256x256	256
bn4b18_branch2b	Batch Normalization	res4b18_branch2b	res4b18_branch2bx		
res4b18_branch2b_relu	ReLu	res4b18_branch2bx	res4b18_branch2bxxx		
res4b18_branch2c	Convolution	res4b18_branch2bxxx	res4b18_branch2c	1x1x256x1024	1024
bn4b18_branch2c	Batch Normalization	res4b18_branch2c	res4b18_branch2cx		

Table 22. ResNet152 Architecture (cont'd)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
res4b18	Summation	res4b17x, res4b18_branch2cx	res4b18		
res4b18_relu	ReLu	res4b18	res4b18x		
res4b19_branch2a	Convolution	res4b18x	res4b19_branch2a	1x1x1024x256	256
bn4b19_branch2a	Batch Normalization	res4b19_branch2a	res4b19_branch2ax		
res4b19_branch2a_relu	ReLu	res4b19_branch2ax	res4b19_branch2axxx		
res4b19_branch2b	Convolution	res4b19_branch2axxx	res4b19_branch2b	3x3x256x256	256
bn4b19_branch2b	Batch Normalization	res4b19_branch2b	res4b19_branch2bx		
res4b19_branch2b_relu	ReLu	res4b19_branch2bx	res4b19_branch2bxxx		
res4b19_branch2c	Convolution	res4b19_branch2bxxx	res4b19_branch2c	1x1x256x1024	1024
bn4b19_branch2c	Batch Normalization	res4b19_branch2c	res4b19_branch2cx		
res4b19	Summation	res4b18x, res4b19_branch2cx	res4b19		
res4b19_relu	ReLu	res4b19	res4b19x		
res4b20_branch2a	Convolution	res4b19x	res4b20_branch2a	1x1x1024x256	256
bn4b20_branch2a	Batch Normalization	res4b20_branch2a	res4b20_branch2ax		
res4b20_branch2a_relu	ReLu	res4b20_branch2ax	res4b20_branch2axxx		
res4b20_branch2b	Convolution	res4b20_branch2axxx	res4b20_branch2b	3x3x256x256	256
bn4b20_branch2b	Batch Normalization	res4b20_branch2b	res4b20_branch2bx		
res4b20_branch2b_relu	ReLu	res4b20_branch2bx	res4b20_branch2bxxx		
res4b20_branch2c	Convolution	res4b20_branch2bxxx	res4b20_branch2c	1x1x256x1024	1024
bn4b20_branch2c	Batch Normalization	res4b20_branch2c	res4b20_branch2cx		
res4b20	Summation	res4b19x, res4b20_branch2cx	res4b20		
res4b20_relu	ReLu	res4b20	res4b20x		
res4b21_branch2a	Convolution	res4b20x	res4b21_branch2a	1x1x1024x256	256
bn4b21_branch2a	Batch Normalization	res4b21_branch2a	res4b21_branch2ax		
res4b21_branch2a_relu	ReLu	res4b21_branch2ax	res4b21_branch2axxx		
res4b21_branch2b	Convolution	res4b21_branch2axxx	res4b21_branch2b	3x3x256x256	256
bn4b21_branch2b	Batch Normalization	res4b21_branch2b	res4b21_branch2bx		
res4b21_branch2b_relu	ReLu	res4b21_branch2bx	res4b21_branch2bxxx		
res4b21_branch2c	Convolution	res4b21_branch2bxxx	res4b21_branch2c	1x1x256x1024	1024
bn4b21_branch2c	Batch Normalization	res4b21_branch2c	res4b21_branch2cx		
res4b21	Summation	res4b20x, res4b21_branch2cx	res4b21		
res4b21_relu	ReLu	res4b21	res4b21x		
res4b22_branch2a	Convolution	res4b21x	res4b22_branch2a	1x1x1024x256	256
bn4b22_branch2a	Batch Normalization	res4b22_branch2a	res4b22_branch2ax		
res4b22_branch2a_relu	ReLu	res4b22_branch2ax	res4b22_branch2axxx		
res4b22_branch2b	Convolution	res4b22_branch2axxx	res4b22_branch2b	3x3x256x256	256
bn4b22_branch2b	Batch Normalization	res4b22_branch2b	res4b22_branch2bx		
res4b22_branch2b_relu	ReLu	res4b22_branch2bx	res4b22_branch2bxxx		
res4b22_branch2c	Convolution	res4b22_branch2bxxx	res4b22_branch2c	1x1x256x1024	1024
bn4b22_branch2c	Batch Normalization	res4b22_branch2c	res4b22_branch2cx		
res4b22	Summation	res4b21x, res4b22_branch2cx	res4b22		
res4b22_relu	ReLu	res4b22	res4b22x		
res4b23_branch2a	Convolution	res4b22x	res4b23_branch2a	1x1x1024x256	256
bn4b23_branch2a	Batch Normalization	res4b23_branch2a	res4b23_branch2ax		

Table 22. ResNet152 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
res4b23_branch2a_relu	ReLu	res4b23_branch2ax	res4b23_branch2axxx		
res4b23_branch2b	Convolution	res4b23_branch2axxx	res4b23_branch2b	3x3x256x256	256
bn4b23_branch2b	Batch Normalization	res4b23_branch2b	res4b23_branch2bx		
res4b23_branch2b_relu	ReLu	res4b23_branch2bx	res4b23_branch2bxxx		
res4b23_branch2c	Convolution	res4b23_branch2bxxx	res4b23_branch2c	1x1x256x1024	1024
bn4b23_branch2c	Batch Normalization	res4b23_branch2c	res4b23_branch2cx		
res4b23	Summation	res4b22x, res4b23_branch2cx	res4b23		
res4b23_relu	ReLu	res4b23	res4b23x		
res4b24_branch2a	Convolution	res4b23x	res4b24_branch2a	1x1x1024x256	256
bn4b24_branch2a	Batch Normalization	res4b24_branch2a	res4b24_branch2ax		
res4b24_branch2a_relu	ReLu	res4b24_branch2ax	res4b24_branch2axxx		
res4b24_branch2b	Convolution	res4b24_branch2axxx	res4b24_branch2b	3x3x256x256	256
bn4b24_branch2b	Batch Normalization	res4b24_branch2b	res4b24_branch2bx		
res4b24_branch2b_relu	ReLu	res4b24_branch2bx	res4b24_branch2bxxx		
res4b24_branch2c	Convolution	res4b24_branch2bxxx	res4b24_branch2c	1x1x256x1024	1024
bn4b24_branch2c	Batch Normalization	res4b24_branch2c	res4b24_branch2cx		
res4b24	Summation	res4b23x, res4b24_branch2cx	res4b24		
res4b24_relu	ReLu	res4b24	res4b24x		
res4b25_branch2a	Convolution	res4b24x	res4b25_branch2a	1x1x1024x256	256
bn4b25_branch2a	Batch Normalization	res4b25_branch2a	res4b25_branch2ax		
res4b25_branch2a_relu	ReLu	res4b25_branch2ax	res4b25_branch2axxx		
res4b25_branch2b	Convolution	res4b25_branch2axxx	res4b25_branch2b	3x3x256x256	256
bn4b25_branch2b	Batch Normalization	res4b25_branch2b	res4b25_branch2bx		
res4b25_branch2b_relu	ReLu	res4b25_branch2bx	res4b25_branch2bxxx		
res4b25_branch2c	Convolution	res4b25_branch2bxxx	res4b25_branch2c	1x1x256x1024	1024
bn4b25_branch2c	Batch Normalization	res4b25_branch2c	res4b25_branch2cx		
res4b25	Summation	res4b24x, res4b25_branch2cx	res4b25		
res4b25_relu	ReLu	res4b25	res4b25x		
res4b26_branch2a	Convolution	res4b25x	res4b26_branch2a	1x1x1024x256	256
bn4b26_branch2a	Batch Normalization	res4b26_branch2a	res4b26_branch2ax		
res4b26_branch2a_relu	ReLu	res4b26_branch2ax	res4b26_branch2axxx		
res4b26_branch2b	Convolution	res4b26_branch2axxx	res4b26_branch2b	3x3x256x256	256
bn4b26_branch2b	Batch Normalization	res4b26_branch2b	res4b26_branch2bx		
res4b26_branch2b_relu	ReLu	res4b26_branch2bx	res4b26_branch2bxxx		
res4b26_branch2c	Convolution	res4b26_branch2bxxx	res4b26_branch2c	1x1x256x1024	1024
bn4b26_branch2c	Batch Normalization	res4b26_branch2c	res4b26_branch2cx		
res4b26	Summation	res4b25x, res4b26_branch2cx	res4b26		
res4b26_relu	ReLu	res4b26	res4b26x		
res4b27_branch2a	Convolution	res4b26x	res4b27_branch2a	1x1x1024x256	256
bn4b27_branch2a	Batch Normalization	res4b27_branch2a	res4b27_branch2ax		
res4b27_branch2a_relu	ReLu	res4b27_branch2ax	res4b27_branch2axxx		
res4b27_branch2b	Convolution	res4b27_branch2axxx	res4b27_branch2b	3x3x256x256	256
bn4b27_branch2b	Batch Normalization	res4b27_branch2b	res4b27_branch2bx		
res4b27_branch2b_relu	ReLu	res4b27_branch2bx	res4b27_branch2bxxx		
res4b27_branch2c	Convolution	res4b27_branch2bxxx	res4b27_branch2c	1x1x256x1024	1024

Table 22. ResNet152 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
bn4b27_branch2c	Batch Normalization	res4b27_branch2c	res4b27_branch2cx		
res4b27	Summation	res4b26x, res4b27_branch2cx	res4b27		
res4b27_relu	ReLu	res4b27	res4b27x		
res4b28_branch2a	Convolution	res4b27x	res4b28_branch2a	1x1x1024x256	256
bn4b28_branch2a	Batch Normalization	res4b28_branch2a	res4b28_branch2ax		
res4b28_branch2a_relu	ReLu	res4b28_branch2ax	res4b28_branch2axxx		
res4b28_branch2b	Convolution	res4b28_branch2axxx	res4b28_branch2b	3x3x256x256	256
bn4b28_branch2b	Batch Normalization	res4b28_branch2b	res4b28_branch2bx		
res4b28_branch2b_relu	ReLu	res4b28_branch2bx	res4b28_branch2bxxx		
res4b28_branch2c	Convolution	res4b28_branch2bxxx	res4b28_branch2c	1x1x256x1024	1024
bn4b28_branch2c	Batch Normalization	res4b28_branch2c	res4b28_branch2cx		
res4b28	Summation	res4b27x, res4b28_branch2cx	res4b28		
res4b28_relu	ReLu	res4b28	res4b28x		
res4b29_branch2a	Convolution	res4b28x	res4b29_branch2a	1x1x1024x256	256
bn4b29_branch2a	Batch Normalization	res4b29_branch2a	res4b29_branch2ax		
res4b29_branch2a_relu	ReLu	res4b29_branch2ax	res4b29_branch2axxx		
res4b29_branch2b	Convolution	res4b29_branch2axxx	res4b29_branch2b	3x3x256x256	256
bn4b29_branch2b	Batch Normalization	res4b29_branch2b	res4b29_branch2bx		
res4b29_branch2b_relu	ReLu	res4b29_branch2bx	res4b29_branch2bxxx		
res4b29_branch2c	Convolution	res4b29_branch2bxxx	res4b29_branch2c	1x1x256x1024	1024
bn4b29_branch2c	Batch Normalization	res4b29_branch2c	res4b29_branch2cx		
res4b29	Summation	res4b28x, res4b29_branch2cx	res4b29		
res4b29_relu	ReLu	res4b29	res4b29x		
res4b30_branch2a	Convolution	res4b29x	res4b30_branch2a	1x1x1024x256	256
bn4b30_branch2a	Batch Normalization	res4b30_branch2a	res4b30_branch2ax		
res4b30_branch2a_relu	ReLu	res4b30_branch2ax	res4b30_branch2axxx		
res4b30_branch2b	Convolution	res4b30_branch2axxx	res4b30_branch2b	3x3x256x256	256
bn4b30_branch2b	Batch Normalization	res4b30_branch2b	res4b30_branch2bx		
res4b30_branch2b_relu	ReLu	res4b30_branch2bx	res4b30_branch2bxxx		
res4b30_branch2c	Convolution	res4b30_branch2bxxx	res4b30_branch2c	1x1x256x1024	1024
bn4b30_branch2c	Batch Normalization	res4b30_branch2c	res4b30_branch2cx		
res4b30	Summation	res4b29x, res4b30_branch2cx	res4b30		
res4b30_relu	ReLu	res4b30	res4b30x		
res4b31_branch2a	Convolution	res4b30x	res4b31_branch2a	1x1x1024x256	256
bn4b31_branch2a	Batch Normalization	res4b31_branch2a	res4b31_branch2ax		
res4b31_branch2a_relu	ReLu	res4b31_branch2ax	res4b31_branch2axxx		
res4b31_branch2b	Convolution	res4b31_branch2axxx	res4b31_branch2b	3x3x256x256	256
bn4b31_branch2b	Batch Normalization	res4b31_branch2b	res4b31_branch2bx		
res4b31_branch2b_relu	ReLu	res4b31_branch2bx	res4b31_branch2bxxx		
res4b31_branch2c	Convolution	res4b31_branch2bxxx	res4b31_branch2c	1x1x256x1024	1024
bn4b31_branch2c	Batch Normalization	res4b31_branch2c	res4b31_branch2cx		
res4b31	Summation	res4b30x, res4b31_branch2cx	res4b31		
res4b31_relu	ReLu	res4b31	res4b31x		
res4b32_branch2a	Convolution	res4b31x	res4b32_branch2a	1x1x1024x256	256

Table 22. ResNet152 Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
bn4b32_branch2a	Batch Normalization	res4b32_branch2a	res4b32_branch2ax		
res4b32_branch2a_relu	ReLu	res4b32_branch2ax	res4b32_branch2axxx		
res4b32_branch2b	Convolution	res4b32_branch2axxx	res4b32_branch2b	3x3x256x256	256
bn4b32_branch2b	Batch Normalization	res4b32_branch2b	res4b32_branch2bx		
res4b32_branch2b_relu	ReLu	res4b32_branch2bx	res4b32_branch2bxxx		
res4b32_branch2c	Convolution	res4b32_branch2bxxx	res4b32_branch2c	1x1x256x1024	1024
bn4b32_branch2c	Batch Normalization	res4b32_branch2c	res4b32_branch2cx		
res4b32	Summation	res4b31x, res4b32_branch2cx	res4b32		
res4b32_relu	ReLu	res4b32	res4b32x		
res4b33_branch2a	Convolution	res4b32x	res4b33_branch2a	1x1x1024x256	256
bn4b33_branch2a	Batch Normalization	res4b33_branch2a	res4b33_branch2ax		
res4b33_branch2a_relu	ReLu	res4b33_branch2ax	res4b33_branch2axxx		
res4b33_branch2b	Convolution	res4b33_branch2axxx	res4b33_branch2b	3x3x256x256	256
bn4b33_branch2b	Batch Normalization	res4b33_branch2b	res4b33_branch2bx		
res4b33_branch2b_relu	ReLu	res4b33_branch2bx	res4b33_branch2bxxx		
res4b33_branch2c	Convolution	res4b33_branch2bxxx	res4b33_branch2c	1x1x256x1024	1024
bn4b33_branch2c	Batch Normalization	res4b33_branch2c	res4b33_branch2cx		
res4b33	Summation	res4b32x, res4b33_branch2cx	res4b33		
res4b33_relu	ReLu	res4b33	res4b33x		
res4b34_branch2a	Convolution	res4b33x	res4b34_branch2a	1x1x1024x256	256
bn4b34_branch2a	Batch Normalization	res4b34_branch2a	res4b34_branch2ax		
res4b34_branch2a_relu	ReLu	res4b34_branch2ax	res4b34_branch2axxx		
res4b34_branch2b	Convolution	res4b34_branch2axxx	res4b34_branch2b	3x3x256x256	256
bn4b34_branch2b	Batch Normalization	res4b34_branch2b	res4b34_branch2bx		
res4b34_branch2b_relu	ReLu	res4b34_branch2bx	res4b34_branch2bxxx		
res4b34_branch2c	Convolution	res4b34_branch2bxxx	res4b34_branch2c	1x1x256x1024	1024
bn4b34_branch2c	Batch Normalization	res4b34_branch2c	res4b34_branch2cx		
res4b34	Summation	res4b33x, res4b34_branch2cx	res4b34		
res4b34_relu	ReLu	res4b34	res4b34x		
res4b35_branch2a	Convolution	res4b34x	res4b35_branch2a	1x1x1024x256	256
bn4b35_branch2a	Batch Normalization	res4b35_branch2a	res4b35_branch2ax		
res4b35_branch2a_relu	ReLu	res4b35_branch2ax	res4b35_branch2axxx		
res4b35_branch2b	Convolution	res4b35_branch2axxx	res4b35_branch2b	3x3x256x256	256
bn4b35_branch2b	Batch Normalization	res4b35_branch2b	res4b35_branch2bx		
res4b35_branch2b_relu	ReLu	res4b35_branch2bx	res4b35_branch2bxxx		
res4b35_branch2c	Convolution	res4b35_branch2bxxx	res4b35_branch2c	1x1x256x1024	1024
bn4b35_branch2c	Batch Normalization	res4b35_branch2c	res4b35_branch2cx		
res4b35	Summation	res4b34x, res4b35_branch2cx	res4b35		
res4b35_relu	ReLu	res4b35	res4b35x		
res5a_branch1	Convolution	res4b35x	res5a_branch1	1x1x1024x2048	2048
bn5a_branch1	Batch Normalization	res5a_branch1	res5a_branch1x		
res5a_branch2a	Convolution	res4b35x	res5a_branch2a	1x1x1024x512	512
bn5a_branch2a	Batch Normalization	res5a_branch2a	res5a_branch2ax		
res5a_branch2a_relu	ReLu	res5a_branch2ax	res5a_branch2axxx		

Table 22. ResNet152 Architecture (cont’d)

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
res5a_branch2b	Convolution	res5a_branch2axxx	res5a_branch2b	3x3x512x512	512
bn5a_branch2b	Batch Normalization	res5a_branch2b	res5a_branch2bx		
res5a_branch2b_relu	ReLu	res5a_branch2bx	res5a_branch2bxxx		
res5a_branch2c	Convolution	res5a_branch2bxxx	res5a_branch2c	1x1x512x2048	2048
bn5a_branch2c	Batch Normalization	res5a_branch2c	res5a_branch2cx		
res5a	Summation	res5a_branch1x, res5a_branch2cx	res5a		
res5a_relu	ReLu	res5a	res5ax		
res5b_branch2a	Convolution	res5ax	res5b_branch2a	1x1x2048x512	512
bn5b_branch2a	Batch Normalization	res5b_branch2a	res5b_branch2ax		
res5b_branch2a_relu	ReLu	res5b_branch2ax	res5b_branch2axxx		
res5b_branch2b	Convolution	res5b_branch2axxx	res5b_branch2b	3x3x512x512	512
bn5b_branch2b	Batch Normalization	res5b_branch2b	res5b_branch2bx		
res5b_branch2b_relu	ReLu	res5b_branch2bx	res5b_branch2bxxx		
res5b_branch2c	Convolution	res5b_branch2bxxx	res5b_branch2c	1x1x512x2048	2048
bn5b_branch2c	Batch Normalization	res5b_branch2c	res5b_branch2cx		
res5b	Summation	res5ax, res5b_branch2cx	res5b		
res5b_relu	ReLu	res5b	res5bx		
res5c_branch2a	Convolution	res5bx	res5c_branch2a	1x1x2048x512	512
bn5c_branch2a	Batch Normalization	res5c_branch2a	res5c_branch2ax		
res5c_branch2a_relu	ReLu	res5c_branch2ax	res5c_branch2axxx		
res5c_branch2b	Convolution	res5c_branch2axxx	res5c_branch2b	3x3x512x512	512
bn5c_branch2b	Batch Normalization	res5c_branch2b	res5c_branch2bx		
res5c_branch2b_relu	ReLu	res5c_branch2bx	res5c_branch2bxxx		
res5c_branch2c	Convolution	res5c_branch2bxxx	res5c_branch2c	1x1x512x2048	2048
bn5c_branch2c	Batch Normalization	res5c_branch2c	res5c_branch2cx		
res5c	Summation	res5bx, res5c_branch2cx	res5c		
res5c_relu	ReLu	res5c	res5cx		
pool5	Average Pooling	res5cx	pool5		
fc1000	Convolution	pool5	fc1000	1x1x2048x1000	1000
prob	SoftMax	fc1000	prob		

Table 23. Fully Convolutional Network Architecture

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
conv1_1	Convolution	data	conv1_1	3x3x3x64	64
relu1_1	ReLu	conv1_1	conv1_1x		
conv1_2	Convolution	conv1_1x	conv1_2	3x3x64x64	64
relu1_2	ReLu	conv1_2	conv1_2x		
pool1	Max Pooling	conv1_2x	pool1		
conv2_1	Convolution	pool1	conv2_1	3x3x64x128	128
relu2_1	ReLu	conv2_1	conv2_1x		
conv2_2	Convolution	conv2_1x	conv2_2	3x3x128x128	128
relu2_2	ReLu	conv2_2	conv2_2x		
pool2	Max Pooling	conv2_2x	pool2		
conv3_1	Convolution	pool2	conv3_1	3x3x128x256	256
relu3_1	ReLu	conv3_1	conv3_1x		
conv3_2	Convolution	conv3_1x	conv3_2	3x3x256x256	256
relu3_2	ReLu	conv3_2	conv3_2x		
conv3_3	Convolution	conv3_2x	conv3_3	3x3x256x256	256
relu3_3	ReLu	conv3_3	conv3_3x		
pool3	Max Pooling	conv3_3x	pool3		
conv4_1	Convolution	pool3	conv4_1	3x3x256x512	512
relu4_1	ReLu	conv4_1	conv4_1x		
conv4_2	Convolution	conv4_1x	conv4_2	3x3x512x512	512
relu4_2	ReLu	conv4_2	conv4_2x		
conv4_3	Convolution	conv4_2x	conv4_3	3x3x512x512	512
relu4_3	ReLu	conv4_3	conv4_3x		
pool4	Max Pooling	conv4_3x	pool4		
conv5_1	Convolution	pool4	conv5_1	3x3x512x512	512
relu5_1	ReLu	conv5_1	conv5_1x		
conv5_2	Convolution	conv5_1x	conv5_2	3x3x512x512	512
relu5_2	ReLu	conv5_2	conv5_2x		
conv5_3	Convolution	conv5_2x	conv5_3	3x3x512x512	512
relu5_3	ReLu	conv5_3	conv5_3x		
pool5	Max Pooling	conv5_3x	pool5		
fc6	Convolution	pool5	fc6	7x7x512x4096	4096
relu6	ReLu	fc6	fc6x		
fc7	Convolution	fc6x	fc7	1x1x4096x4096	4096
relu7	ReLu	fc7	fc7x		
score_fr	Convolution	fc7x	score	1x1x4096x21	21
score2	Convolution Transpose	score	score2	4x4x21x21	21
score_pool4	Convolution	pool4	score_pool4	1x1x512x21	21
crop	Crop	score_pool4, score2	score_pool4c		
fuse	Summation	score2, score_pool4c	score_fused		
score4	Convolution Transpose	score_fused	score4	4x4x21x21	21
score_pool3	Convolution	pool3	score_pool3	1x1x256x21	21
cropx	Crop	score_pool3, score4	score_pool3c		
fusex	Summation	score4, score_pool3c	score_final		
upsample	Convolution Transpose	score_final	bigscore	16x16x21x21	21
cropxx	Crop	bigscore, data	upscore		

Table 24. SegNet Architecture

Layer Name	**Block Type**	**Input**	**Output**	**Filter Weights**	**Bias Weights**
conv1_1	Convolution	data	conv1_1	3x3x3x64	64
bn_conv1_1	Batch Normalization	conv1_1	bn_conv1_1		
relu1_1	ReLu	bn_conv1_1	relu1_1		
conv1_2	Convolution	relu1_1	conv1_2	3x3x64x64	64
bn_conv1_2	Batch Normalization	conv1_2	bn_conv1_2		
relu1_2	ReLu	bn_conv1_2	relu1_2		
pool1	Max Pooling	relu1_2	pool1		
conv2_1	Convolution	pool1	conv2_1	3x3x64x128	128
bn_conv2_1	Batch Normalization	conv2_1	bn_conv2_1		
relu2_1	ReLu	bn_conv2_1	relu2_1		
conv2_2	Convolution	relu2_1	conv2_2	3x3x128x128	128
bn_conv2_2	Batch Normalization	conv2_2	bn_conv2_2		
relu2_2	ReLu	bn_conv2_2	relu2_2		
pool2	Max Pooling	relu2_2	pool2		
conv3_1	Convolution	pool2	conv3_1	3x3x128x256	256
bn_conv3_1	Batch Normalization	conv3_1	bn_conv3_1		
relu3_1	ReLu	bn_conv3_1	relu3_1		
conv3_2	Convolution	relu3_1	conv3_2	3x3x256x256	256
bn_conv3_2	Batch Normalization	conv3_2	bn_conv3_2		
relu3_2	ReLu	bn_conv3_2	relu3_2		
conv3_3	Convolution	relu3_2	conv3_3	3x3x256x256	256
bn_conv3_3	Batch Normalization	conv3_3	bn_conv3_3		
relu3_3	ReLu	bn_conv3_3	relu3_3		
pool3	Max Pooling	relu3_3	pool3		
conv4_1	Convolution	pool3	conv4_1	3x3x256x512	512
bn_conv4_1	Batch Normalization	conv4_1	bn_conv4_1		
relu4_1	ReLu	bn_conv4_1	relu4_1		
conv4_2	Convolution	relu4_1	conv4_2	3x3x512x512	512
bn_conv4_2	Batch Normalization	conv4_2	bn_conv4_2		
relu4_2	ReLu	bn_conv4_2	relu4_2		
conv4_3	Convolution	relu4_2	conv4_3	3x3x512x512	512
bn_conv4_3	Batch Normalization	conv4_3	bn_conv4_3		
relu4_3	ReLu	bn_conv4_3	relu4_3		
pool4	Max Pooling	relu4_3	pool4		
conv5_1	Convolution	pool4	conv5_1	3x3x512x512	512
bn_conv5_1	Batch Normalization	conv5_1	bn_conv5_1		
relu5_1	ReLu	bn_conv5_1	relu5_1		
conv5_2	Convolution	relu5_1	conv5_2	3x3x512x512	512
bn_conv5_2	Batch Normalization	conv5_2	bn_conv5_2		
relu5_2	ReLu	bn_conv5_2	relu5_2		
conv5_3	Convolution	relu5_2	conv5_3	3x3x512x512	512
bn_conv5_3	Batch Normalization	conv5_3	bn_conv5_3		
relu5_3	ReLu	bn_conv5_3	relu5_3		
pool5	Max Pooling	relu5_3	pool5		
decoder5_unpool	Max Unpooling	pool5	decoder5_unpool		
decoder5_conv3	Convolution	decoder5_unpool	decoder5_conv3	3x3x512x512	512
decoder5_bn_3	Batch Normalization	decoder5_conv3	decoder5_bn_3		
decoder5_relu_3	ReLu	decoder5_bn_3	decoder5_relu_3		

Table 24. SegNet Architecture (cont'd)

Layer Name	Block Type	Input	Output	Filter Weights	Bias Weights
decoder5_conv2	Convolution	decoder5_relu_3	decoder5_conv2	3x3x512x512	512
decoder5_bn_2	Batch Normalization	decoder5_conv2	decoder5_bn_2		
decoder5_relu_2	ReLu	decoder5_bn_2	decoder5_relu_2		
decoder5_conv1	Convolution	decoder5_relu_2	decoder5_conv1	3x3x512x512	512
decoder5_bn_1	Batch Normalization	decoder5_conv1	decoder5_bn_1		
decoder5_relu_1	ReLu	decoder5_bn_1	decoder5_relu_1		
decoder4_unpool	Max Unpooling	decoder5_relu_1, pool4	decoder4_unpool		
decoder4_conv3	Convolution	decoder4_unpool	decoder4_conv3	3x3x512x512	512
decoder4_bn_3	Batch Normalization	decoder4_conv3	decoder4_bn_3		
decoder4_relu_3	ReLu	decoder4_bn_3	decoder4_relu_3		
decoder4_conv2	Convolution	decoder4_relu_3	decoder4_conv2	3x3x512x512	512
decoder4_bn_2	Batch Normalization	decoder4_conv2	decoder4_bn_2		
decoder4_relu_2	ReLu	decoder4_bn_2	decoder4_relu_2		
decoder4_conv1	Convolution	decoder4_relu_2	decoder4_conv1	3x3x512x256	256
decoder4_bn_1	Batch Normalization	decoder4_conv1	decoder4_bn_1		
decoder4_relu_1	ReLu	decoder4_bn_1	decoder4_relu_1		
decoder3_unpool	Max Unpooling	decoder4_relu_1, pool3	decoder3_unpool		
decoder3_conv3	Convolution	decoder3_unpool	decoder3_conv3	3x3x256x256	256
decoder3_bn_3	Batch Normalization	decoder3_conv3	decoder3_bn_3		
decoder3_relu_3	ReLu	decoder3_bn_3	decoder3_relu_3		
decoder3_conv2	Convolution	decoder3_relu_3	decoder3_conv2	3x3x256x256	256
decoder3_bn_2	Batch Normalization	decoder3_conv2	decoder3_bn_2		
decoder3_relu_2	ReLu	decoder3_bn_2	decoder3_relu_2		
decoder3_conv1	Convolution	decoder3_relu_2	decoder3_conv1	3x3x256x128	128
decoder3_bn_1	Batch Normalization	decoder3_conv1	decoder3_bn_1		
decoder3_relu_1	ReLu	decoder3_bn_1	decoder3_relu_1		
decoder2_unpool	Max Unpooling	decoder3_relu_1, pool2	decoder2_unpool		
decoder2_conv2	Convolution	decoder2_unpool	decoder2_conv2	3x3x128x128	128
decoder2_bn_2	Batch Normalization	decoder2_conv2	decoder2_bn_2		
decoder2_relu_2	ReLu	decoder2_bn_2	decoder2_relu_2		
decoder2_conv1	Convolution	decoder2_relu_2	decoder2_conv1	3x3x128x64	64
decoder2_bn_1	Batch Normalization	decoder2_conv1	decoder2_bn_1		
decoder2_relu_1	ReLu	decoder2_bn_1	decoder2_relu_1		
decoder1_unpool	Max Unpooling	decoder2_relu_1, pool1	decoder1_unpool		
decoder1_conv2	Convolution	decoder1_unpool	decoder1_conv2	3x3x64x64	64
decoder1_bn_2	Batch Normalization	decoder1_conv2	decoder1_bn_2		
decoder1_relu_2	ReLu	decoder1_bn_2	decoder1_relu_2		
decoder1_conv1	Convolution	decoder1_relu_2	decoder1_conv1	3x3x64x2	2
decoder1_bn_1	Batch Normalization	decoder1_conv1	decoder1_bn_1		
decoder1_relu_1	ReLu	decoder1_bn_1	decoder1_relu_1		
softmax	Softmax	decoder1_relu_1	softmax		
pixelLabels	Cross Entropy Loss	softmax	pixelLabels		

APPENDIX II

CLASSIFICATION TEST RESULTS

Table 25. Test 1 Classification Accuracy Results

AlexNet - Test 1		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,8800	0,9467	0,9800	0,9867	0,9867	0,9867	0,9867	0,9867	0,9867	0,9867
	0.7K	0,9267	0,9933	0,9900	0,9900	0,9900	0,9933	0,9933	0,9933	0,9933	0,9933
	1.75K	0,9907	0,9907	0,9933	0,9947	0,9933	0,9933	0,9933	0,9933	0,9947	0,9947
	3.5K	0,9964	0,9972	0,9978	0,9981	0,9978	0,9979	0,9981	0,9981	0,9981	0,9983
	7K	0,9967	0,9974	0,9972	0,9966	0,9969	0,9971	0,9969	0,9971	0,9971	0,9971
	14K	0,9974	0,9974	0,9979	0,9981	0,9979	0,9979	0,9979	0,9979	0,9979	0,9979
	21K	0,9971	0,9976	0,9978	0,9978	0,9976	0,9976	0,9976	0,9978	0,9978	0,9978
	28K	0,9983	**0,9988**	**0,9988**	**0,9988**	**0,9988**	**0,9988**	**0,9986**	**0,9988**	**0,9988**	**0,9988**

VGG16 - Test1		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,9667	**1,0000**	0,9867	0,9800	0,9800	0,9867	0,9933	0,9933	0,9933	**1,0000**
	0.7K	0,9933	0,9700	0,9900	0,9900	0,9900	0,9900	0,9900	0,9900	0,9900	0,9900
	1.75K	0,9867	0,9893	0,9653	0,9920	0,9880	0,9880	0,9907	0,9907	0,9907	0,9907
	3.5K	0,9993	0,9995	0,9995	0,9995	0,9995	0,9995	0,9995	0,9995	0,9995	0,9995
	7K	0,9986	0,9983	0,9991	0,9993	0,9993	0,9993	0,9993	0,9993	0,9993	0,9993
	14K	0,9991	0,9993	0,9995	0,9993	0,9993	0,9995	0,9993	0,9993	0,9993	0,9993
	21K	0,9998	0,9998	0,9997	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998
	28K	0,9995	0,9986	0,9997	0,9990	0,9995	0,9997	0,9995	0,9995	0,9995	0,9997

VGG19 - Test1		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,8933	0,8267	0,9400	0,9133	**1,0000**	0,9733	0,9733	0,9733	0,9733	0,9733
	0.7K	**1,0000**	0,9800	0,9933	0,9867	0,9867	0,9867	0,9867	0,9867	0,9867	0,9867
	1.75K	0,9907	0,9920	0,9920	0,9920	0,9920	0,9920	0,9920	0,9920	0,9920	0,9920
	3.5K	0,9988	0,9988	0,9988	0,9991	0,9993	0,9993	0,9993	0,9993	0,9993	0,9993
	7K	0,9986	0,9991	0,9991	0,9993	0,9993	0,9993	0,9993	0,9993	0,9993	0,9993
	14K	0,9967	0,9988	0,9991	0,9988	0,9988	0,9990	0,9990	0,9990	0,9990	0,9990
	21K	0,9990	0,9995	0,9997	0,9995	0,9995	0,9997	0,9997	0,9995	0,9997	0,9997
	28K	0,9988	0,9991	0,9990	0,9990	0,9993	0,9993	0,9993	0,9993	0,9993	0,9997

Table 25. Test 1 Classification Results (cont'd)

GoogleNet - Test1		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,8667	0,9200	0,9333	0,9333	0,9333	0,9667	0,9667	0,9733	0,9733	0,9733
	0.7K	0,9367	0,9633	0,9733	0,9800	0,9833	0,9867	0,9900	0,9900	0,9900	0,9933
	1.75K	0,9560	0,9827	0,9893	0,9960	0,9973	0,9973	0,9973	0,9960	0,9960	0,9960
	3.5K	0,9954	0,9978	0,9978	0,9974	0,9976	0,9981	0,9981	0,9981	0,9981	0,9983
	7K	0,9972	0,9978	0,9979	0,9978	0,9978	0,9979	0,9978	0,9978	0,9978	0,9979
	14K	0,9979	0,9974	0,9981	0,9981	0,9988	0,9985	0,9985	0,9985	0,9986	0,9985
	21K	0,9976	0,9986	0,9986	0,9990	0,9990	0,9990	0,9990	0,9991	**0,9993**	**0,9993**
	28K	0,9979	0,9985	0,9985	0,9986	0,9985	0,9986	0,9986	0,9985	0,9986	0,9986

ResNet50 - Test1		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,7367	0,9500	0,9667	0,9833	0,9867	0,9900	0,9917	0,9933	0,9933	0,9933
	0.7K	0,8608	0,9700	0,9817	0,9858	0,9875	0,9883	0,9892	0,9900	0,9883	0,9900
	1.75K	0,9857	0,9957	0,9960	0,9967	0,9973	0,9973	0,9967	0,9973	0,9973	0,9977
	3.5K	0,9931	0,9955	0,9959	0,9957	0,9962	0,9960	0,9960	0,9969	0,9964	0,9960
	7K	0,9966	0,9978	0,9976	0,9981	0,9978	0,9981	0,9988	0,9985	0,9983	0,9983
	14K	0,9983	**0,9991**	**0,9991**	**0,9991**	**0,9991**	0,9990	**0,9991**	0,9990	0,9990	0,9990
	21K	0,9972	0,9983	0,9983	0,9981	0,9988	0,9986	0,9988	0,9990	**0,9991**	**0,9991**
	28K	0,9979	0,9988	0,9988	0,9990	0,9986	0,9988	0,9986	0,9988	0,9988	**0,9991**

ResNet101 - Test1		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,8367	0,9750	0,9917	0,9900	0,9883	0,9883	0,9883	0,9883	0,9883	0,9883
	0.7K	0,9350	0,9908	0,9933	0,9933	0,9933	0,9933	0,9933	0,9933	0,9933	0,9933
	1.75K	0,9950	0,9983	0,9987	0,9970	0,9977	0,9973	0,9970	0,9983	0,9983	0,9977
	3.5K	0,9954	0,9960	0,9966	0,9960	0,9969	0,9969	0,9974	0,9962	0,9969	0,9969
	7K	0,9936	0,9952	0,9966	0,9971	0,9969	0,9971	0,9967	0,9974	0,9974	0,9971
	14K	0,9971	0,9978	0,9983	0,9981	0,9978	0,9985	0,9986	0,9986	0,9986	0,9986
	21K	0,9988	0,9979	0,9986	0,9983	0,9985	0,9990	0,9988	0,9986	0,9986	0,9986
	28K	0,9967	0,9983	0,9979	0,9991	0,9990	**0,9993**	0,9991	0,9985	0,9978	0,9990

ResNet152 - Test1		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5733	0,9700	0,9750	0,9733	0,9750	0,9783	0,9783	0,9783	0,9817	0,9817
	0.7K	0,9442	0,9867	0,9875	0,9892	0,9875	0,9892	0,9892	0,9908	0,9908	0,9883
	1.75K	0,9917	0,9950	0,9947	0,9960	0,9977	0,9973	0,9977	**0,9990**	**0,9990**	**0,9990**
	3.5K	0,9720	0,9740	0,9733	0,9768	0,9699	0,9745	0,9766	0,9720	0,9744	0,9775
	7K	0,9929	0,9952	0,9957	0,9969	0,9971	0,9969	0,9972	0,9976	0,9974	0,9978
	14K	0,9940	0,9947	0,9929	0,9897	0,9907	0,9921	0,9921	0,9912	0,9905	0,9921
	21K	0,9921	0,9923	0,9914	0,9948	0,9943	0,9969	0,9954	0,9964	0,9959	0,9983
	28K	0,9768	0,9813	0,9782	0,9773	0,9825	0,9831	0,9806	0,9830	0,9854	0,9826

Table 26. Test2 Classification Accuracy Results

AlexNet - Test 2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,6021	0,5694	0,5654	0,5648	0,5654	0,5648	0,5681	0,5681	0,5681	0,5694
	0.7K	0,7847	0,5694	0,5452	0,5419	0,5510	0,5641	0,5681	0,5674	0,5674	0,5661
	1.75K	0,5766	0,5864	0,6126	0,6165	0,6106	0,6099	0,6106	0,6106	0,6113	0,6113
	3.5K	0,6963	0,6957	0,6872	0,6891	0,6819	0,6813	0,6793	0,6767	0,6760	0,6767
	7K	0,7637	0,7219	0,7029	0,6819	0,6931	0,6924	0,6904	0,6813	0,6819	0,6800
	14K	0,6859	0,6558	0,6571	0,6603	0,6453	0,6505	0,6518	0,6486	0,6479	0,6479
	21K	0,6800	0,6708	0,6944	0,6872	0,6787	0,6715	0,6715	0,6649	0,6721	0,6728
	28K	0,7736	**0,7984**	0,7893	0,7906	0,7834	0,7762	0,7742	0,7755	0,7644	0,7605

VGG16 - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,9634	0,9666	0,9143	0,8802	0,8724	0,8809	0,8855	0,8914	0,8959	0,8979
	0.7K	0,6970	0,9470	0,9025	0,8861	0,9018	0,9123	0,9149	0,9175	0,9175	0,9162
	1.75K	0,6603	0,7055	0,6414	0,9045	0,8554	0,8554	0,8586	0,8586	0,8573	0,8567
	3.5K	0,9332	0,9490	0,9463	0,9457	0,9470	0,9470	0,9470	0,9450	0,9463	0,9463
	7K	0,7186	0,7821	0,6774	0,6721	0,6728	0,6728	0,6728	0,6708	0,6708	0,6708
	14K	0,8213	0,8089	0,7788	0,8488	0,8298	0,8213	0,8318	0,8292	0,8312	0,8318
	21K	0,9516	**0,9673**	0,8881	0,8750	0,8704	0,8868	0,9005	0,8815	0,8946	0,8855
	28K	0,8750	0,8704	0,8370	0,8527	0,8586	0,8488	0,8580	0,8639	0,8632	0,8580

VGG19 - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5903	0,4503	0,5831	0,5046	0,9071	0,7101	0,7349	0,7277	0,7199	0,7160
	0.7K	0,7480	0,7304	0,7723	0,8259	0,8115	0,8069	0,8056	0,8010	0,8004	0,7978
	1.75K	0,8259	0,8645	0,8514	0,8541	0,8547	0,8541	0,8527	0,8514	0,8508	0,8501
	3.5K	**0,9823**	0,9548	0,9319	0,9516	0,9522	0,9522	0,9522	0,9522	0,9542	0,9542
	7K	0,9771	0,9149	0,9202	0,9130	0,9123	0,9123	0,9123	0,9103	0,9103	0,9103
	14K	0,7297	0,6414	0,5746	0,5982	0,5798	0,5602	0,5668	0,5654	0,5622	0,5654
	21K	0,9090	0,8613	0,8495	0,8174	0,8168	0,8305	0,8298	0,8272	0,8390	0,8344
	28K	0,5969	0,7029	0,7912	0,7657	0,7441	0,7467	0,7336	0,7389	0,7317	0,7297

GoogleNet - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,9136	0,9483	0,9627	0,9647	0,9725	0,9810	**0,9869**	0,9863	**0,9869**	**0,9869**
	0.7K	0,7232	0,7526	0,7899	0,8298	0,8599	0,8757	0,8861	0,8927	0,8959	0,8999
	1.75K	0,9365	0,9555	0,9496	0,9516	0,9483	0,9431	0,9424	0,9431	0,9476	0,9470
	3.5K	0,6315	0,5903	0,6165	0,6263	0,6355	0,6302	0,6224	0,6243	0,6276	0,6276
	7K	0,8063	0,7513	0,7140	0,6865	0,7094	0,7173	0,7225	0,7094	0,7173	0,7120
	14K	0,6623	0,6060	0,6152	0,6099	0,5792	0,5877	0,5864	0,5759	0,5733	0,5805
	21K	0,6073	0,5955	0,5903	0,5766	0,5700	0,5504	0,5537	0,5373	0,5563	0,5537
	28K	0,7343	0,6747	0,6675	0,6250	0,6374	0,6217	0,6027	0,6250	0,5923	0,5694

Table 26. Test 2 Classification Results (cont'd)

ResNet50 - Test2		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	**0,9025**	0,6793	0,4974	0,4810	0,4908	0,4961	0,4980	0,4980	0,4980	0,4980
	0.7K	0,6885	0,4921	0,5039	0,5033	0,5046	0,5007	0,5007	0,5020	0,5020	0,5007
	1.75K	0,5000	0,4993	0,4993	0,4987	0,4987	0,4987	0,4980	0,4987	0,4987	0,4987
	3.5K	0,5137	0,5118	0,4699	0,4234	0,4156	0,4077	0,4332	0,4025	0,3842	0,4077
	7K	0,5687	0,5628	0,5393	0,5229	0,5386	0,5975	0,5445	0,5412	0,5412	0,5517
	14K	0,4980	0,4967	0,4935	0,4941	0,4941	0,4882	0,5020	0,4849	0,4836	0,4882
	21K	0,4836	0,4679	0,4483	0,4725	0,4254	0,4280	0,4679	0,4031	0,4352	0,4143
	28K	0,4601	0,4313	0,4457	0,4522	0,3619	0,3220	0,3874	0,3920	0,4228	0,4071

ResNet101 - Test2		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,5065	0,5177	0,5020	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000
	0.7K	0,5105	0,5000	0,5007	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000
	1.75K	0,7363	0,7232	0,7349	0,7395	0,7467	0,7330	0,7552	0,7624	0,7637	0,7441
	3.5K	0,5223	0,5347	0,5524	0,5798	0,5753	0,5720	0,6027	0,6374	0,6342	0,5753
	7K	0,7160	0,7565	0,7277	0,7402	0,7565	0,7349	0,7291	0,7258	**0,7709**	0,7382
	14K	0,6178	0,6139	0,6152	0,6047	0,5838	0,5609	0,5550	0,5687	0,5497	0,5563
	21K	0,6584	0,6643	0,6446	0,6918	0,7173	0,6760	0,6616	0,6433	0,6204	0,6302
	28K	0,6008	0,7160	0,6839	0,6996	0,7212	0,7363	0,7343	0,7251	0,7402	0,7258

ResNet152 - Test2		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,4791	0,5020	0,4961	0,4954	0,4974	0,4967	0,4974	0,4961	0,4967	0,4961
	0.7K	0,5000	0,5000	0,5020	0,5039	0,5033	0,5020	0,5033	0,5020	0,5020	0,5013
	1.75K	0,5039	0,5033	0,5046	0,5033	0,5033	0,5033	0,5007	0,5046	0,5026	0,5026
	3.5K	0,5000	0,5000	0,5000	0,5000	0,5000	0,4987	0,4993	0,4993	0,5000	0,5000
	7K	**0,6171**	0,5694	0,5864	0,5641	0,5465	0,5589	0,5425	0,5419	0,5497	0,5491
	14K	0,4725	0,4732	0,4771	0,4758	0,4791	0,4745	0,4797	0,4692	0,4712	0,4699
	21K	0,5177	0,5065	0,5046	0,5295	0,5079	0,5380	0,5301	0,5321	0,5223	0,5596
	28K	0,5079	0,5124	0,5065	0,5020	0,5065	0,5033	0,5065	0,5098	0,5124	0,5000

Table 27. Test 3 Classification Accuracy Results

AlexNet - Test 3		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,6124	0,5964	0,6064	0,6185	0,6145	0,6185	0,6205	0,6225	0,6325	0,6345
	0.7K	**0,8574**	0,6365	0,5863	0,5863	0,5984	0,6225	0,6325	0,6305	0,6265	0,6265
	1.75K	0,6827	0,6807	0,7369	0,7430	0,7390	0,7390	0,7390	0,7390	0,7369	0,7369
	3.5K	0,6767	0,6747	0,6546	0,6506	0,6446	0,6426	0,6365	0,6365	0,6345	0,6345
	7K	0,8193	0,7711	0,7088	0,6827	0,6888	0,6948	0,6968	0,6867	0,6888	0,6888
	14K	0,7751	0,7309	0,7269	0,7390	0,7088	0,7149	0,7229	0,7048	0,7088	0,7068
	21K	0,6627	0,6345	0,6707	0,6667	0,6466	0,6345	0,6446	0,6325	0,6486	0,6526
	28K	0,7410	0,7530	0,7450	0,7470	0,7530	0,7490	0,7490	0,7490	0,7470	0,7450

VGG16 - Test3		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,8373	0,8695	0,7731	0,7590	0,7550	0,7590	0,7631	0,7651	0,7731	0,7731
	0.7K	0,8153	0,7992	0,8153	0,8173	0,8133	0,8173	0,8133	0,8133	0,8133	0,8133
	1.75K	0,8655	0,8594	0,6948	**0,9779**	0,9699	0,9719	0,9719	0,9719	0,9719	0,9719
	3.5K	0,8614	0,8474	0,8474	0,8474	0,8494	0,8494	0,8494	0,8474	0,8494	0,8494
	7K	0,7088	0,7691	0,7008	0,6928	0,6928	0,6908	0,6888	0,6847	0,6847	0,6847
	14K	0,7309	0,7229	0,7390	0,7771	0,7691	0,7570	0,7731	0,7731	0,7771	0,7811
	21K	0,9518	0,9458	0,8092	0,8313	0,8293	0,8434	0,8755	0,8494	0,8614	0,8534
	28K	0,8434	0,8614	0,8273	0,8594	0,8715	0,8755	0,8815	0,8896	0,8896	0,8896

VGG19 - Test3		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,6265	0,6024	0,6024	0,5201	0,8976	0,8213	0,8434	0,8353	0,8313	0,8293
	0.7K	0,6928	0,6566	0,7008	0,7470	0,7369	0,7309	0,7289	0,7229	0,7229	0,7229
	1.75K	0,8554	0,9116	0,9056	0,9076	0,9116	0,9116	0,9116	0,9116	0,9116	0,9116
	3.5K	**0,9639**	0,8896	0,8715	0,8855	0,8876	0,8876	0,8876	0,8876	0,8876	0,8896
	7K	0,7932	0,7751	0,7851	0,7811	0,7811	0,7811	0,7831	0,7831	0,7871	0,7871
	14K	0,7369	0,6486	0,6165	0,6305	0,6225	0,6124	0,6225	0,6185	0,6124	0,6185
	21K	0,9498	0,9317	0,9157	0,9036	0,9036	0,9197	0,9197	0,9157	0,9197	0,9197
	28K	0,7108	0,8333	0,8956	0,8855	0,8815	0,8835	0,8815	0,8835	0,8795	0,8815

GoogleNet - Test3		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	0.35K	0,6667	0,6908	0,7008	0,7149	0,7289	0,7329	0,7390	0,7309	0,7229	0,7149
	0.7K	0,8213	0,8193	0,8233	0,8112	0,8133	0,8112	0,8112	0,8133	0,8153	0,8112
	1.75K	0,8936	0,8996	0,8896	0,9056	0,9036	0,9076	0,9116	0,9137	**0,9217**	**0,9217**
	3.5K	0,6707	0,6667	0,6867	0,6968	0,7028	0,7048	0,7048	0,7068	0,7068	0,7108
	7K	0,7892	0,7731	0,7430	0,7390	0,7450	0,7470	0,7470	0,7470	0,7470	0,7470
	14K	0,7309	0,6827	0,6968	0,6928	0,6606	0,6747	0,6747	0,6667	0,6667	0,6727
	21K	0,6486	0,6325	0,6285	0,6285	0,6185	0,5964	0,6024	0,5843	0,6004	0,5984
	28K	0,8112	0,7871	0,7811	0,7590	0,7671	0,7590	0,7530	0,7691	0,7510	0,7329

Table 27. Test 3 Classification Accuracy Results (cont’d)

ResNet50 - Test3		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	**0.35K**	**0,6446**	0,5683	0,5020	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000
	0.7K	0,5382	0,5803	0,5803	0,5904	0,5984	0,5924	0,6024	0,6004	0,6004	0,5884
	1.75K	0,5060	0,5060	0,5100	0,5060	0,5040	0,5020	0,5000	0,5040	0,5000	0,5080
	3.5K	0,4859	0,4859	0,4900	0,4779	0,4759	0,4659	0,4699	0,4719	0,4920	0,4779
	7K	0,4719	0,4679	0,4578	0,4699	0,4719	0,4739	0,4679	0,4618	0,4739	0,4719
	14K	0,5000	0,5060	0,5100	0,5020	0,5100	0,5100	0,5181	0,5141	0,5060	0,5100
	21K	0,4900	0,4880	0,4880	0,4920	0,4679	0,4639	0,4779	0,4538	0,4558	0,4699
	28K	0,4960	0,4940	0,4980	0,5000	0,4940	0,4859	0,4940	0,4940	0,4980	0,4940

ResNet101 - Test3		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	**0.35K**	0,6968	0,6446	0,5221	0,5020	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000
	0.7K	**0,7430**	0,5161	0,5321	0,5402	0,5422	0,5321	0,5382	0,5502	0,5502	0,5502
	1.75K	0,5341	0,5321	0,5301	0,5261	0,5261	0,5321	0,5261	0,5341	0,5261	0,5341
	3.5K	0,5884	0,5863	0,5863	0,6225	0,5984	0,5904	0,6325	0,6466	0,6486	0,6285
	7K	0,6325	0,6225	0,6165	0,6305	0,6265	0,6285	0,6104	0,6325	0,6225	0,6265
	14K	0,5502	0,5301	0,5221	0,5241	0,5221	0,5221	0,5201	0,4960	0,5141	0,5241
	21K	0,5984	0,5924	0,5884	0,6104	0,5884	0,5743	0,5904	0,5984	0,5743	0,5843
	28K	0,5402	0,5803	0,5683	0,5904	0,5843	0,5823	0,6104	0,6225	0,6064	0,5944

ResNet152 - Test3		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	**0.35K**	0,5201	0,5120	0,5080	0,4940	0,4940	0,4980	0,4960	0,5000	0,4960	0,4960
	0.7K	0,5020	0,5020	0,5100	0,5281	0,5321	0,5321	0,5321	0,5321	0,5241	0,5321
	1.75K	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000
	3.5K	0,5020	0,5040	0,5020	0,5020	0,5040	0,5000	0,5060	0,5040	0,5040	0,5060
	7K	0,5141	0,5141	0,5060	0,5060	0,5040	0,5100	0,5040	0,5040	0,5060	0,5020
	14K	0,5201	0,5141	0,5080	0,5040	0,5161	**0,5422**	0,5040	0,5100	0,5060	0,5141
	21K	0,5181	0,5141	0,5020	0,5201	0,5060	0,5281	0,5261	0,5241	0,5221	0,5402
	28K	0,4940	0,4880	0,5020	0,5020	0,4980	0,5060	0,5020	0,4900	0,5040	0,5100

Table 28. Test 4 Classification Accuracy Results

AlexNet - Test 4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5741	0,5684	0,5722	0,5760	0,5798	0,5875	0,5913	0,5970	0,6008	0,5989
	0.7K	0,7624	0,5779	0,5475	0,5456	0,5513	0,5589	0,5608	0,5589	0,5589	0,5589
	1.75K	0,7757	0,7947	0,8365	0,8460	0,8327	0,8289	0,8232	0,8232	0,8251	0,8232
	3.5K	0,7015	0,6977	0,6787	0,6768	0,6673	0,6635	0,6616	0,6616	0,6578	0,6578
	7K	**0,8669**	0,8232	0,7871	0,7738	0,7776	0,7776	0,7776	0,7757	0,7738	0,7719
	14K	0,7567	0,6920	0,6920	0,7015	0,6654	0,6711	0,6768	0,6692	0,6654	0,6692
	21K	0,8346	0,8232	0,8479	0,8460	0,8384	0,8384	0,8384	0,8365	0,8403	0,8422
	28K	0,7757	0,7890	0,7757	0,7700	0,7738	0,7700	0,7719	0,7776	0,7757	0,7738

VGG19 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,6711	0,5342	0,5608	0,5019	0,8707	0,8308	0,8441	0,8403	0,8384	0,8384
	0.7K	0,6768	0,7186	0,7433	0,7586	0,7567	0,7605	0,7662	0,7605	0,7605	0,7643
	1.75K	0,7738	0,8213	0,8194	0,8175	0,8175	0,8194	0,8194	0,8194	0,8194	0,8194
	3.5K	0,8460	0,8764	0,8745	0,8821	**0,8840**	0,8821	0,8821	0,8821	0,8783	0,8783
	7K	0,7243	0,7985	0,8023	0,8042	0,8042	0,8061	0,8080	0,8099	0,8118	0,8156
	14K	0,7833	0,7795	0,7338	0,7833	0,7529	0,7281	0,7395	0,7376	0,7376	0,7414
	21K	0,7700	0,8251	0,8156	0,8232	0,8289	0,8232	0,8232	0,8213	0,8232	0,8213
	28K	0,8384	0,8308	0,8346	0,8441	0,8384	0,8422	0,8441	0,8422	0,8460	0,8460

VGG16 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,8042	0,8479	0,8821	0,8726	0,8745	0,8745	0,8764	0,8745	0,8783	0,8764
	0.7K	0,6768	0,8346	0,8137	0,8004	0,8004	0,8118	0,8156	0,8156	0,8156	0,8156
	1.75K	0,7719	0,8688	0,7643	**0,9563**	0,9392	0,9373	0,9392	0,9392	0,9411	0,9373
	3.5K	0,8232	0,8118	0,8137	0,8080	0,8061	0,8099	0,8099	0,8061	0,8118	0,8080
	7K	0,8175	0,8384	0,8365	0,8308	0,8289	0,8289	0,8289	0,8289	0,8270	0,8270
	14K	0,8080	0,7909	0,8232	0,8669	0,8460	0,8403	0,8593	0,8593	0,8612	0,8631
	21K	0,8612	0,8593	0,8726	0,8726	0,8707	0,8745	0,8669	0,8688	0,8650	0,8650
	28K	0,8289	0,7947	0,8004	0,8099	0,8175	0,8175	0,8137	0,8099	0,8099	0,8080

GoogleNet - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5323	0,5627	0,5837	0,6331	0,6711	0,6882	0,6996	0,7186	0,7338	0,7414
	0.7K	0,8175	0,8346	0,8707	0,8859	0,8973	0,8992	0,9011	0,9011	0,9049	0,9049
	1.75K	0,7529	0,8327	0,8536	0,8726	0,8821	0,8897	0,9030	0,9049	**0,9068**	**0,9068**
	3.5K	0,8517	0,8593	0,8935	0,9049	0,8992	0,9030	0,9030	0,9030	0,9030	0,8992
	7K	0,8745	0,8707	0,8593	0,8555	0,8631	0,8688	0,8783	0,8669	0,8802	0,8840
	14K	0,7662	0,7015	0,7072	0,7034	0,6654	0,6825	0,6806	0,6616	0,6578	0,6673
	21K	0,7053	0,6901	0,6863	0,6787	0,6730	0,6445	0,6578	0,6426	0,6635	0,6654
	28K	0,8973	0,8631	0,8745	0,8498	0,8783	0,8726	0,8574	0,8783	0,8669	0,8403

Table 28. Test 4 Classification Accuracy Results (cont’d)

ResNet50 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	**0,6901**	0,6806	0,5494	0,5057	0,5076	0,5000	0,5000	0,5019	0,5019	0,5038
	0.7K	0,6274	0,5989	0,5551	0,5418	0,5475	0,5323	0,5342	0,5418	0,5475	0,5399
	1.75K	0,5266	0,5171	0,5209	0,5684	0,5285	0,5456	0,5171	0,5342	0,5171	0,5513
	3.5K	0,5209	0,5342	0,5057	0,4981	0,4905	0,4867	0,4943	0,5038	0,4981	0,5095
	7K	0,4563	0,4449	0,4297	0,4411	0,4392	0,4544	0,4468	0,4240	0,4392	0,4354
	14K	0,5266	0,5380	0,5418	0,5304	0,5437	0,5323	0,5741	0,5494	0,5494	0,5532
	21K	0,5399	0,5304	0,5190	0,5304	0,5380	0,5570	0,5418	0,5894	0,5266	0,5684
	28K	0,5038	0,4981	0,5057	0,5038	0,4962	0,4981	0,5019	0,5038	0,5038	0,4962

ResNet101 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5494	0,5913	0,5627	0,5380	0,5095	0,5133	0,5152	0,5190	0,5133	0,5133
	0.7K	0,5570	0,5418	0,5722	0,5837	0,5741	0,5589	0,5646	0,5703	0,5760	0,6008
	1.75K	0,6388	0,6407	0,6559	0,6616	0,6464	0,6806	0,6445	0,6787	0,6407	0,6692
	3.5K	0,6141	0,6103	0,6331	0,6578	0,6350	0,6464	0,6540	0,6521	0,6502	0,6445
	7K	0,6179	0,6521	0,6445	0,6350	0,6464	0,6654	0,6654	0,6844	0,6711	0,6711
	14K	0,5779	0,5989	0,5703	0,5722	0,5760	0,5722	0,5665	0,5703	0,5513	0,5703
	21K	0,6502	0,6692	0,6464	0,6901	0,6730	0,6635	0,6749	0,7091	0,6977	**0,7243**
	28K	0,5722	0,5684	0,5722	0,5837	0,5894	0,6065	0,6027	0,6046	0,5875	0,6046

ResNet152 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,4924	0,5437	0,5285	0,5114	0,5171	0,5095	0,5076	0,5000	0,5114	0,5057
	0.7K	0,5190	0,5076	0,5114	0,5342	0,5285	0,5266	0,5285	0,5228	0,5323	0,5342
	1.75K	0,5076	0,5095	0,5114	0,5076	0,5095	0,5057	0,5114	0,5114	0,5114	0,5114
	3.5K	0,5114	0,5076	0,5076	0,5095	0,5095	0,5095	0,5114	0,5114	0,5095	0,5114
	7K	0,5266	0,5209	0,5247	0,5247	0,5209	0,5247	0,5228	0,5228	0,5171	0,5247
	14K	0,5760	0,5741	0,5475	0,5209	0,5513	0,5817	0,5513	0,5380	0,5266	0,5532
	21K	0,5570	0,5418	0,5228	0,5532	0,5399	0,5665	0,5589	0,5665	0,5532	**0,6198**
	28K	0,5190	0,5399	0,5342	0,5190	0,5285	0,5171	0,5228	0,5418	0,5627	0,5551

Table 29. Test 1 Classification confidence weighted Accuracy Results

AlexNet - Test 1		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	**0.35K**	0,8965	0,9737	0,9886	0,9914	0,9924	0,9936	0,9935	0,9938	0,9942	0,9943
	0.7K	0,9413	0,9936	0,9890	0,9912	0,9948	0,9964	0,9969	0,9969	0,9969	0,9969
	1.75K	0,9948	0,9950	0,9964	0,9966	0,9966	0,9964	0,9964	0,9964	0,9966	0,9966
	3.5K	0,9973	0,9979	0,9982	0,9984	0,9982	0,9983	0,9984	0,9984	0,9984	0,9985
	7K	0,9974	0,9979	0,9979	0,9974	0,9977	0,9978	0,9978	0,9978	0,9978	0,9978
	14K	0,9979	0,9979	0,9982	0,9984	0,9982	0,9983	0,9983	0,9983	0,9983	0,9983
	21K	0,9976	0,9979	0,9980	0,9981	0,9980	0,9980	0,9980	0,9981	0,9982	0,9982
	28K	0,9985	0,9989	0,9990	0,9990	0,9990	0,9991	0,9990	**0,9991**	**0,9991**	**0,9991**

VGG16 - Test1		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	**0.35K**	0,9957	0,9957	0,9981	0,9985	0,9981	0,9979	0,9978	0,9979	0,9979	0,9979
	0.7K	0,9964	0,9979	0,9986	0,9986	0,9986	0,9986	0,9986	0,9985	0,9985	0,9985
	1.75K	0,9981	0,9983	0,9931	0,9981	0,9986	0,9986	0,9986	0,9986	0,9986	0,9986
	3.5K	0,9994	0,9995	0,9995	0,9995	0,9995	0,9995	0,9995	0,9995	0,9995	0,9995
	7K	0,9989	0,9985	0,9993	0,9994	0,9994	0,9994	0,9994	0,9994	0,9994	0,9994
	14K	0,9992	0,9994	0,9995	0,9994	0,9994	0,9995	0,9994	0,9994	0,9994	0,9994
	21K	0,9998	**0,9998**	0,9997	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998
	28K	0,9996	0,9990	0,9997	0,9993	0,9996	0,9997	0,9996	0,9996	0,9996	0,9997

VGG19 - Test1		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	**0.35K**	0,9905	0,9759	0,9847	0,9733	0,9943	0,9967	0,9967	0,9967	0,9967	0,9967
	0.7K	0,9945	0,9948	0,9979	0,9969	0,9969	0,9966	0,9971	0,9974	0,9974	0,9974
	1.75K	0,9969	0,9985	0,9986	0,9988	0,9988	0,9988	0,9988	0,9988	0,9988	0,9988
	3.5K	0,9990	0,9990	0,9990	0,9993	0,9994	0,9994	0,9994	0,9994	0,9994	0,9994
	7K	0,9989	0,9992	0,9992	0,9994	0,9994	0,9994	0,9994	0,9994	0,9994	0,9994
	14K	0,9972	0,9989	0,9992	0,9989	0,9990	0,9991	0,9990	0,9991	0,9991	0,9990
	21K	0,9991	0,9996	0,9997	0,9996	0,9996	0,9997	0,9997	0,9996	0,9997	0,9997
	28K	0,9991	0,9994	0,9993	0,9993	0,9995	0,9995	0,9995	0,9995	0,9995	**0,9997**

GoogleNet - Test1		Number of Epoch									
		1	2	3	4	5	6	7	8	9	10
Training Dataset Size	**0.35K**	0,8968	0,9229	0,9374	0,9512	0,9572	0,9637	0,9692	0,9749	0,9776	0,9799
	0.7K	0,9155	0,9443	0,9656	0,9768	0,9814	0,9856	0,9880	0,9893	0,9912	0,9923
	1.75K	0,9596	0,9830	0,9892	0,9928	0,9950	0,9954	0,9955	0,9960	0,9966	0,9962
	3.5K	0,9964	0,9982	0,9982	0,9981	0,9982	0,9985	0,9985	0,9985	0,9986	0,9987
	7K	0,9975	0,9980	0,9981	0,9979	0,9979	0,9980	0,9980	0,9979	0,9979	0,9980
	14K	0,9981	0,9979	0,9984	0,9984	0,9989	0,9987	0,9987	0,9987	0,9988	0,9987
	21K	0,9982	0,9989	0,9990	0,9992	0,9992	0,9991	0,9992	0,9993	**0,9994**	0,9994
	28K	0,9982	0,9987	0,9987	0,9988	0,9987	0,9988	0,9988	0,9987	0,9988	0,9989

Table 29. Test 1 Classification confidence weighted Accuracy Results (cont’d)

ResNet50 - Test1		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,8099	0,9451	0,9723	0,9799	0,9813	0,9821	0,9830	0,9831	0,9833	0,9838
	0.7K	0,8535	0,9675	0,9868	0,9892	0,9911	0,9912	0,9917	0,9919	0,9921	0,9923
	1.75K	0,9673	0,9911	0,9909	0,9938	0,9945	0,9954	0,9954	0,9955	0,9955	0,9962
	3.5K	0,9947	0,9965	0,9968	0,9968	0,9970	0,9971	0,9971	0,9977	0,9973	0,9969
	7K	0,9975	0,9984	0,9983	0,9987	0,9983	0,9986	0,9991	0,9989	0,9986	0,9987
	14K	0,9988	**0,9994**	0,9993	0,9993	0,9994	0,9991	0,9993	0,9991	0,9991	0,9991
	21K	0,9979	0,9986	0,9987	0,9985	0,9990	0,9990	0,9991	0,9991	0,9992	0,9992
	28K	0,9985	0,9990	0,9991	0,9992	0,9990	0,9991	0,9990	0,9991	0,9991	0,9994

ResNet101 - Test1		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,8584	0,9283	0,9584	0,9723	0,9754	0,9756	0,9766	0,9776	0,9778	0,9780
	0.7K	0,8619	0,9579	0,9818	0,9861	0,9866	0,9880	0,9881	0,9900	0,9886	0,9893
	1.75K	0,9811	0,9916	0,9926	0,9948	0,9936	0,9952	0,9950	0,9957	0,9952	0,9952
	3.5K	0,9966	0,9970	0,9974	0,9972	0,9976	0,9977	0,9979	0,9973	0,9977	0,9975
	7K	0,9953	0,9966	0,9974	0,9977	0,9976	0,9977	0,9974	0,9979	0,9979	0,9977
	14K	0,9978	0,9983	0,9987	0,9986	0,9984	0,9988	0,9988	0,9988	0,9988	0,9989
	21K	0,9991	0,9985	0,9990	0,9987	0,9987	0,9991	0,9989	0,9989	0,9990	0,9989
	28K	0,9977	0,9986	0,9984	0,9992	0,9991	**0,9994**	0,9993	0,9988	0,9982	0,9992

ResNet152 - Test1		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,6063	0,9257	0,9714	0,9778	0,9799	0,9814	0,9816	0,9802	0,9809	0,9819
	0.7K	0,8464	0,9630	0,9819	0,9864	0,9866	0,9862	0,9866	0,9864	0,9868	0,9857
	1.75K	0,9739	0,9861	0,9874	0,9923	0,9909	0,9926	0,9926	0,9933	0,9931	0,9931
	3.5K	0,9760	0,9768	0,9765	0,9797	0,9723	0,9778	0,9793	0,9752	0,9769	0,9799
	7K	0,9949	0,9966	0,9970	0,9976	0,9977	0,9977	0,9979	0,9981	0,9980	0,9982
	14K	0,9952	0,9957	0,9939	0,9909	0,9921	0,9933	0,9929	0,9923	0,9916	0,9931
	21K	0,9937	0,9936	0,9924	0,9959	0,9951	0,9975	0,9961	0,9971	0,9967	**0,9986**
	28K	0,9783	0,9827	0,9794	0,9785	0,9837	0,9844	0,9817	0,9842	0,9863	0,9838

Table 30. Test 2 Classification confidence weighted Accuracy Results

AlexNet - Test 2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,6107	0,5800	0,5746	0,5726	0,5719	0,5714	0,5729	0,5731	0,5734	0,5744
	0.7K	0,8060	0,5760	0,5474	0,5458	0,5562	0,5682	0,5730	0,5724	0,5719	0,5708
	1.75K	0,5842	0,5917	0,6210	0,6251	0,6185	0,6177	0,6179	0,6182	0,6186	0,6188
	3.5K	0,7142	0,7144	0,7041	0,7050	0,6977	0,6973	0,6944	0,6922	0,6913	0,6916
	7K	0,7825	0,7428	0,7204	0,6980	0,7084	0,7086	0,7067	0,6976	0,6985	0,6964
	14K	0,7017	0,6663	0,6686	0,6749	0,6548	0,6610	0,6626	0,6582	0,6577	0,6579
	21K	0,6883	0,6802	0,7059	0,6995	0,6894	0,6811	0,6818	0,6739	0,6832	0,6843
	28K	0,7951	**0,8201**	0,8078	0,8094	0,8024	0,7962	0,7950	0,7958	0,7839	0,7796

VGG16 - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,9697	0,9724	0,9233	0,8896	0,8825	0,8901	0,8956	0,9008	0,9048	0,9067
	0.7K	0,7160	0,9583	0,9169	0,9016	0,9169	0,9269	0,9296	0,9320	0,9319	0,9313
	1.75K	0,6643	0,7128	0,6453	0,9136	0,8631	0,8628	0,8660	0,8660	0,8652	0,8641
	3.5K	0,9446	0,9566	0,9549	0,9541	0,9550	0,9551	0,9551	0,9536	0,9547	0,9547
	7K	0,7305	0,7941	0,6883	0,6812	0,6827	0,6826	0,6821	0,6803	0,6804	0,6804
	14K	0,8370	0,8245	0,7955	0,8620	0,8440	0,8337	0,8455	0,8429	0,8442	0,8456
	21K	0,9600	**0,9734**	0,9009	0,8881	0,8827	0,8986	0,9116	0,8931	0,9062	0,8970
	28K	0,8895	0,8824	0,8509	0,8653	0,8695	0,8601	0,8688	0,8753	0,8738	0,8681

VGG19 - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5928	0,4461	0,5957	0,5065	0,9209	0,7264	0,7515	0,7438	0,7363	0,7318
	0.7K	0,7602	0,7389	0,7838	0,8372	0,8249	0,8197	0,8175	0,8133	0,8118	0,8094
	1.75K	0,8476	0,8808	0,8688	0,8713	0,8720	0,8716	0,8702	0,8694	0,8688	0,8684
	3.5K	**0,9859**	0,9619	0,9422	0,9589	0,9594	0,9597	0,9598	0,9598	0,9612	0,9613
	7K	0,9823	0,9256	0,9302	0,9244	0,9239	0,9240	0,9237	0,9223	0,9226	0,9225
	14K	0,7424	0,6481	0,5781	0,6030	0,5830	0,5640	0,5700	0,5680	0,5644	0,5673
	21K	0,9246	0,8797	0,8699	0,8361	0,8348	0,8481	0,8471	0,8449	0,8553	0,8515
	28K	0,6140	0,7211	0,8107	0,7867	0,7636	0,7661	0,7536	0,7584	0,7512	0,7477

GoogleNet - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,9313	0,9602	0,9719	0,9749	0,9809	0,9866	0,9904	0,9903	**0,9907**	0,9906
	0.7K	0,7421	0,7764	0,8167	0,8551	0,8820	0,8968	0,9063	0,9123	0,9158	0,9189
	1.75K	0,9500	0,9658	0,9611	0,9622	0,9599	0,9566	0,9563	0,9566	0,9598	0,9591
	3.5K	0,6556	0,6079	0,6387	0,6481	0,6585	0,6485	0,6404	0,6421	0,6438	0,6441
	7K	0,8265	0,7689	0,7313	0,7044	0,7257	0,7321	0,7376	0,7239	0,7327	0,7275
	14K	0,6745	0,6145	0,6247	0,6193	0,5862	0,5968	0,5951	0,5827	0,5800	0,5877
	21K	0,6316	0,6177	0,6085	0,5923	0,5838	0,5618	0,5661	0,5474	0,5673	0,5650
	28K	0,7593	0,7006	0,6932	0,6419	0,6561	0,6387	0,6176	0,6424	0,6067	0,5830

ResNet50 - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	**0,9219**	0,6937	0,4953	0,4803	0,4898	0,4954	0,4972	0,4974	0,4976	0,4981
	0.7K	0,7047	0,4941	0,5075	0,5063	0,5076	0,5030	0,5028	0,5047	0,5043	0,5037
	1.75K	0,4988	0,4988	0,4990	0,4980	0,4984	0,4982	0,4976	0,4977	0,4980	0,4975
	3.5K	0,5131	0,5137	0,4695	0,4197	0,4141	0,4059	0,4289	0,4011	0,3814	0,4061
	7K	0,5720	0,5657	0,5425	0,5252	0,5419	0,6007	0,5443	0,5458	0,5444	0,5547
	14K	0,4977	0,4955	0,4928	0,4914	0,4921	0,4870	0,4991	0,4843	0,4830	0,4868
	21K	0,4855	0,4681	0,4447	0,4691	0,4209	0,4221	0,4643	0,4011	0,4282	0,4075
	28K	0,4594	0,4301	0,4454	0,4533	0,3586	0,3223	0,3851	0,3903	0,4197	0,4045

Table 30. Test 2 Classification confidence weighted Accuracy Results (cont’d)

ResNet101 - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5229	0,5270	0,5088	0,5015	0,5004	0,5002	0,5002	0,5002	0,5002	0,5002
	0.7K	0,5338	0,5005	0,5011	0,5007	0,5006	0,5004	0,5004	0,5006	0,5006	0,5007
	1.75K	0,7585	0,7446	0,7577	0,7666	0,7701	0,7594	0,7758	**0,7951**	0,7845	0,7744
	3.5K	0,5515	0,5643	0,5816	0,6085	0,6030	0,6009	0,6288	0,6646	0,6599	0,6057
	7K	0,7346	0,7768	0,7488	0,7616	0,7770	0,7553	0,7507	0,7489	0,7894	0,7622
	14K	0,6287	0,6265	0,6287	0,6188	0,5944	0,5658	0,5640	0,5732	0,5550	0,5636
	21K	0,6762	0,6850	0,6618	0,7094	0,7307	0,6868	0,6783	0,6547	0,6316	0,6407
	28K	0,6185	0,7315	0,7012	0,7106	0,7328	0,7459	0,7456	0,7393	0,7547	0,7374

ResNet152 - Test2		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,4679	0,4964	0,4957	0,4943	0,4950	0,4948	0,4954	0,4939	0,4955	0,4950
	0.7K	0,5003	0,5001	0,5027	0,5049	0,5038	0,5023	0,5037	0,5022	0,5024	0,5015
	1.75K	0,5060	0,5070	0,5094	0,5087	0,5079	0,5084	0,5024	0,5092	0,5062	0,5061
	3.5K	0,4996	0,4996	0,4996	0,4991	0,4999	0,4976	0,4986	0,4989	0,4996	0,4992
	7K	**0,6349**	0,5805	0,5942	0,5761	0,5545	0,5714	0,5545	0,5532	0,5588	0,5573
	14K	0,4739	0,4784	0,4756	0,4755	0,4772	0,4732	0,4781	0,4693	0,4709	0,4702
	21K	0,5192	0,5085	0,5047	0,5294	0,5103	0,5381	0,5297	0,5328	0,5245	0,5602
	28K	0,5077	0,5104	0,5061	0,5012	0,5059	0,5022	0,5063	0,5085	0,5119	0,4998

Table 31. Test 3 Classification confidence weighted Accuracy Results

AlexNet - Test 3		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,6209	0,6080	0,6193	0,6293	0,6290	0,6334	0,6364	0,6394	0,6465	0,6491
	0.7K	**0,8812**	0,6536	0,5961	0,5969	0,6137	0,6371	0,6476	0,6457	0,6423	0,6414
	1.75K	0,6960	0,6938	0,7433	0,7489	0,7434	0,7431	0,7432	0,7437	0,7429	0,7431
	3.5K	0,6895	0,6880	0,6673	0,6624	0,6550	0,6526	0,6472	0,6467	0,6446	0,6445
	7K	0,8355	0,7863	0,7342	0,7044	0,7126	0,7172	0,7169	0,7058	0,7076	0,7065
	14K	0,7843	0,7395	0,7381	0,7495	0,7198	0,7263	0,7322	0,7188	0,7205	0,7194
	21K	0,6681	0,6427	0,6774	0,6729	0,6555	0,6433	0,6511	0,6405	0,6552	0,6581
	28K	0,7524	0,7657	0,7585	0,7595	0,7648	0,7611	0,7606	0,7620	0,7565	0,7552

VGG16 - Test3		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,8532	0,8822	0,7837	0,7649	0,7618	0,7668	0,7715	0,7744	0,7799	0,7807
	0.7K	0,8358	0,8060	0,8260	0,8295	0,8259	0,8270	0,8247	0,8247	0,8250	0,8252
	1.75K	0,8709	0,8627	0,7001	**0,9818**	0,9731	0,9741	0,9744	0,9744	0,9744	0,9742
	3.5K	0,8700	0,8556	0,8565	0,8561	0,8582	0,8588	0,8593	0,8575	0,8598	0,8601
	7K	0,7157	0,7755	0,7043	0,6960	0,6959	0,6942	0,6925	0,6894	0,6893	0,6890
	14K	0,7320	0,7306	0,7439	0,7861	0,7768	0,7668	0,7809	0,7808	0,7835	0,7871
	21K	0,9575	0,9544	0,8260	0,8418	0,8407	0,8585	0,8865	0,8625	0,8767	0,8671
	28K	0,8539	0,8650	0,8405	0,8688	0,8804	0,8830	0,8911	0,8988	0,8986	0,8974

VGG19 - Test3		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,6285	0,6191	0,6049	0,5253	0,9048	0,8336	0,8543	0,8481	0,8441	0,8418
	0.7K	0,6987	0,6637	0,7079	0,7539	0,7443	0,7386	0,7363	0,7315	0,7304	0,7295
	1.75K	0,8742	0,9280	0,9195	0,9215	0,9239	0,9239	0,9236	0,9235	0,9234	0,9235
	3.5K	**0,9700**	0,8983	0,8817	0,8977	0,8987	0,8988	0,8989	0,8987	0,8991	0,9003
	7K	0,7982	0,7816	0,7899	0,7866	0,7870	0,7874	0,7886	0,7883	0,7908	0,7908
	14K	0,7423	0,6512	0,6193	0,6347	0,6252	0,6140	0,6216	0,6185	0,6143	0,6186
	21K	0,9551	0,9416	0,9239	0,9108	0,9121	0,9245	0,9245	0,9218	0,9258	0,9255
	28K	0,7306	0,8513	0,9109	0,9008	0,8951	0,8978	0,8944	0,8969	0,8934	0,8935

GoogleNet - Test3		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,6729	0,6989	0,7122	0,7254	0,7366	0,7404	0,7437	0,7382	0,7322	0,7261
	0.7K	0,8377	0,8377	0,8405	0,8340	0,8332	0,8303	0,8290	0,8291	0,8293	0,8263
	1.75K	0,9124	0,9143	0,9036	0,9144	0,9150	0,9186	0,9224	0,9247	0,9305	**0,9308**
	3.5K	0,6821	0,6717	0,6945	0,7068	0,7165	0,7156	0,7144	0,7171	0,7183	0,7216
	7K	0,7974	0,7796	0,7522	0,7440	0,7531	0,7559	0,7579	0,7537	0,7568	0,7549
	14K	0,7375	0,6918	0,7055	0,7012	0,6679	0,6842	0,6851	0,6728	0,6723	0,6810
	21K	0,6577	0,6452	0,6390	0,6347	0,6239	0,6010	0,6095	0,5901	0,6068	0,6060
	28K	0,8331	0,8044	0,8015	0,7731	0,7853	0,7751	0,7664	0,7876	0,7659	0,7451

ResNet50 - Test3		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	**0,6673**	0,5645	0,5012	0,4996	0,5000	0,5000	0,5000	0,5000	0,5000	0,5000
	0.7K	0,5412	0,5830	0,5866	0,5948	0,6012	0,5943	0,6014	0,6070	0,6055	0,5921
	1.75K	0,5056	0,5055	0,5086	0,5082	0,5040	0,5041	0,5014	0,5034	0,5007	0,5087
	3.5K	0,4864	0,4862	0,4888	0,4774	0,4761	0,4675	0,4718	0,4755	0,4935	0,4816
	7K	0,4715	0,4684	0,4586	0,4694	0,4680	0,4734	0,4678	0,4613	0,4737	0,4719
	14K	0,5010	0,5053	0,5106	0,5021	0,5092	0,5084	0,5167	0,5101	0,5048	0,5080
	21K	0,4894	0,4865	0,4865	0,4906	0,4678	0,4612	0,4738	0,4542	0,4551	0,4667
	28K	0,4963	0,4947	0,4975	0,4996	0,4938	0,4868	0,4947	0,4942	0,4971	0,4948

Table 31. Test 3 Classification confidence weighted Accuracy Results (cont’d)

ResNet101 - Test3		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,7208	0,6424	0,5244	0,5020	0,4991	0,4991	0,4991	0,4988	0,4989	0,4997
	0.7K	**0,7606**	0,5245	0,5330	0,5380	0,5407	0,5307	0,5372	0,5462	0,5486	0,5461
	1.75K	0,5403	0,5359	0,5365	0,5335	0,5329	0,5389	0,5325	0,5412	0,5329	0,5398
	3.5K	0,6071	0,6049	0,6099	0,6402	0,6204	0,6141	0,6530	0,6609	0,6606	0,6433
	7K	0,6249	0,6213	0,6165	0,6292	0,6295	0,6292	0,6167	0,6280	0,6259	0,6287
	14K	0,5506	0,5354	0,5275	0,5276	0,5226	0,5218	0,5265	0,5024	0,5245	0,5269
	21K	0,6100	0,6070	0,5977	0,6178	0,5962	0,5850	0,6013	0,6075	0,5859	0,5962
	28K	0,5478	0,5806	0,5753	0,6011	0,5862	0,5922	0,6159	0,6257	0,6094	0,6044

ResNet152 - Test3		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5487	0,5132	0,5079	0,4964	0,4968	0,5000	0,4984	0,4997	0,4975	0,4979
	0.7K	0,5046	0,5030	0,5150	0,5337	0,5349	0,5338	0,5331	0,5309	0,5268	0,5400
	1.75K	0,5006	0,5007	0,5007	0,5007	0,5007	0,5008	0,5004	0,5008	0,5007	0,5006
	3.5K	0,5022	0,5033	0,5018	0,5032	0,5030	0,5011	0,5057	0,5035	0,5029	0,5071
	7K	0,5259	0,5237	0,5148	0,5145	0,5101	0,5176	0,5124	0,5111	0,5123	0,5098
	14K	0,5233	0,5189	0,5107	0,5051	0,5175	**0,5497**	0,5090	0,5132	0,5093	0,5167
	21K	0,5188	0,5134	0,5026	0,5212	0,5077	0,5296	0,5239	0,5237	0,5213	0,5424
	28K	0,4931	0,4900	0,5029	0,5019	0,4977	0,5070	0,5026	0,4914	0,5068	0,5134

Table 32. Test 4 Classification confidence weighted Accuracy Results

AlexNet - Test 4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5841	0,5759	0,5799	0,5840	0,5883	0,5950	0,5991	0,6040	0,6076	0,6079
	0.7K	0,7754	0,5882	0,5526	0,5508	0,5583	0,5666	0,5696	0,5678	0,5669	0,5663
	1.75K	0,7949	0,8091	0,8506	0,8585	0,8486	0,8461	0,8429	0,8431	0,8440	0,8430
	3.5K	0,7190	0,7156	0,6958	0,6916	0,6822	0,6788	0,6755	0,6745	0,6715	0,6713
	7K	**0,8821**	0,8449	0,8130	0,7952	0,7998	0,7998	0,7983	0,7935	0,7925	0,7904
	14K	0,7755	0,7104	0,7095	0,7196	0,6829	0,6901	0,6950	0,6866	0,6842	0,6870
	21K	0,8501	0,8388	0,8630	0,8615	0,8543	0,8517	0,8524	0,8483	0,8541	0,8559
	28K	0,7984	0,8128	0,8001	0,7965	0,8015	0,7980	0,7991	0,8041	0,7984	0,7973

VGG16 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,8261	0,8680	0,8936	0,8846	0,8850	0,8863	0,8881	0,8873	0,8893	0,8882
	0.7K	0,6813	0,8553	0,8294	0,8150	0,8183	0,8273	0,8301	0,8305	0,8301	0,8300
	1.75K	0,7835	0,8816	0,7823	**0,9633**	0,9476	0,9465	0,9484	0,9485	0,9494	0,9471
	3.5K	0,8412	0,8252	0,8282	0,8250	0,8233	0,8250	0,8249	0,8232	0,8258	0,8236
	7K	0,8372	0,8487	0,8517	0,8466	0,8444	0,8437	0,8430	0,8425	0,8411	0,8407
	14K	0,8298	0,8069	0,8397	0,8819	0,8655	0,8589	0,8738	0,8734	0,8748	0,8766
	21K	0,8769	0,8725	0,8941	0,8933	0,8911	0,8935	0,8870	0,8887	0,8854	0,8854
	28K	0,8400	0,8081	0,8212	0,8214	0,8302	0,8294	0,8256	0,8212	0,8214	0,8210

VGG19 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,6900	0,5467	0,5678	0,5059	0,8962	0,8474	0,8606	0,8570	0,8546	0,8537
	0.7K	0,6853	0,7273	0,7587	0,7721	0,7711	0,7733	0,7764	0,7735	0,7735	0,7756
	1.75K	0,7914	0,8323	0,8299	0,8282	0,8278	0,8286	0,8287	0,8287	0,8286	0,8284
	3.5K	0,8547	0,8949	0,8926	0,8976	**0,8987**	0,8976	0,8975	0,8975	0,8953	0,8952
	7K	0,7374	0,8147	0,8171	0,8200	0,8207	0,8221	0,8237	0,8251	0,8262	0,8285
	14K	0,7975	0,7940	0,7499	0,7918	0,7664	0,7410	0,7534	0,7514	0,7494	0,7548
	21K	0,7764	0,8345	0,8256	0,8364	0,8391	0,8348	0,8340	0,8327	0,8316	0,8309
	28K	0,8551	0,8473	0,8540	0,8614	0,8580	0,8619	0,8620	0,8615	0,8634	0,8633

GoogleNet - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5415	0,5789	0,6088	0,6543	0,6893	0,7091	0,7238	0,7414	0,7552	0,7633
	0.7K	0,8419	0,8646	0,8944	0,9078	0,9158	0,9181	0,9200	0,9207	0,9231	**0,9233**
	1.75K	0,7745	0,8572	0,8762	0,8896	0,8968	0,9028	0,9108	0,9122	0,9131	0,9134
	3.5K	0,8757	0,8809	0,9106	0,9184	0,9167	0,9167	0,9154	0,9153	0,9150	0,9131
	7K	0,8898	0,8854	0,8739	0,8661	0,8766	0,8840	0,8914	0,8823	0,8922	0,8933
	14K	0,7836	0,7169	0,7252	0,7188	0,6762	0,6941	0,6928	0,6717	0,6677	0,6793
	21K	0,7173	0,7024	0,7011	0,6923	0,6860	0,6537	0,6713	0,6492	0,6762	0,6790
	28K	0,9106	0,8830	0,8896	0,8659	0,8885	0,8820	0,8701	0,8899	0,8768	0,8548

ResNet50 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	**0,7066**	0,6834	0,5475	0,5053	0,5056	0,4991	0,4994	0,5007	0,5009	0,5019
	0.7K	0,6446	0,6089	0,5603	0,5451	0,5498	0,5383	0,5396	0,5471	0,5499	0,5435
	1.75K	0,5291	0,5227	0,5239	0,5684	0,5335	0,5485	0,5186	0,5392	0,5203	0,5540
	3.5K	0,5182	0,5298	0,5127	0,5038	0,4956	0,4944	0,5010	0,5113	0,5046	0,5177
	7K	0,4566	0,4470	0,4314	0,4381	0,4394	0,4573	0,4450	0,4297	0,4406	0,4390
	14K	0,5257	0,5384	0,5412	0,5317	0,5414	0,5318	0,5677	0,5476	0,5480	0,5486
	21K	0,5379	0,5282	0,5189	0,5279	0,5372	0,5594	0,5381	0,5914	0,5285	0,5685
	28K	0,5047	0,5007	0,5058	0,5044	0,4989	0,4989	0,5022	0,5033	0,5045	0,4983

Table 32. Test 4 Classification confidence weighted Accuracy Results (cont'd)

ResNet101 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5611	0,6078	0,5737	0,5401	0,5151	0,5141	0,5143	0,5162	0,5126	0,5124
	0.7K	0,5770	0,5504	0,5825	0,5938	0,5874	0,5691	0,5786	0,5847	0,5948	0,6119
	1.75K	0,6494	0,6539	0,6700	0,6707	0,6588	0,6944	0,6590	0,6960	0,6566	0,6819
	3.5K	0,6198	0,6182	0,6354	0,6659	0,6440	0,6546	0,6612	0,6660	0,6642	0,6558
	7K	0,6308	0,6677	0,6632	0,6559	0,6704	0,6807	0,6849	0,7002	0,6865	0,6916
	14K	0,5903	0,6046	0,5788	0,5803	0,5844	0,5735	0,5741	0,5771	0,5596	0,5731
	21K	0,6618	0,6816	0,6573	0,7027	0,6845	0,6751	0,6888	0,7244	0,7103	**0,7403**
	28K	0,5691	0,5745	0,5786	0,5951	0,5954	0,6153	0,6124	0,6171	0,5987	0,6145

ResNet152 - Test4		**Number of Epoch**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training Dataset Size	**0.35K**	0,5024	0,5481	0,5301	0,5127	0,5156	0,5124	0,5105	0,5032	0,5115	0,5088
	0.7K	0,5296	0,5095	0,5184	0,5411	0,5381	0,5363	0,5389	0,5330	0,5381	0,5357
	1.75K	0,5086	0,5116	0,5162	0,5133	0,5129	0,5117	0,5130	0,5152	0,5141	0,5137
	3.5K	0,5096	0,5071	0,5073	0,5096	0,5075	0,5089	0,5104	0,5101	0,5083	0,5112
	7K	0,5377	0,5308	0,5317	0,5306	0,5256	0,5332	0,5298	0,5291	0,5228	0,5299
	14K	0,5810	0,5818	0,5513	0,5205	0,5516	0,5880	0,5524	0,5405	0,5306	0,5541
	21K	0,5553	0,5433	0,5232	0,5602	0,5421	0,5719	0,5619	0,5701	0,5567	**0,6212**
	28K	0,5222	0,5413	0,5361	0,5185	0,5333	0,5221	0,5268	0,5426	0,5646	0,5581

APPENDIX III

CRACK BASED SEMANTIC SEGMENTATION METRIC RESULTS

Table 33. Crack based semantic segmentation metric results (per sample)

Number of Objects						**Mean Orientation (Degrees)**				
Test 1 - Building - Concrete						**Test 1 - Building - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
1	6	5	55	4060		1	61,00	-72,01	-1,06	10,81
2	1	1	18	628		2	-32,55	-32,53	-29,73	-28,40
3	3	4	105	1197		3	-78,51	-78,98	-30,48	-43,45
4	3	2	59	883		4	-61,49	44,52	-17,62	-34,33
5	2	1	20	639		5	-72,80	-73,15	-73,23	-17,05
6	1	10	39	805		6	-48,17	-45,36	-50,07	-44,24
7	2	1	25	720		7	-62,16	-65,12	-29,99	-53,47
8	3	3	92	1154		8	-9,39	-12,03	-8,18	-6,35
9	2	4	61	830		9	-7,39	-12,49	-15,25	-12,30
10	2	2	46	705		10	-12,33	-12,35	-14,11	-9,22
11	2	3	53	756		11	-11,10	-11,05	-10,83	-6,55
12	1	1	73	727		12	-19,77	-20,13	-11,82	-14,78
13	1	5	78	793		13	-14,79	-8,41	-2,93	-9,62
14	3	2	52	734		14	-5,91	-3,73	-1,01	-4,32
15	4	4	76	580		15	-6,80	-7,64	-6,55	-10,00
16	5	1	92	775		16	1,75	-4,38	-6,75	-1,14
17	2	1	70	877		17	-54,44	-56,31	-43,91	-41,39
18	2	6	26	598		18	-51,88	-55,00	-51,19	-48,62
19	2	1	50	728		19	-53,44	-46,00	-42,06	-43,39
20	2	3	33	814		20	-21,80	-23,21	-20,95	-19,05
21	1	2	45	711		21	-11,08	-11,06	-9,33	-11,55
22	6	5	69	816		22	-28,43	-25,55	-21,73	-23,80
23	5	9	54	895		23	6,88	3,57	11,52	10,12
24	2	5	69	1280		24	10,00	18,56	9,87	7,31
25	4	34	126	1464		25	-17,60	-14,52	-16,79	-10,10
26	3	8	34	820		26	-14,38	-15,60	-16,35	-12,86
27	6	12	55	1013		27	9,14	16,46	3,37	4,92
28	4	6	53	910		28	-25,13	-23,09	-27,39	-21,64
29	8	6	68	1027		29	-20,60	-19,32	-9,88	-10,43
30	2	3	73	1000		30	-8,64	-9,14	-10,06	-9,89
31	8	6	58	898		31	-14,45	-3,74	-8,99	-9,91
32	3	5	45	787		32	-15,95	-5,47	-8,58	-9,11
33	4	12	60	1767		33	-41,56	-57,68	7,24	-4,55
34	4	97	75	1494		34	54,10	51,19	47,74	39,34
35	1	7	35	773		35	-75,55	-82,47	-2,99	0,80
36	2	3	33	844		36	-38,93	-39,81	-38,65	-34,89
37	3	9	43	1112		37	-46,64	-50,88	-42,71	-40,68
38	2	15	46	936		38	-25,88	-26,36	-33,09	-28,08
39	1	9	35	891		39	-32,65	-30,92	-34,81	-35,18

Table 33. Crack based semantic segmentation metric results (per sample) (cont'd)

Number of Objects						**Mean Orientation (Degrees)**				
Test 1 - Building - Concrete						**Test 1 - Building - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
40	1	4	28	734		40	-19,53	-18,75	-19,93	-20,38
41	1	28	49	1161		41	-44,21	-43,14	-43,42	-37,94
42	4	8	44	1062		42	-45,08	-45,23	-44,05	-39,67
43	2	2	34	1094		43	-43,64	-43,14	-42,54	-33,25
44	1	79	42	777		44	-16,36	-16,23	-12,44	-13,12
45	2	8	53	1154		45	-39,52	-40,22	-36,20	-35,81
46	1	5	68	1645		46	-88,67	-86,90	27,86	-12,10
47	2	8	48	1166		47	61,71	61,46	63,66	62,05
48	3	7	50	1117		48	39,10	41,96	42,35	39,77
49	2	11	79	1188		49	27,60	24,00	25,64	27,23
50	1	4	33	921		50	9,38	9,20	12,81	17,43
51	1	5	31	798		51	31,71	31,11	25,78	29,94
52	2	31	24	927		52	0,56	-0,55	0,04	-0,93
53	2	7	28	1457		53	-7,22	-7,01	-8,36	-7,45
54	6	5	116	1243		54	-3,96	-19,54	-6,93	-7,41
55	5	61	118	1745		55	9,84	13,94	-0,98	3,73
56	1	40	57	1223		56	-23,78	-28,28	-21,18	-19,41
57	2	12	49	1244		57	-10,14	-7,56	-13,83	-10,98
58	2	68	35	857		58	2,88	0,15	2,86	2,73
59	2	10	42	976		59	-0,38	1,24	-0,09	-4,93
60	1	9	33	961		60	-15,31	-13,45	-13,73	-17,88
61	1	13	130	1903		61	-24,93	-32,52	-12,61	-7,21
62	1	12	179	1905		62	-1,37	6,31	-8,14	-9,24
63	2	10	51	875		63	39,86	39,88	34,86	36,35
64	1	3	28	845		64	65,00	64,07	50,49	33,29
65	2	17	40	1145		65	53,86	64,73	54,95	29,15
66	1	23	24	938		66	86,31	85,86	-12,03	15,14
67	1	7	33	1016		67	85,06	85,19	21,59	17,44
68	2	2	34	794		68	-71,66	87,93	-55,68	-2,25
69	1	9	42	728		69	-75,67	-74,63	-17,66	-37,24
70	3	7	56	984		70	-74,43	-76,55	-63,49	-60,20
71	1	7	33	875		71	85,79	78,07	25,06	1,03
72	4	24	104	1598		72	-70,70	-74,83	-49,88	-32,98
73	1	47	38	1135		73	72,51	62,41	18,42	38,15
74	2	14	51	1304		74	72,68	70,30	72,63	54,52
75	3	39	60	1703		75	69,78	59,65	42,15	47,48
76	3	13	40	1441		76	68,71	67,62	51,86	53,82
77	2	20	68	1617		77	36,44	33,31	33,20	31,97
78	1	23	66	1214		78	20,38	16,11	17,55	22,88
79	4	6	38	988		79	16,20	20,60	15,75	14,42
80	2	16	38	948		80	18,88	23,22	21,86	19,19
81	2	10	108	1242		81	-7,40	-10,75	-0,74	-1,26
82	4	3	150	1205		82	-9,51	-32,92	-12,49	-5,29
83	2	14	42	870		83	13,31	12,90	9,88	8,56
84	1	7	98	1508		84	-18,25	-20,88	-11,84	-12,86
85	3	10	39	683		85	1,19	6,95	3,74	2,77
86	2	9	27	530		86	5,33	6,81	5,03	3,68
87	1	5	36	1638		87	-78,64	-66,29	-53,64	-43,37
88	5	7	41	837		88	-5,29	-5,88	-9,78	-8,92
89	3	6	84	2280		89	4,06	1,25	-5,36	-5,93
90	1	3	8	663		90	1,43	2,77	-1,42	-3,63
91	2	5	18	674		91	0,12	2,78	1,13	-0,02
92	2	6	23	844		92	6,07	7,37	6,43	7,58
93	3	8	37	836		93	-3,07	-1,06	-7,62	-8,30
94	6	9	44	948		94	-2,64	-6,73	-4,57	-5,95
95	3	23	61	1122		95	9,52	9,11	-1,17	2,88
96	4	5	74	1094		96	-23,14	-28,02	-21,72	-14,68
97	3	8	67	1050		97	4,08	-9,44	-9,81	-6,50
98	2	20	65	1560		98	-12,11	-9,16	-9,55	-6,58

Table 33. Crack based semantic segmentation metric results (per sample) (cont'd)

Number of Objects						Mean Orientation (Degrees)				
Test 1 - Building - Concrete						Test 1 - Building - Concrete				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
99	2	2	21	3814		99	56,03	62,25	17,73	11,47
100	3	2	27	4816		100	-70,21	-88,14	-84,98	1,51
101	3	5	28	1115		101	-63,04	-58,54	-62,78	-59,21
102	1	9	22	2514		102	-34,81	-35,29	-34,46	-30,61
103	2	27	74	3707		103	-34,20	-32,86	-30,96	-28,19
104	2	4	29	1065		104	-6,58	-9,30	-6,30	-5,80
105	1	2	23	911		105	-0,12	0,03	-2,11	-5,37
106	1	3	28	1120		106	61,97	61,91	61,72	32,48
107	2	8	29	3299		107	-2,22	-0,11	3,81	-1,88
108	2	17	54	1444		108	2,67	8,61	5,56	1,93
109	2	4	21	5369		109	77,51	77,08	25,44	7,91
110	2	4	64	1793		110	-77,19	-76,05	-63,16	-7,54
111	4	8	101	2838		111	-47,58	-48,03	-29,98	-21,44
112	2	19	16	660		112	-58,24	-58,23	-50,17	-40,77
113	3	8	61	1081		113	-39,16	-35,13	-30,21	-28,65
114	1	7	95	3358		114	-13,24	-5,49	-8,38	10,74
115	1	4	56	1400		115	-22,65	-23,04	-16,37	-18,80
116	3	9	38	1005		116	-11,35	-10,60	-18,46	-17,71
117	1	2	25	8812		117	85,77	84,69	23,90	18,17
118	3	32	49	2429		118	-20,24	-20,64	-16,26	-5,33
119	2	35	144	3059		119	-15,60	-16,99	-11,25	5,09
120	2	46	193	6160		120	-68,87	-37,04	-47,22	-7,05
121	3	32	65	1104		121	-50,26	-50,35	-55,14	-43,88
122	1	2	32	734		122	87,11	88,05	22,65	12,70
123	2	5	31	855		123	74,74	74,81	68,89	34,31
124	3	10	54	1042		124	37,21	38,03	36,98	37,56
125	3	5	69	11570		125	20,35	24,42	17,29	8,17
126	1	2	44	4387		126	12,89	13,48	22,30	10,26
127	6	11	76	8953		127	2,80	5,92	7,53	3,61
128	2	6	24	1043		128	60,06	52,93	58,69	55,29
129	1	11	29	998		129	83,04	73,39	75,10	19,62
130	2	4	17	810		130	-7,97	-9,95	-7,31	-10,82
131	4	12	45	1066		131	-8,94	-9,43	-11,03	-9,59
132	1	8	10	790		132	-19,30	-18,76	-18,25	-18,96
133	2	2	33	942		133	-11,56	-18,66	-11,09	-9,11
134	2	1	7	626		134	-26,13	-19,82	-26,28	-18,49
135	2	2	16	706		135	8,27	8,63	9,38	8,08
136	1	1	14	598		136	16,21	17,03	10,02	10,73
137	1	1	14	709		137	9,34	9,87	13,46	2,42
138	2	3	15	646		138	1,40	4,78	0,98	-0,87
139	1	1	11	490		139	-3,17	-2,75	1,08	-0,12
140	1	1	11	495		140	-6,76	-6,25	-1,20	-4,83
141	2	5	22	866		141	1,48	-0,84	3,69	-1,15
142	2	8	29	816		142	1,66	0,10	4,62	4,39
143	5	15	70	1249		143	3,80	-0,11	6,02	2,90
144	1	9	24	770		144	-5,69	-7,54	-7,43	0,64
145	1	9	40	1128		145	-35,06	-37,03	-34,01	-26,26
146	2	4	27	848		146	-24,21	-23,40	-20,16	-21,13
147	1	8	28	902		147	-18,04	-18,64	-13,64	-17,05
148	2	9	64	1061		148	23,54	23,53	20,57	22,81
149	3	7	24	887		149	30,52	28,74	30,43	27,56
150	2	3	50	842		150	29,17	29,32	31,19	26,29
151	2	13	28	684		151	-39,85	-40,81	-38,73	-40,82
152	2	3	36	691		152	-29,92	-26,55	-24,37	-27,97
153	1	3	21	681		153	3,03	2,37	-2,29	3,70
154	2	13	28	1026		154	-7,64	-7,60	-9,48	-4,56
155	3	44	27	700		155	-28,29	-28,55	-28,20	-24,18
156	6	64	43	875		156	-15,97	-15,43	-11,82	-10,27
157	3	3	8	506		157	-8,29	-5,26	-8,49	-9,10

Table 33. Crack based semantic segmentation metric results (per sample) (cont'd)

Number of Objects						Mean Orientation (Degrees)				
Test 1 - Building - Concrete						Test 1 - Building - Concrete				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
158	1	1	32	856		158	16,87	15,92	13,07	14,57
159	3	14	74	1108		159	33,69	26,32	28,44	27,16
160	3	16	40	813		160	30,52	39,16	36,56	39,35
161	1	7	68	1039		161	21,40	4,65	11,97	3,21
162	4	31	84	1128		162	-23,75	-22,66	-24,16	-14,46
163	1	4	52	826		163	-1,70	-2,07	5,00	0,40
164	1	15	26	584		164	-33,77	-31,37	-24,22	-22,53
165	4	1	32	930		165	-26,31	-34,34	-33,82	-26,29
166	2	5	60	1255		166	55,01	56,00	39,95	36,22
167	1	8	34	2253		167	75,73	69,60	51,41	47,98
168	1	3	18	804		168	78,51	80,20	45,28	44,40
169	1	7	30	769		169	30,18	29,71	32,09	25,37
170	2	6	14	685		170	34,84	30,34	29,59	30,04
171	1	5	17	744		171	34,33	33,43	33,05	29,62
172	1	6	18	814		172	-5,93	-13,48	-12,76	-16,90
173	1	4	45	967		173	2,30	-0,02	3,55	3,30
174	2	3	27	595		174	6,44	6,43	3,37	5,36
175	2	12	41	872		175	-2,32	10,21	-2,60	-6,01
176	3	4	23	723		176	14,46	9,53	5,83	5,98
177	2	1	21	706		177	-11,91	-11,71	-8,84	-9,59
178	3	7	57	945		178	-14,49	-6,64	-9,35	-4,25
179	1	9	16	516		179	-13,09	-13,68	-11,16	-13,26
180	2	2	37	576		180	-14,51	-13,43	-10,25	-11,65
181	2	3	30	714		181	-7,34	-1,99	-12,08	-6,98
182	2	4	46	746		182	-21,28	-32,08	-27,52	-19,99
183	2	5	20	600		183	-2,40	0,64	-4,10	-2,55
184	3	17	19	742		184	-12,24	-12,34	-9,70	-10,42
185	2	35	144	2330		185	76,21	-11,08	-1,29	-3,11
186	4	5	72	1281		186	-54,71	-45,06	-43,39	-37,66
187	2	5	63	1183		187	-19,41	-17,04	-29,67	-33,35
188	4	5	61	978		188	-24,66	-23,97	-27,35	-23,25
189	2	17	45	862		189	-34,97	-39,28	-29,01	-28,63
190	2	39	56	898		190	-16,62	-18,13	-18,98	-10,67
191	5	17	87	1536		191	-19,32	-22,84	-17,59	-15,54
192	4	9	78	1001		192	-25,62	-23,40	-25,69	-20,02
193	2	9	56	1163		193	4,64	9,44	-3,19	1,82
194	1	1	47	693		194	-7,34	-6,60	-7,34	-4,03
195	2	8	58	980		195	-14,50	-9,66	-7,24	-6,59
196	2	6	41	947		196	-39,90	-40,65	-39,84	-38,34
197	1	5	29	835		197	-26,83	-37,65	-32,18	-22,90
198	2	5	39	732		198	-23,74	-26,20	-32,05	-28,94
199	1	3	56	815		199	-33,97	-33,51	-30,58	-37,51
200	1	1	34	729		200	-34,42	-34,18	-38,18	-35,69
201	3	4	92	1252		201	16,51	20,92	21,63	19,29
202	1	1	31	984		202	-42,23	-42,20	-42,48	-37,75
203	2	9	53	892		203	-33,81	-39,51	-32,79	-32,33
204	2	9	43	1117		204	-11,82	-14,62	-11,14	-10,18
205	1	6	31	674		205	0,44	1,18	-1,91	1,60
206	2	45	57	1032		206	-28,36	-27,84	-21,04	-18,21
207	1	1	24	583		207	-11,24	-11,19	-10,94	-7,89
208	1	9	55	965		208	-14,45	-7,01	-24,30	-18,68
209	1	4	23	542		209	-25,21	-28,03	-27,12	-27,86
210	3	5	86	903		210	6,31	-0,23	9,46	20,89
211	2	3	28	1270		211	68,21	66,77	59,25	22,04
212	4	4	42	855		212	-4,07	-4,81	-4,65	-3,85
213	4	2	31	771		213	-0,78	-4,05	-5,22	-2,04
214	1	1	26	622		214	-20,29	-20,95	-18,60	-16,59
215	3	2	32	716		215	-16,78	-18,55	-17,64	-17,01
216	2	5	49	730		216	27,38	26,99	24,19	23,02

Table 33. Crack based semantic segmentation metric results (per sample) (cont’d)

Number of Objects						**Mean Orientation (Degrees)**				
Test 1 - Building - Concrete						**Test 1 - Building - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
217	2	3	39	820		217	33,09	30,84	27,01	26,27
218	1	3	47	1887		218	2,96	4,28	1,65	1,39
219	2	4	39	785		219	-9,38	-11,72	-6,71	-4,23
220	2	4	36	786		220	-26,39	-27,98	-23,95	-17,37
221	2	2	22	876		221	-19,54	-20,08	-12,75	-14,22
222	3	6	39	982		222	16,14	18,22	17,44	14,93
223	2	6	26	717		223	-28,16	-31,45	-30,35	-27,80
224	3	6	37	916		224	-27,35	-23,43	-29,23	-22,50
225	3	17	27	669		225	-19,39	-23,46	-16,67	-17,85
226	2	5	18	916		226	-32,28	-31,54	-28,70	-25,71
227	3	7	37	712		227	-19,75	-19,83	-14,76	-15,60
228	2	8	52	1787		228	-4,47	1,79	-6,65	32,98
229	1	8	48	1236		229	-8,57	-10,27	-6,71	-5,67
230	5	8	33	758		230	-7,98	-4,70	-9,84	-11,09
231	3	21	39	834		231	38,00	37,70	36,49	39,47
232	4	27	86	1486		232	27,49	29,23	26,90	23,49
233	1	6	46	660		233	14,36	13,99	6,28	8,30
234	1	8	38	651		234	5,62	5,69	8,60	6,88
235	6	6	84	1083		235	4,10	-15,21	-11,67	-6,07
236	3	3	36	681		236	-1,05	-0,41	-3,69	-1,66
237	2	8	42	714		237	5,14	7,69	11,85	7,76
238	5	5	115	1577		238	14,12	-7,32	6,26	0,56
239	2	22	71	1851		239	11,14	59,33	11,91	15,38
240	3	8	22	692		240	-11,60	-12,11	-9,30	-10,28
241	5	7	70	4851		241	-0,97	2,13	-2,67	-0,71
242	1	4	31	788		242	1,64	0,78	1,74	3,17
243	1	3	27	727		243	-1,10	-1,09	-1,18	1,72
244	1	8	32	567		244	-4,95	-4,53	-7,20	-6,06
245	3	12	126	2005		245	-0,05	3,65	-9,46	-3,44
246	1	6	80	3268		246	-74,27	28,48	-38,35	-13,83
247	1	7	31	3062		247	15,10	15,17	24,39	16,40
248	2	4	55	2153		248	71,97	49,69	2,41	26,66
249	5	7	83	1763		249	56,95	59,60	52,79	54,70
250	4	12	82	2195		250	26,27	25,50	23,38	23,53
251	1	3	47	4857		251	7,04	6,64	6,00	3,25
252	1	5	42	5437		252	87,74	36,17	-35,31	-11,55
253	2	2	39	4175		253	47,30	44,43	45,70	32,60
254	3	3	44	3012		254	39,88	37,08	35,84	27,07
255	1	31	32	7871		255	-43,32	-49,86	-37,58	-24,11
256	2	3	36	5812		256	9,44	4,86	4,56	3,96
257	1	2	52	5463		257	-89,72	89,68	-17,29	-11,75
258	5	3	27	4033		258	-6,25	1,80	4,82	2,91
259	2	5	35	1174		259	27,28	30,04	26,98	26,20
260	1	6	26	2642		260	55,45	50,99	54,86	44,10
261	3	19	81	4309		261	49,34	49,81	54,78	38,37
262	2	7	24	1350		262	83,34	59,42	28,77	16,53
263	1	2	20	927		263	89,88	89,91	-24,49	26,94
264	1	3	19	1147		264	-28,15	-27,00	-26,43	-16,35
265	2	6	25	4903		265	-86,90	-84,24	-71,71	0,93
266	5	10	59	1520		266	-47,47	-73,31	-66,60	-11,65
267	1	3	26	6654		267	-13,41	-12,92	-12,53	5,62
268	6	4	66	2433		268	12,84	13,62	9,47	10,01
269	5	6	100	3024		269	36,52	42,52	39,54	34,47
270	2	3	14	752		270	31,76	30,58	34,90	27,02
271	1	9	59	1105		271	48,98	42,98	56,31	48,89
272	3	7	98	3488		272	73,20	68,28	13,39	-22,96
273	1	4	62	1601		273	67,31	66,91	69,98	43,00
274	4	8	33	1136		274	77,98	76,75	76,67	31,53
275	3	50	46	2785		275	66,52	69,23	72,10	7,93

Table 33. Crack based semantic segmentation metric results (per sample) (cont'd)

Number of Objects						**Mean Orientation (Degrees)**				
Test 1 - Building - Concrete						**Test 1 - Building - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
276	5	76	137	2843		276	72,99	69,86	28,39	-25,12
277	2	39	177	6928		277	10,64	1,36	13,38	-9,93
278	2	18	65	1154		278	20,16	15,67	17,56	16,68
279	2	2	31	820		279	-5,22	-1,93	1,54	-3,27
280	2	5	24	881		280	-15,56	-15,00	-18,58	-16,87
281	6	12	53	1210		281	-44,85	-46,00	-51,86	-45,66
282	3	5	59	12494		282	-68,85	-64,88	-60,33	-14,46
283	3	3	49	3855		283	-74,71	-67,11	-45,99	-17,80
284	5	9	70	9359		284	27,47	-77,81	-19,05	-8,68
285	2	8	23	1158		285	-30,11	-35,75	-27,63	-27,48
286	1	11	28	1131		286	-7,06	-11,35	-3,28	-13,50
287	2	4	19	893		287	82,11	72,88	81,04	57,82
288	6	7	50	1174		288	59,76	78,76	70,22	43,94
289	1	4	9	941		289	70,58	71,08	70,86	65,38
290	3	4	28	993		290	79,76	71,77	48,80	34,29
291	2	3	6	642		291	63,81	70,21	63,73	70,15
292	3	2	20	687		292	-80,95	-80,67	-74,10	-16,35
293	3	1	12	773		293	-70,65	-73,20	-70,51	-1,26
294	1	1	12	762		294	-80,90	-80,12	-77,16	-6,91
295	2	7	18	720		295	68,84	-84,49	54,74	-24,01
296	1	1	8	501		296	83,28	84,03	-39,84	-18,19
297	3	2	22	819		297	-61,04	89,27	-36,62	-8,24
298	3	13	28	833		298	-83,55	-89,37	20,89	-31,45
299	8	8	72	1372		299	-43,19	-83,26	-55,23	-18,59
300	1	13	29	814		300	84,44	82,58	44,99	4,51
301	1	13	50	1388		301	55,49	50,54	51,61	50,23
302	1	6	47	1151		302	55,15	53,09	52,21	51,45
303	2	16	20	888		303	63,51	67,25	64,57	64,88
304	1	10	25	966		304	72,06	69,81	14,17	31,74
305	2	10	68	1075		305	-65,99	-65,62	-60,14	-59,27
306	3	8	33	856		306	-59,61	-61,03	-59,36	-61,38
307	3	6	44	872		307	-60,85	-60,89	-57,40	-64,79
308	2	1	27	646		308	50,15	49,21	52,20	49,01
309	2	2	36	670		309	59,98	64,31	64,79	57,44
310	3	4	24	664		310	-84,68	-87,52	-22,71	-40,28
311	1	6	30	1130		311	81,91	82,18	51,72	26,14
312	3	17	29	720		312	61,66	56,32	56,19	42,93
313	5	11	44	924		313	70,29	40,40	63,96	44,66
314	6	3	7	550		314	44,69	82,46	46,58	33,08
315	5	7	35	868		315	-70,39	-73,96	-72,16	-61,17
316	5	9	65	1222		316	-60,13	-58,78	-60,18	-61,57
317	2	24	41	814		317	-46,83	-50,12	-45,88	-45,40
318	8	30	84	1148		318	12,82	16,38	17,10	14,45
319	1	7	53	878		319	88,94	84,66	-17,66	-11,92
320	4	11	30	578		320	52,75	28,87	54,55	45,85
321	6	1	32	925		321	62,05	55,81	57,10	58,13
322	6	8	55	1256		322	-25,02	-20,35	-25,92	-17,80
323	3	4	20	754		323	-7,49	-9,78	-7,40	-6,47
324	1	7	27	830		324	-59,79	-60,25	-58,10	-63,76
325	2	1	15	696		325	-53,87	-54,79	-59,53	-43,72
326	2	21	22	831		326	-54,25	-52,68	-54,13	-53,65
327	1	4	19	867		327	84,48	78,68	76,64	-15,02
328	1	2	45	998		328	-87,95	-89,87	-71,96	1,19
329	2	2	30	614		329	-83,14	-65,46	-75,17	-29,10
330	3	15	37	893		330	80,13	-30,93	-8,90	0,16
331	5	6	24	749		331	-78,22	-78,90	-14,36	-27,13
332	3	1	25	744		332	76,71	78,27	57,67	25,78
333	2	4	52	972		333	74,42	72,95	5,84	4,68
334	3	8	15	572		334	80,90	71,24	77,29	57,05

Table 33. Crack based semantic segmentation metric results (per sample) (cont'd)

Number of Objects						**Mean Orientation (Degrees)**				
Test 1 - Building - Concrete						**Test 1 - Building - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
335	1	7	36	659		335	75,87	76,71	28,80	20,41
336	2	4	29	730		336	82,32	-18,78	-8,41	-10,28
337	2	7	42	765		337	67,95	65,02	51,89	69,43
338	2	6	22	627		338	87,49	-89,25	23,54	5,94
339	4	15	17	681		339	77,53	77,20	78,01	49,65
340	3	6	68	1332		340	34,98	44,80	36,45	33,37
341	3	15	65	1246		341	65,21	69,43	58,41	54,30
342	3	4	57	978		342	64,86	57,55	52,28	33,80
343	2	13	42	914		343	54,90	43,95	44,29	22,36
344	1	38	49	984		344	73,33	71,02	67,24	47,57
345	4	21	70	1015		345	62,15	55,45	62,97	68,54
346	2	22	59	1139		346	-85,55	-79,08	67,20	26,24
347	1	2	46	658		347	82,58	83,20	19,75	39,43
348	1	8	60	1058		348	80,24	80,43	2,86	-2,89
349	1	6	43	1019		349	50,31	49,30	49,62	46,59
350	1	8	28	893		350	63,15	50,80	49,92	51,13
351	1	2	57	882		351	55,96	55,26	59,31	50,70
352	1	1	41	712		352	55,55	55,80	50,49	53,17
353	2	1	95	1239		353	-73,84	-74,18	-54,52	-61,72
354	1	2	30	1081		354	47,77	47,86	46,84	51,23
355	1	7	55	875		355	50,23	49,74	48,72	57,65
356	2	11	43	1077		356	77,12	75,72	59,93	47,82
357	1	52	26	670		357	-89,58	15,05	54,66	30,92
358	1	40	60	1124		358	62,24	60,56	66,00	29,40
359	1	1	24	661		359	78,76	78,95	78,71	52,26
360	3	4	25	604		360	60,28	65,09	60,19	61,49
361	2	7	28	1209		361	-22,51	-23,63	-25,76	-18,36
362	5	5	42	877		362	-6,39	-3,46	-4,67	-4,21
363	3	3	27	800		363	-0,50	-4,17	-4,66	-2,06
364	3	1	30	744		364	-18,96	-20,95	-18,69	-15,53
365	2	2	33	775		365	-17,32	-18,50	-17,21	-17,04
366	1	5	48	828		366	27,29	27,13	24,40	22,84
367	3	2	40	841		367	-9,38	-11,87	-10,55	-3,75
368	2	3	35	853		368	-26,49	-27,53	-23,57	-16,67
369	7	11	124	1572		369	17,83	74,41	18,47	-2,18
370	2	3	38	986		370	16,20	16,37	17,59	14,55
371	2	5	29	765		371	-27,85	-31,67	-30,32	-28,05
372	3	8	40	985		372	-27,37	-24,83	-28,90	-26,12
373	2	16	28	751		373	-18,53	-23,68	-17,14	-17,55
374	1	6	19	907		374	-32,39	-31,72	-29,81	-26,71
375	3	9	38	784		375	-19,74	-19,70	-14,26	-16,78
376	3	3	53	1905		376	-5,94	1,71	-4,94	32,23
377	3	14	33	737		377	-8,08	-4,75	-9,88	-11,28
378	2	10	44	813		378	37,92	38,19	36,59	39,26
379	3	61	78	1506		379	27,42	28,49	27,35	21,90
380	2	9	32	822		380	75,38	73,53	57,39	41,90
381	1	5	48	685		381	14,44	13,95	6,29	8,59
382	1	15	39	718		382	5,60	7,01	7,35	6,94
383	6	5	92	1191		383	-1,99	-15,12	-10,37	-6,40
384	2	3	38	732		384	-0,94	-0,87	-3,52	-1,71
385	2	8	44	789		385	5,17	7,13	12,07	8,94
386	3	7	27	753		386	78,44	77,54	54,64	41,78
387	5	2	64	6233		387	82,35	-87,94	-22,63	7,23
388	1	3	32	879		388	-88,51	-88,39	61,95	-8,16
389	5	12	51	1030		389	-4,26	-70,12	-34,69	-14,27
390	1	7	29	730		390	88,90	88,89	56,34	1,50
391	1	6	28	572		391	85,04	85,69	80,93	12,50
392	7	3	55	980		392	57,75	62,86	59,83	38,34
393	9	5	71	1115		393	71,03	59,60	56,66	35,40

Table 33. Crack based semantic segmentation metric results (per sample) (cont’d)

Number of Objects						**Mean Orientation (Degrees)**				
Test 1 - Building - Concrete						**Test 1 - Building - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
394	2	26	67	1077		394	80,62	80,72	15,97	46,83
395	3	3	58	896		395	11,68	-19,71	6,89	8,01
396	3	5	67	989		396	72,78	78,83	46,51	15,08
397	3	3	46	953		397	30,50	82,68	39,36	10,26
398	1	8	32	819		398	17,89	7,38	12,08	4,91
399	1	7	30	929		399	50,66	50,21	50,55	49,15
400	4	15	48	1159		400	43,11	39,03	45,56	46,77
401	2	12	49	1232		401	64,13	63,48	52,40	60,06
402	1	11	35	976		402	57,34	55,80	55,20	53,27
403	1	3	21	703		403	17,19	16,79	16,74	24,24
404	1	35	51	1222		404	46,19	45,78	43,58	48,50
405	4	8	51	1096		405	45,20	44,91	43,44	46,77
406	1	2	37	1224		406	46,65	46,49	46,72	52,25
407	1	66	47	855		407	73,63	77,90	68,50	52,84
408	3	9	57	1147		408	50,09	50,22	47,02	51,13
409	3	6	64	1624		409	0,85	6,60	-2,73	1,98
410	2	8	43	1140		410	-24,11	-24,70	-22,35	-23,14
411	3	6	49	1176		411	-51,04	-52,45	-46,74	-44,34
412	2	12	81	1317		412	-62,10	-53,37	-56,33	-39,03
413	1	7	38	989		413	-80,57	-80,38	28,98	-17,87
414	1	3	31	814		414	-58,28	-58,56	-60,85	-47,64
415	2	8	25	994		415	88,35	-84,55	-49,01	-11,47
416	3	14	32	1422		416	80,00	67,66	80,33	20,67
417	5	7	108	1414		417	74,14	68,46	69,87	18,86
418	9	60	121	1859		418	-79,51	-74,57	-18,91	-19,96
419	1	9	60	1493		419	66,18	61,45	67,98	53,30
420	2	15	44	1424		420	79,58	73,13	74,12	17,75
421	6	66	36	897		421	-77,29	-82,39	-82,12	20,40
422	3	11	43	1141		422	-85,12	-88,50	-88,05	17,27
423	1	16	35	936		423	74,57	57,76	56,56	46,01
424	6	2	23	835		424	21,39	24,89	25,19	33,96
425	4	8	55	882		425	-48,68	-49,98	-53,19	-47,45
426	1	8	26	835		426	-24,96	-24,18	-22,14	-23,93
427	2	38	37	1434		427	-8,83	-9,57	-0,89	0,03
428	2	14	24	1053		428	-3,53	-5,27	-7,86	-10,95
429	1	8	36	1032		429	-4,61	-4,63	-13,32	-8,48
430	3	2	36	838		430	0,30	-2,21	8,00	-2,89
431	4	3	50	874		431	16,43	10,05	8,78	10,29
432	1	14	55	1041		432	13,18	12,68	20,21	19,36
433	2	2	82	1294		433	65,37	77,82	34,80	28,32
434	2	9	35	1005		434	-4,67	-4,88	4,98	8,42
435	7	62	109	1892		435	9,70	14,86	12,16	10,40
436	1	60	39	1309		436	-17,35	-14,52	-12,46	-14,93
437	4	21	58	1427		437	-16,09	-15,87	-16,25	-24,56
438	4	44	59	1729		438	-20,16	-21,92	-20,75	-19,90
439	1	28	41	1499		439	-19,53	-20,98	-22,06	-19,16
440	6	21	78	1604		440	-51,44	-52,80	-52,55	-49,99
441	4	51	71	1257		441	-66,22	-60,72	-68,41	-47,79
442	6	11	41	1075		442	-72,14	-69,02	-36,51	-8,88
443	3	19	42	1049		443	-71,12	-65,47	-67,50	-25,86
444	3	4	66	832		444	74,29	78,33	43,15	28,28
445	7	10	110	1300		445	-28,65	78,05	36,37	15,05
446	3	14	41	881		446	-74,80	-77,40	-75,21	-43,31
447	1	12	47	773		447	-84,08	-80,87	-35,57	-4,88
448	1	15	31	510		448	-82,80	-82,36	-61,24	-7,16
449	3	8	99	3011		449	-24,77	-88,04	-15,44	16,37
450	3	3	9	641		450	-75,57	-87,50	-58,70	38,54
451	1	7	22	777		451	-82,44	-82,68	19,54	5,58
452	1	8	23	878		452	-82,81	-82,75	-71,50	-11,10

Table 33. Crack based semantic segmentation metric results (per sample) (cont’d)

Number of Objects						**Mean Orientation (Degrees)**				
Test 1 - Building - Concrete						**Test 1 - Building - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
453	3	35	40	872		453	78,08	74,18	-27,50	34,55
454	6	9	49	923		454	46,64	77,09	56,17	19,91
455	4	38	65	1278		455	64,89	-80,79	29,56	-11,35
456	7	11	76	1204		456	53,87	62,38	46,86	39,42
457	5	8	67	1102		457	-20,59	80,40	23,14	29,70
458	6	28	69	1591		458	76,02	73,64	34,86	34,33
Test 2 - Pavement - Concrete						**Test 2 - Pavement - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
1	7	89	2811	68764		1	-0,02	-22,06	2,58	-18,13
2	14	21	906	38776		2	-82,97	-82,10	-27,26	-41,86
3	13	20	1130	35694		3	-82,26	-78,32	-55,79	-44,97
4	7	28	1245	29537		4	-89,48	-77,05	-59,52	-49,29
5	6	131	737	22984		5	83,41	5,82	53,46	61,38
6	16	44	1469	53109		6	-79,84	-69,28	-53,41	-41,88
7	20	17	1509	33522		7	89,12	70,70	-64,63	58,08
8	6	99	1741	37914		8	84,89	69,57	24,04	55,18
9	14	35	1349	64007		9	84,00	71,96	-10,92	50,87
10	19	47	1678	43158		10	-72,58	-4,41	-17,16	-33,07
11	4	89	4054	78870		11	33,46	18,86	-10,93	4,16
12	8	134	3490	65867		12	-61,60	-45,45	-9,17	-18,11
13	10	46	3115	75428		13	36,89	30,53	-0,87	6,45
14	8	176	3766	89222		14	4,68	-28,92	4,42	-7,10
15	12	37	1785	55651		15	-89,93	21,10	-10,70	-47,49
16	7	51	1228	46590		16	82,83	-12,59	-11,72	60,93
Test 3 - Building - Concrete						**Test 3 - Building - Concrete**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
1	3	7	154	2290		1	-21,41	-60,83	14,22	-45,68
2	3	31	268	3679		2	-45,73	-73,73	-10,58	-30,37
3	10	6	189	3108		3	43,91	43,81	34,79	35,63
4	4	11	246	6147		4	-8,65	7,90	2,91	1,27
5	8	4	10	26386		5	22,14	24,35	21,34	14,20
6	4	5	38	2509		6	-77,71	-60,45	-68,37	-54,81
7	8	25	158	4472		7	-77,55	-59,29	-55,03	-32,53
8	27	43	999	97998		8	78,14	68,72	46,54	52,55
9	4	156	1428	35960		9	63,67	42,42	39,48	13,56
Test 4 - Building - Brickwork						**Test 4 - Building Brickwork**				
Image Number	**Ground Truth**	**QuadP**	**FCN**	**SegNet**		**Image Number**	**Ground Truth**	**QuadP**	**FCN**	**SegNet**
1	15	73	787	14846		1	9,28	15,74	45,39	83,75
2	1	23	698	16156		2	-86,95	-69,95	-41,46	24,39
3	8	32	743	22193		3	-80,05	-55,54	-46,16	16,45
4	7	28	592	35660		4	-0,98	-22,35	-20,24	-84,95
5	4	21	454	40490		5	-81,05	10,27	0,34	-86,34
6	31	39	804	43550		6	-54,22	-53,40	-0,95	83,55
7	9	32	669	51982		7	-86,36	-32,31	7,47	70,27
8	2	18	638	42375		8	-85,78	-69,09	-0,92	81,53

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name	Özgenel, Çağlar Fırat
Tel	05364868713
web	www.firatozgenel.com | dds.archweb.metu.edu.tr
e-mail	firatozgenel@gmail.com | fozgenel@metu.edu.tr
Address	Üniversiteler Mah. Dumlupınar Bulvarı No:1 METU Department of Architecture, Çankaya, Ankara, TURKEY

EDUCATION

2012 - 2018	Ph.D. Middle East Technical University (METU) / Building Science
2009 - 2012	M.Sc. Middle East Technical University (METU) / Building Science
2004 - 2009	B.Sc. Middle East Technical University (METU) / Physics
2001- 2004	TED Ankara College (High School) / Ankara

WORK EXPERIENCE

02.2013 – …	Middle East Technical University / Specialist
10.2012 – 02.2013	METU Faculty of Architecture Student Assistant (ARCH470, ARCH475 and ARCD 501)
10.2012 – 03.2013	METU Faculty of Architecture Digital Design Studio - Rhinoceros and Grasshopper Tutor
10.2009 – 02.2010	METU Faculty of Architecture Digital Design Studio Autodesk Maya Tutor

PUBLICATIONS

International Publications

2015 – Gönenç Sorguç A., Özgenel Ç.F., 'Proposal of a New Tool for 3D Pattern Generation' Nexus Network Journal, 17:2, p.655-670, 2015.

2013 – Özgenel, Ç.F., Gönenç Sorguç, A., "New Room Acoustics Tool for Architects: RAT (Room Acoustics Tool)", LAP Lambert Academic Publishing, 2013.

2011 – Özgenel Ç.F., Gönenç Sorguç, A., 'Sayısal Ortamda Tasarımın Deneyimlenmesi için Arayüzlerin Geliştirilmesi: Bir Ön Tasarım Parametresi Olarak Ses' (Developing an Interface for Experiencing Design in Computational Medium: Sound as a Pre-Design Parameter), METU Journal of Architecture, 28:2, p.248-253, 2011(2).

National and International Conference and Symposium Presentations

2018 – Özgenel, Ç.F., Gönenç Sorguç, A. "Performance Comparison of Pretrained Convolutional Neural Networks on Crack Detection in Buildings", ISARC 2018, Berlin (accepted)

2018 – Gönenç Sorguç, A., Kruşa Yemişcioğlu, M., Özgenel, Ç.F.," Multiverse of a Form: Snowflake to Shelter", eCAADe 2018, Lodz. (accepted)

2018 –Kruşa Yemişcioğlu, M., Gönenç Sorguç, A., Özgenel, Ç.F.,"Crystal Formation and Symmetry in the Search of Patterns in Architecture", eCAADe 2018, Lodz. (accepted)

2017 – Gönenç Sorguç, A., Kruşa Yemişcioğlu, M., Özgenel, Ç.F., Katipoğlu, M. O., Rasulzade, R. "The Role of VR as a New Game Changer in Computational Design Education", eCAADe 2017, Rome.

2017 – Gönenç Sorguç, A., Özgenel, Ç.F., Küçüksubaşı, F., Kruşa Yemişcioğlu, M., Ülgen S. "Mimarlık Eğitiminde Tepkimeli Kinetik Sistem Taklaşımı"(Responsive Kinetic System Approach in Architectural Education), XI. MSTAS, Ankara.

2016 – Özgenel, Ç.F. "Otonom Çatlak Tespitinde Kullanılan Görüntü Işleme Yöntemleri Ve Teknolojinin Potansiyelleri," (Potentials of Image Processing Methods and Technology Used in Autonomous Crack Detection), 3. Yapı Kongresi (3rd Building Conferene), Ankara.

2016 – Gönenç Sorguç,A., Özgenel,Ç.F., Kruşa,M.,Küçüksubaşı,F., Ülgen, S. “Biçim Arayışında Sayısal İmalat/Üretim Teknolojilerin Dönüştürücü Gücü” (Transformative Power of Digital Manufacturing/Production Technologies in Form Finding), X. MSTAS, İstanbul.

2015 – Kilis, S., Alkış, Y., Kadirhan, Z., Özgenel, Ç.F., Çetinkaya, H.H., Tokel, S.T. “Development of a Virtual Learning Environment: Hittites Empire”, EIED 2015, Paris.

2014 – Gönenç Sorguç,A., Özgenel,Ç.F. “Proposal of a New Tool for 3D Pattern Generation”, Nexus 2014: Relationships between Architecture & Mathematics / Ankara.

2012 – Özgenel Ç. F., Gönenç Sorguç, A., “A New Method Of Curve Fitting For Calculating Reverberation Time From Impulse Responses With Insufficient Length”, Internoise 2012 New York City.

2010 – Özgenel Ç. F., Gönenç Sorguç, A., “Sayısal Ortamda, Tasarımın Deneyimlenebilmesi için Arayüzlerin Geliştirilmesi: Bir Ön-Tasarım Parametresi olarak Ses” (Developing an Interface for Experiencing Design in Computational Medium: Sound as a Pre-Design Parameter) *“,* IV. MSTAS, İstanbul.

Editorial

2017 - Gönenç Sorguç, Arzu; Kruşa Yemişcioğlu, Müge; Özgenel, Çağlar Fırat (eds.) (2017). MSTAS2017: XI. Mimarlıkta Sayısal Tasarım Ulusal Sempozyumu (XI. National Symposium on Digital Design in Architecture): İmkansız Mekanlar: Olanaksızın Olanağı (Impossible Spaces: Possibility of Impossible), Book, Editor, METU Faculty of Architecture Press

Refereeing

2014 – Referee for METU Journal of the Faculty of Architecture

ORGANIZED EVENTS

15 – 16.06.2017	MSTAS.2017: 11. Mimarlıkta Sayısal Tasarım Ulusal Sempozyumu (National Symposium of Computational Design in Architecture), Member of the Organizing Committee (mstas2017.metu.edu.tr)
22.10 – 20.11.2016	3. Istanbul Design Bienale Contributor

09 – 12.06.2014	Nexus2014: Relationship between Mathematics and Architecture. Conference organizing committee member, PhD-Day Coordinator.
10 – 13.05.2013	Aggregated Workshop / Workshop Assistant
06 – 09.12.2012	Parametric Design Workshop / Workshop Assistant

RESEARCH PROJECTS

01.2017 – …	Greening the Skills of Architecture Students via STEAM Education (ARCHISTEAM) EU Project Erasmus + KA2 2016-1-TR01-KA203-034962 Project Researcher / METU
05.2016 – 06.2016	Istanbul New Airport Steel Construction Consultant, Researcher
02.2015 – 12.2017	Design of Curriculum for Woodworking CNC Operators in Turkey EU Project Erasmus + KA2 2014-1-TR01-KA200-013304 Project Researcher / METU
01.2016 – 12.2016	Development of Navigation Algorithm for Autonomous/Semi-autonomous Robotic Systems to be Utilized in Building Inspection/Monitoring BAP-02-01-2016-004 Project Researcher / METU
02.2014 – 12.2016	Gişe Geçişleri İçin Aktif Gürültü Bariyeri (Active Noise Barrier for Toll Booths) SanTez - 0553.STZ.2013-2 Funded by Turkish Ministry of Science and Industry Project Researcher/ METU
07.2013 – 03.2014	METU Campus Vertical and Horizontal Noise Mapping BAP – 0811- DPT.2013K120500-3. Project Researcher / METU

06.2012 – 01.2013	Hacettepe University Conservatory Building Acoustical Consultancy/ Project Researcher
04.2012 – 08.2012	Türkiye Büyük Millet Meclisi (Turkish National Assembly of Turkey) Acoustical Consultancy Project Researcher
2012	Türk Telekom Çağrı Merkezi Gürültü Azaltımı (Türk Telekom Call Center Noise Reduction), 2012-02-01-1-0011
03.2012 - 08.2012	Yunus Emre Power Plant Noise Control Project Researcher

ASSISTED COURSES

ARCH 470 | Digital Design Studio (2009 - …)

ARCH 475 | Advanced Digital Design Studio (2009 - …)

ARCH 333 | Mathematics in Architecture (2013 - …)

ARCH 479 | Acoustical Design of Halls for Musical Performance (2013 - …)

BS 770 | Digital Fabrication Techniques in Architecture (2015 - …)

FOREIGN LANGUAGES

- English : Fluent
- German : Intermediate

ATTENDED WORKSHOPS AND CERTIFICATES

- Autodesk Revit Workshop at METU
- SoundPlan Noise Mapping Certificate
- A-2 Type Evaluation of Environmental Noise Certificate (Chamber of Physics Engineering of Turkey)

REFERENCES

Available upon request