Tiny ImageNet Challenge

Jiayu Wu
Stanford University
jiayuwu@stanford.edu

Qixiang Zhang
Stanford University
qixiang@stanford.edu

Guoxi Xu
Stanford University
guoxixu@stanford.edu

Abstract

We present image classification systems using Residual Network(ResNet), Inception-Resnet and Very Deep Convolutional Networks(VGGNet) architectures. We apply data augmentation, dropout and other regularization techniques to prevent over-fitting of our models. What's more, we present error analysis based on per-class accuracy. We also explore impact of initialization methods, weight decay and network depth on system performance. Moreover, visualization of intermediate outputs and convolutional filters are shown. Besides, we complete an extra object localization system base upon a combination of Recurrent Neural Network(RNN) and Long Short Term Memroy(LSTM) units. Our best classification model achieves a top-1 test error rate of 43.10% on the Tiny ImageNet dataset, and our best localization model can localize with high accuracy more than 1 objects, given training images with 1 object labeled.

1. Introduction

The ImageNet Large Scale Visual Recognition Challenge(ILSVRC) started in 2010 and has become the standard benchmark of image recognition. Tiny ImageNet Challenge is a similar challenge with a smaller dataset but less image classes. It contains 200 image classes, a training dataset of 100,000 images, a validation dataset of 10,000 images, and a test dataset of 10,000 images. All images are of size 64×64.

The goal of our project is to do as well as possible on the image classification problem in Tiny ImageNet Challenge. In order to overcome the problem of a small training dataset, we applied data augmentation methods to training images, hoping to artificially create variations that help our models generalize better. We built our models based on the idea of VGGNet [13], ResNet [6], and Inception-ResNet [17]. **All our image classification models were trained from scratch**. We tried a large number of different settings, including update rules, regularization methods, network depth, number of filters, strength of weight decay, etc. Our best model, a fine-tuned Inception-ResNet, achieves a top-1 error rate of **43.10%** on test dataset. Moreover, we implemented an **object localization network** based on a RNN with LSTM [7] cells, which achieves precise results.

In the Experiments and Evaluations section, We will present thorough analysis on the results, including per-class error analysis, intermediate output distribution, the impact of initialization, etc.

2. Related Work

Deep convolutional neural networks have enabled the field of image recognition to advance in an unprecedented pace over the past decade.

[10] introduces AlexNet, which has 60 million parameters and 650,000 neurons. The model consists of five convolutional layers, and some of them are followed by max-pooling layers. To fight overfitting, [10] proposes data augmentation methods and includes the technique of dropout[14] in the model. The model achieves a top-5 error rate of 18.9% in the ILSVRC-2010 contest. A technique called local response normalization is employed to help generalization, which is similar to the idea of batch normalization[8]. [13] evaluates a much deeper convolutional neural network with smaller filters of size 3×3. As the model goes deeper, the number of filters increases while the feature map size decreases by max-pooling. The best VGGNet model achieves a top-5 test error of 6.8% in ILSVRC-2014. [13] shows that a deeper CNN with small filters can achieve better results than AlexNet-like networks. However, as a CNN-based model goes deeper, we have the degradation problem. More specifically, a deeper CNN is supposed to have at least the same performance as a shallower CNN, but in practice, a shallower CNN converges to a lower error rate than a deeper CNN. To solve this problem, [6] proposes ResNet. Residual networks are designed to overcome the difficulty of training a deep convolutional neural network. Suppose the mapping we want to approximate is $\mathcal{H} : \mathcal{R}^n \to \mathcal{R}^m$. Instead of approximating $\mathcal{H}$ directly, we approximate a mapping $\mathcal{F}$ such that $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$. [6] shows empirically that the mapping $\mathcal{F}$ is easier to approximate through training. [6] re-

ports their best ResNet achieves a top-5 test error of 3.57% in ILSVRC-2015.

[17] combines inception structures and residual connections. Inception-Network is able to achieve better performance than traditional residual networks with less parameters. Inception module was introduced by [18]. The basic idea is that we apply different filters and also max-pooling to the same volume and then concatenate all the output so that each layer can choose the best methods during learning.

For object localization, [16] gives a model based on decoding an image into a set of people detections. The approach is related to OverFeat model[12], but has some improvement. The model relies on LSTM cells to generate variable length outputs, and the loss function encourages the model to make predictions in order of descending confidence.

3. Approach

3.1. Data Augmentation

Preventing overfitting is an essential problem to overcome, especially for Tiny ImageNet Challenge, because we only have 500 training images per class [1].

First, during training, each time when an image is fed to the model, a 56×56 crop randomly generated from the image will be used instead. During validation and testing, we use the center crop.

Besides, following [10], we then augment data by horizontal flipping, translation and rotation. We also used random contrast correction as an augmentation method. We set the scaling factor to be $random([0.9, 1.08])$ at every batch, and clip pixel values to the range $[0, 255]$ after correction process to guarantee valid augmented images. Another data augmentation method we used is random Gamma correction [11] for luminance adjustment. After some experiments, we used correction coefficient $\gamma = random([0.9, 1.08])$, which ensures both significant luminance change and recognizable augmented images. Fig. 1 gives concrete instances of above methods.

In order to speed up the training process, we apply these methods in a random fashion. When an image is fed to the model, every augmentation method is applied randomly to this image. In this way, the total number of training examples is the same but we managed to have our models see slightly different but highly recognizable images at each epoch.

3.2. Modified Residual Network

Fig. 2 shows the architecture of our modified ResNet. It consists of a series of convolutional layers of different number of filters, an average pooling layer, and finally a fully-connected affine scoring layer. A batch normalization layer is added between a convolutional layer and its ReLU activations. As [6] suggests, batch normalization layers can serve as a source of regularization. Since the random crop generated by our data augmentation methods is of size 56×56, although [6] uses 34 or more convolutional layers, we hypothesize that we only need less layers than the models in [6] since [6] uses 224×224 image inputs.

Unlike [6], we do not use a max pooling layer immediately after the first 7×7 convolutional layer because our input images already have a smaller size than 224×224 in [6]. Furthermore, our first 7×7 convolutional layer does not use a stride of 2 like [6]. Each building block for our modified ResNet includes $2n$ convolutional layers with same number of 3×3 filters, where we can adjust the depth of the network by varying n. In this project, we have tried $n = 1$ and $n = 2$. Notice that down-sampling is performed at the first layer of each building block by using a stride of 2.

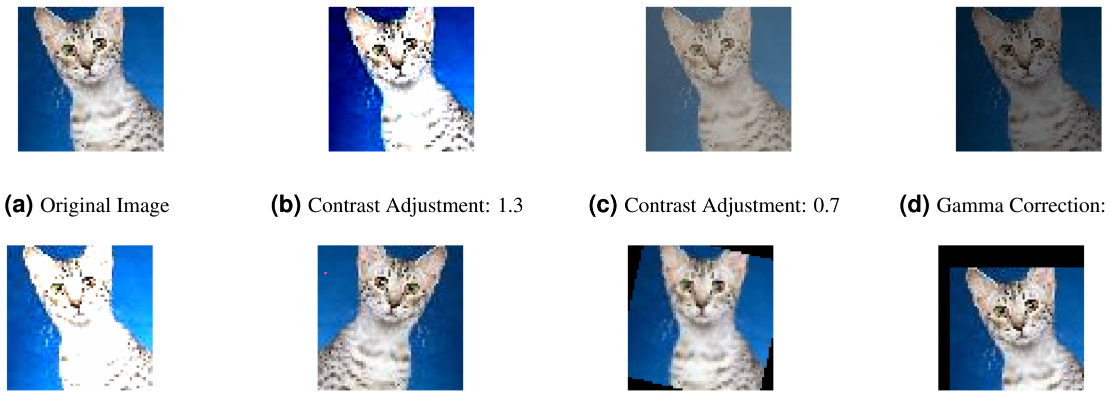
Combining two volumes with same dimensions is straightforward. For two volumes with different depths and different feature map sizes, [6] suggests two ways to create a shortcut, (A) identity mapping with zero-padding; and (B) a convolutional layer with 1×1 filters with a stride of 2. We use option (B) for our residual networks. In our project, we used option (B). Furthermore, when a 1×1 convolution is applied, batch normalization is applied to each of the two incoming volumes before they are merged into one volume. Finally, ReLU is applied on the merged volume.

3.3. Modified Inception-ResNet

We reorganized the state-of-art Inception-ResNet v2 model [17]. The architecture of our modified Inception-ResNet model is shown in Fig.3. The input images are first down-sampled in a stem module. The stem module has parallel convolutional blocks, whose outputs are later concatenated. Moreover, one convolutional layer of spatial filter size 1×7 and one of size 7×1 are combined to replace a 7×7 sized layer, which significantly reduces number of parameters while maintaining same receptive field. Then the data flows through n Inception-Resnet-A modules, which has the residual part being an inception module. A 1×1 convolutional layer is applied at the end of each inception module to keep the output depth same as the input's, thus enabling the final addition operation. Then after another down-sampling in a reduction module, the feature map flow passes through $2n$ Inception-Resnet-B modules and finally reaches a fully connected layer.

Fig.3 also shows several modifications we made to fit the Inception-Resnet model with the tiny ImageNet challenge. Firstly, we used 2 Inception-Resnet module types and 1 reduction module, while [17] uses 3 Inception-Resnet module types and 2 reduction modules. This simplifies the structure of model, and reduces number of parameters by

[1]We used $231n\_utils.py$ from assignment starter code for data loading



(a) Original Image **(b)** Contrast Adjustment: 1.3 **(c)** Contrast Adjustment: 0.7 **(d)** Gamma Correction: 0.93

(e) Gamma Correction: 1.06 **(f)** Horizontal Flipping **(g)** Random Rotation **(h)** Random Translation

Figure 1. Data Augmentation Methods, values in sub-figure captions are scaling factor γ

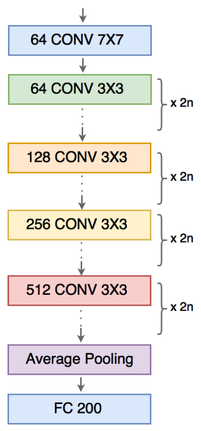


Figure 2. Residual Network Architecture

30%. Moreover, this prevents over-fitting training on tiny ImageNet. Another modification is that we used batch normalization after every convolutional layer, while [17] only applies a batch normalization layer at the end of each module. This modification significantly alleviate the problem of dying neurons. What's more, we used a keep probability of 0.5 in the final dropout layer, which is smaller than that in [17] (0.8). The reason is that with fewer data samples, tiny ImageNet challenge may more easily go into overfit problem, and thus it needs a smaller keep probability to introduce stronger regularization. On the other hand, a too large keep probability does not help in boosting performance, but makes the learning process much slower. Therefore after some tests, we chose 0.5 as the optimal keep probability.

If the depth of feature map exceeds 1000, the residual variants in Inception-ResNet modules becomes instable, which means neurons in the network die very early during the training process. As a result, the last convolutional layer before average pooling may outputs only zeros, reducing prediction accuracy. One effective solution to this problem is to scale down the residuals parts in by a factor before the addition. In this way, weight of the residual part is decreased by multiplying a small scaling factor, and thus the identity mapping part becomes dominant. As a result, the module becomes more residual than inceptional, thus stabilizing training precess. After running independent tests, we found the optimal scaling factor to be 0.1 for the tiny-ImageNet challenge.

3.4. VGGNet

VGGNet architecture [13] is shown in Fig.4. Compared to previous architectures which use large (7×7) convolutional filters, VGGNet uses smaller (3×3) filters but relatively deep networks (16-19 weight layers). In this way, the network has less parameters while maintaining same receptive field for each convolutional filter. Also, 1×1 convolutional filters are used to linearly transform the input. All down-sampling work is done in max pooling layers: spatial size of the feature map is halved in every zero-padded max pooling layer with a stride of 2. Finally, Softmax loss is applied to output of the FC-200 layer.

We implemented VGG-19 and VGG-16 networks with some modifications. We first reduced size of the intermediate fully connected layer from 4096 to 2048, as our relatively tiny dataset doesn't need large model capacity. Moreover, we removed the last max pooling layer in both networks so that the last down-sampling operation is not conducted. This is because our images have a spatial size of only 64×64, and the feature map needs to maintain enough spatial size before the average pooling layer. To avoid overfitting, we used L_2 regularization and apply a dropout layer to each fully connected layer. To avoid killing neurons in the negative half, we used Parametric Rectifier (PReLU)

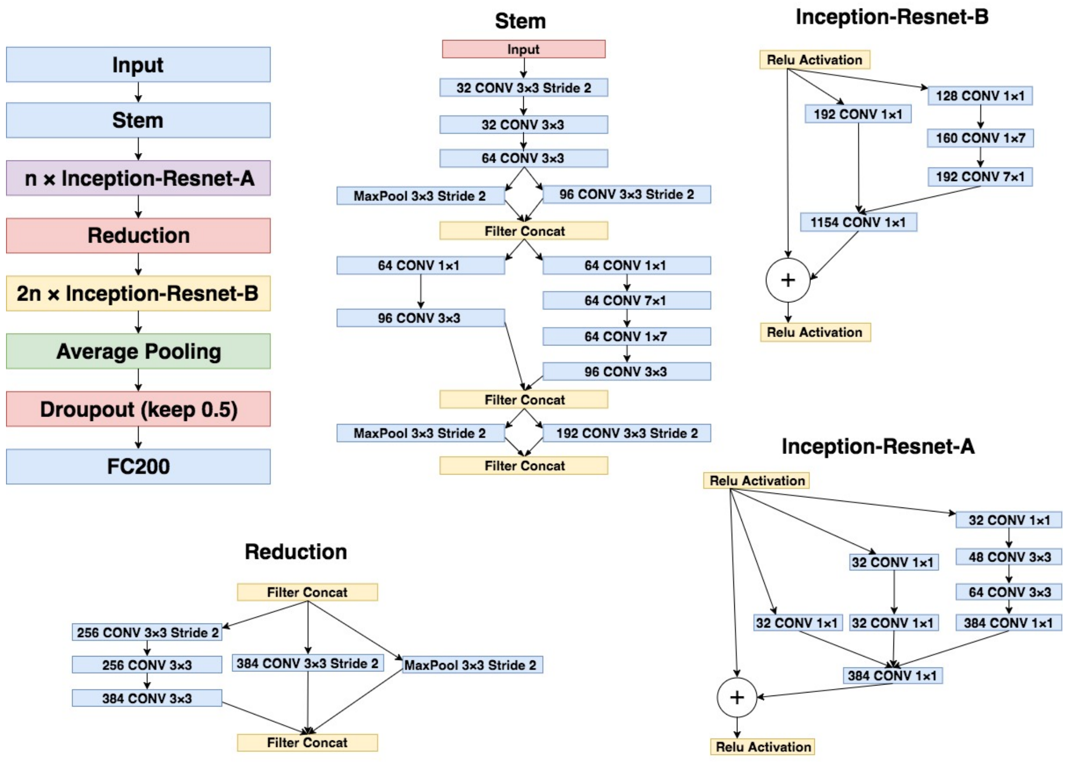


Figure 3. Modified Inception-Resnet Architecture

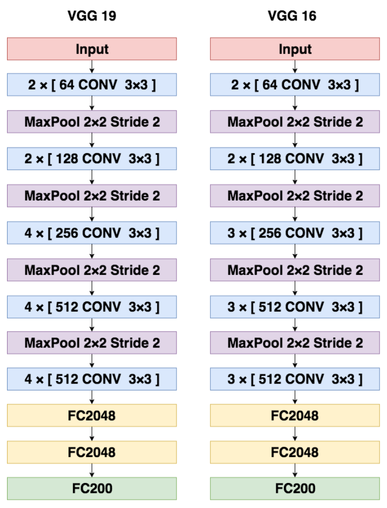


Figure 4. Modified VGGNet Architectures

[5]defined as $f(x) = max(\alpha x, x)$ with a trainable parameter α, and apply a batch normalization layer before each activation layer.

3.5. Object Localization Method

The localization network first encodes an image into a block of high level features using a convolutional architecture, then decodes that representation into a set of bounding boxes. At each step, LSTM gives a new bounding box and a confidence shows an undetected object is possibly in the box, until LSTM is unable to find another box with a confidence greater than the threshold.

In the training process we need to minimize the loss function

$$L(G, C, f) = \alpha \sum_{i=1}^{|G|} l_{pos}(\mathbf{b}_{pos}^i, \tilde{\mathbf{b}}_{pos}^{f(i)}) + \sum_{j=1}^{|C|} l_c(\tilde{\mathbf{b}}_c^j, y_j) \quad (1)$$

where an object bounding box $\mathbf{b} = \{\mathbf{b}_{pos}, b_c\}$, where $\mathbf{b}_{pos} = (b_x, b_y, b_w, b_h)$ gives position, weight and height of bounding box, and $b_c \in [0, 1]$ gives the confidence. l_{pos} gives the L_1 norm between ground truth position and candidate hypotheses, and l_c gives the cross entropy related to candidate's confidence. Only the boxes with confidence higher than a threshold value will be given out, and the box with confidence lower than a it is considered as a stop symbol. Every generated box can be accepted or rejected(see Fig.5).

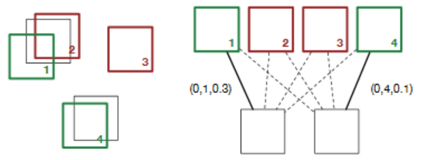


Figure 5. Illustration of the matching of ground-truth instances (black) to accepted (green) and rejected (red) candidates. Matching should respect both precedence (1 vs 2) and localization (4 vs 3).

3.6. Training Methodology

We trained our networks using TensorFlow [1] on a Tesla K80 GPU. For Inception ResNet models, we used RMSProp [19]:

$$M = decay \times M + (1 - decay) \times (\nabla W)^2 \quad (2)$$

$$\delta W = -\frac{lr \times \nabla W}{\sqrt{M} + \epsilon} \quad (3)$$

with $decay = 0.9$ and $\epsilon = 1.0$. RMSProp modulates the learning rate of each weight value based on the magnitudes of its gradients, which has a beneficial equalizing effect. However, the updates do not get monotonically smaller as training process moves forward, as it uses a moving average of the squared gradients instead of an accumulative squared gradient that monotonically grows larger. We used an initial learning rate $init\_lr = 0.05$, decayed exponentially using the equation:

$$lr = init\_lr \times 0.9^{\frac{t}{T}} \quad (4)$$

, where t denotes current step/iteration number, and T denotes total number of steps per epoch. This achieves a 0.9 learning rate decay after each epoch.

For ResNet models, we used SGD with Nesterov momentum [2]:

$$V_{new} = \mu \times V_{old} - lr \times \nabla W \quad (5)$$

$$\delta W = -\mu \times V_{old} + (1 + \mu) \times V \quad (6)$$

with $\mu = 0.9$. The essential idea of SGD with Nestrov momentum is that it computes the gradients ”in advance” at a point supposedly reached at next step, and then uses that gradients for update. This helps to reduce overshoots brought by vanilla SGD-Momentum algorithm, and proves to work better in practice. The initial learning rate is set to 0.1, and we decayed the learning rate by a factor of 0.1 when validation error plateaus.

For VGGNet models, we used Adam optimization[9] as the update rule. We also used equation 4 as learning rate decay rule with $init\_lr = 0.001$.

Model	Top-1 Val	Top-5 Val	Top-1 Test	# Params
Inception-Resnet	37.56%	18.96 %	43.10%	8.3M
ResNet	43.50%	20.30%	46.90%	11.28M
VGG-19	45.31%	23.75%	50.22%	40.2M
VGG-16	47.22%	25.20%	51.93%	36.7M

Table 1. Summary of Model Error Rates

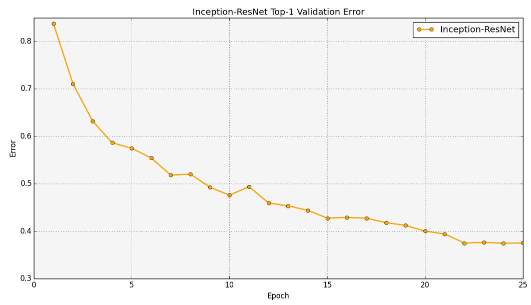


Figure 6. Modified Inception-Resnet Top-1 Validation Error Rate

4. Experiments and Evaluations

4.1. Experimental Results

We trained all our networks from scratch, and Table 1 shows final error rates of our models. Our best model is an Inception-Resnet network achieving **37.56 %** top-1 validation error, **18.96%** top-5 validation error, and **43.10%** top-1 test error. Fig.6 shows the top-1 validation error rate of this model. To summarize, Inception-Resnet achieves highest accuracy and uses least number of parameters. Resnet has almost comparable accuracy and parameter number to Inception-Resnet. Although VGGNet networks use less weight layers, they achieve lower accuracy and have more parameters, and there are two reasons for that. First, VGGNet uses fully connected layers with many hidden states. Secondly, VGGNet uses a large number of medium-sized filters (512×3×3) in each convolutional layer, while Inception-Resnet uses either a large number of small filters (1154×1×1) or a small amount of medium-sized filters (64×3×3) to reduce model size.

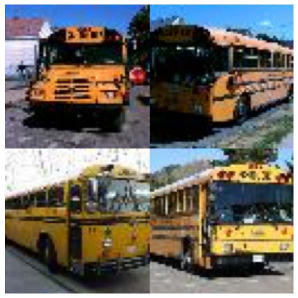
4.2. Error Analysis

Using our best-performing Inception-ResNet model, we calculated per-class error rates for all 200 classes on the validation dataset. We then summarized top-5 accurate/inaccurate classes from these error rates with results shown in Table 2. We also collected images of the best-performing class (school bus) and the worst-performing class (nail) with respect to validation accuracy in Fig 7.

When we went into images themselves, we had some interesting observations. For a class with high validation accuracy, images of this class tend to share same features such as color, texture, outline, etc. Fig.7a presents 4 images of

Class Name	Val Accuracy	Class Name	Val Accuracy
school bus	0.94	nail	0.14
trolleybus	0.9	syringe	0.2
bullet train	0.9	barrel	0.24
sulphur butterfly	0.9	plunger	0.26
maypole	0.88	wooden spoon	0.26

Table 2. Top-5 Accurate/Inaccurate Classes


(a) School Bus Example Images

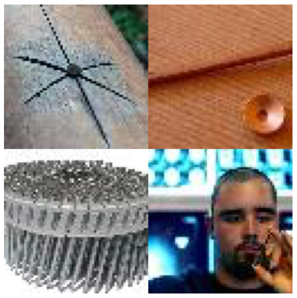
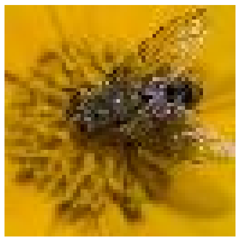
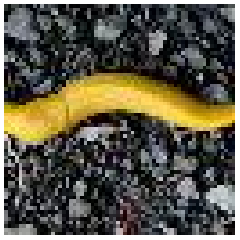

(b) Nail Example Images

Figure 7. Images of the Most Accurate/Inaccurate Class


(a) Fly Recognized as Bee

(b) Puma Recognized as Lion


(c) Slug Recognized as Sea Slug

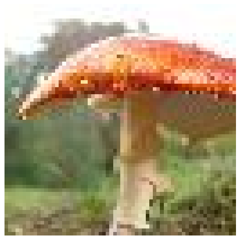

(d) Mushroom Recognized as Umbrella

Figure 8. Mistakenly Recognized Images

the school bus class. The school buses in these images all have cubic shape and are all decorated yellow. Moreover, these school buses are highly recognizable by human being. On the contrary, pictures of inaccurate classes have diverse features and appear confusing to human being. In Fig.7b, 4 images of the nail class seem to share few common features. Moreover, these images are so confusing that even a human being may well classify them into a wrong class. For instance, the image on top-left looks similar to a spider, and the image on bottom-right is dominated by a man's face rather than the nail itself.

We also analyzed some mistakenly recognized pictures (Fig.8) and found two origins of error. Firstly, mistakes may appear when the ground-truth label is very similar to another label. For example, in 8b the ground-true label puma looks like the label lion a lot, and in 8d the label mushroom has the same outline as the label umbrella. Another reason is that although true label is not quite similar to the mistaken label, background of a specific image may be closely associated with that mistaken label. In 8a, a fly lying on a flower is recognized a bee. This is because most training images of the bee label have flowers as background, and thus images with flowers have higher probabilities to be labeled as bees.

4.3. Output Distribution Visualization

During the training process of our best Inception-ResNet model, we used TensorBoard to record the runtime output distribution of some important layers (see Fig.9 and Fig.10). The X-axis denotes current step number, and the Y-axis denotes scalar values of an output matrix. A distribution figure records changing curves of some special values of the output distribution: $maximum$, $\mu + 3\sigma$, $\mu + 2\sigma$, ... $\mu - 3\sigma$, and $minimum$. It also records regions between these values. For instance, the most light-colored region on the top corresponds to values between $\mu + 3\sigma$ and maximum value. Some analysis were then made using these records.

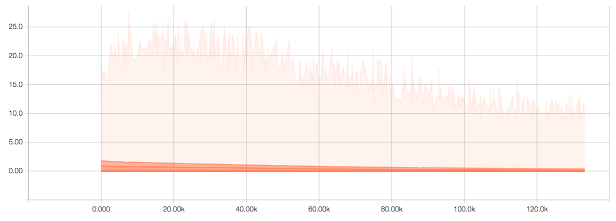


Figure 9. Output Distribution of Inception-ResNet-B

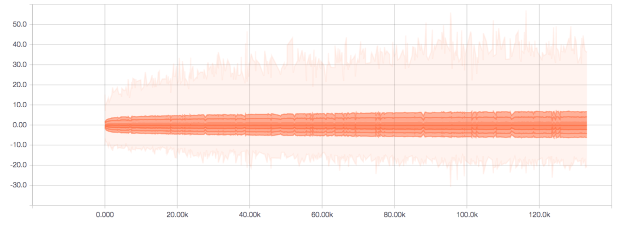


Figure 10. Output Distribution of FC-200

Fig.9 shows output of the last Inception-ResNet-B module, which lies right before the average pooling layer. Since a ReLu activation is applied, all output values are non-negative. The figure also shows that as the training process goes on, the $\mu+3\sigma$ point goes down towards 0, and the maximum value decreases gradually. This suggests that most scalar values in the output matrix become zero, and the reason is that most parameters in convolutional filters become zero. This observation verifies the sparsity assumption in statistical learning, which states that only a small portion

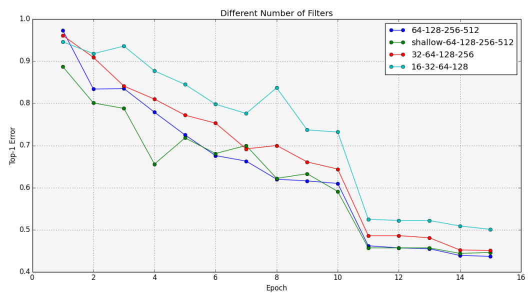


Figure 11. ResNet: Compare # Filters and # Layers

of parameters in the model contribute to the final outputs. This assumption serves as the prerequisite of many important machine learning theorems and attributes.

Fig.10 shows distribution of the final raw score matrix produced by the FC-200 layer. As the training process goes forward, the standard deviation of distribution becomes larger, suggesting that the output score matrix becomes more distributed. As we are using the softmax loss, the ideal case is that the right label has a very high score, while all other labels have low scores. Instead, the worst case is a random guess in which all labels have the same score. Therefore, this distribution curve with a standard deviation growing higher suggests that our network is learning well (given that the network is not yet over-fitted).

4.4. Do We Need to Go Deeper?

Compared to the ImageNet challenge, our tiny ImageNet challenge has a much smaller dataset. Also, our original pictures with 64×64 pixels are simpler than the 299×299 pictures in ImageNet challenge. According to statistical learning theory[20], when number of parameters in a predictor is much larger than number of data samples, the problem may become too flexible such that there are much more optimal solutions, which makes the model harder to train. Thus for our problem, while going deeper with convolutional layers may make the model more powerful, it increases the chance of overfitting and it requires more computational resources and training time. Therefore, we conducted some experiments to explore impacts of going deeper.

Fig. 12 shows that a 10-layer ResNet ([64, 128, 256, 512] × 2) performs very similarly to a 18-layer ResNet ([64, 128, 256, 512] × 4) with the same number of filters for each building block. Furthermore, the 10-layer ResNet is easier to train at the beginning since the validation error drops faster than that of the 18-layer ResNet. On the other hand, Fig. 12 also shows that a 18-layer ResNet with more filters at each building block ([64, 128, 256, 512]×4) does perform significantly better than a 18-layer ResNet with less filters at each building block ([16, 32, 64, 128]×4).

Therefore, the benefits of going deeper in this project may be limited. One reason is that we have a small training dataset, and deeper networks usually need more data in order to be better at generalization than shallower networks.

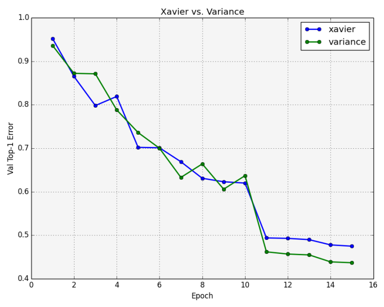


Figure 12. Initialization: Xavier vs. Variance

4.5. Xavier Versus Variance Initialization

We investigated two popular initialization strategies. Since training a deep convolutional neural network is usually a non-convex optimization problem, the results are sensitive to starting points. Therefore, different weight initialization strategies may lead to different results.

[3] proposes an initialization strategy that takes both input dimension and output dimension into consideration, which is called Xavier Initialization. Xavier initialization is defined as: $W_{ij} \sim U\Big(-\frac{1}{\sqrt{n+m}}, \frac{1}{\sqrt{n+m}}\Big)$, where n is the input dimension for the current layer, and m is the output dimension for the current layer.

While xavier initialization has been shown to be very useful in practice, [4] proposes a new initialization strategy that is theoretically more sound and performs better in practice when used on deep networks. This variance-scaling initialization is defined as $W_{ij} \sim N\Big(0, 2/n\Big)$, where n is the input dimension of the current layer.

From the definitions, we can see two differences: first, xavier initialization uses a uniform distribution while variance-scaling initialization relies on a normal distribution; second, xavier initialization considers both input and output dimensions while the variance-scaling intialization only considers the input dimension, which results in that weights can have a larger varying range using variance-scaling initialization.

Here we compare the two strategies on a 18-layer ResNet with filters ([64, 128, 256, 512]×4). Fig. 12 shows that until 10 epochs, namely before the learning rate is decayed, the results of both strategies are similar. However, after the learning rate is decayed, the network initialized with variance scaling is able to converge to a lower level, compared to the network initialized with xavier initialization. The results suggest that the trained weights are more likely to

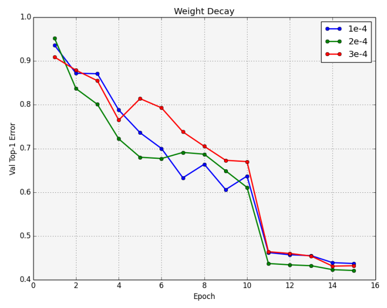


Figure 13. 18-Layer ResNet with Different Weight Decay

Weight Decay	Validation Top-1 Error
1×10^{-4}	43.7%
2×10^{-4}	42.1%
3×10^{-4}	43.2%

Table 3. 18-Layer ResNet with Different Weight Decay

follow a normal distribution rather than a uniform distribution. Furthermore, the discrepancy after the learning rate is decayed suggests that different initialization strategies may lead to different narrow valleys on the loss surface.

4.6. The Impact of Weight Decay

Due to the fact that we only have 500 training images per class, another challenge of this project is how to properly regulate our networks. Apart from applying data augmentation methods, we are also interested in how important weight decay is. We trained a 18-layer ResNet ([64, 128, 256, 512]×4) with a weight decay of 1×10^{-4}, 2×10^{-4}, and 3×10^{-4}.

Fig. 13 and Table 3 shows that the model with a weight decay of 2×10^{-4} achieves a validation top-1 error that is 1.6% or 1.1% lower than that of the model with 1×10^{-4} or 3×10^{-4}.

From the results, we can see that weight decay does have an impact on the performance, however, the impact may be limited to at most around 1-2% boost on error rate.

4.7. Filter Visualization

Here we visualize half of the filters of the fist convolutional layer of our modified 18-layer ResNet in Fig.14.We see that most filter visualizations have different shape outlines at the center. Furthermore, these filter visualizations are dominated by different colors. Therefore, the visualization plot indicates that each filter is generalizing a unique type of features.

4.8. Object Localization

Based on the object detection system TensorBox[15], we used our dataset to train the image localization model. Noticing that the original code only detects people's faces, we only fed one category of training images once, and did some modification on the code in order to fit our data. Here we use gold fish as an example. In the training dataset there are 500 images labeled gold fish, and we pick 80% of them as training data, and the other 20% as test data. We

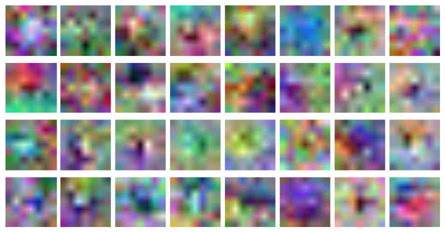


Figure 14. Visualization of First Conv Layer

evaluated the performance on the test data. From Fig.15, we can see for single object localization the performance is quite good. Also, even if the training images only contain one box each, in the test images the network can output more than one boxes. However, for the detection of many objects, the performance is not so good. We need to point out that the small size(64×64) of images limits the performance of detection, and makes it harder to localize more than one objects.

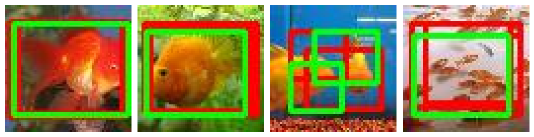


Figure 15. Examples of Object Localization Results

5. Conclusion and Future Work

We tailored the state-of-art deep CNN models to the 200-class image classification problem presented in Tiny ImageNet Challenge. We implemented and modified VGGNet, ResNet, and Inception-ResNet. We trained all our models from scratch and the top-performing model achieves a top-1 test error of **43.1%**.

We also conducted thorough analysis on the results. We found our best model is able to recognize images with clear features, while failing to recognize others with multiple objects or ambiguous features. We also investigated different initialization strategies, and we found variance scaling initialization works better than xavier initialization for our ResNet-based models. Furthermore, we discovered that a proper weight decay strength can give a small boost to the performance. We also learned that babysitting the training process is imperative. Decaying learning rate is very effective on reducing error rates.

For future work, we expect to have a better performance if we employ model ensemble techniques, for example, we let multiple model snapshots during the training process vote and then summarize their opinions to a prediction. Furthermore, it is also interesting to use pre-trained models on full ImageNet Challenge training dataset, which will reveal whether transfer learning works on Tiny ImageNet Challenge. Finally, if we can have some volunteers and test them on the 200-class image recognition task, it may be interesting to compare the performance of humans and deep CNNs.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[3] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

[4] X. Glorot and Y. Bengio. Delving deep into rectifiers:surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[6] He,Zhang,Ren,Sun. Deep residual learning for image recognition. 2016. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.

[11] P. Schneider and D. H. Eberly. *Geometric tools for computer graphics*. Morgan Kaufmann, 2002.

[12] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[15] R. Stewart. Tensorbox. `https://github.com/TensorBox/TensorBox`.

[16] R. Stewart, M. Andriluka, and A. Y. Ng. End-to-end people detection in crowded scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2325–2333, 2016.

[17] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference*, 2015.

[19] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

[20] V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.