FeatureNET: Diversity-Driven Generation of Deep Learning Models

Salah Ghamizi, Maxime Cordy, Mike Papadakis, Yves Le Traon
first.last@uni.lu
SnT, University of Luxembourg

ABSTRACT

We present FeatureNET, an open-source Neural Architecture Search (NAS) tool[1] that generates diverse sets of Deep Learning (DL) models. FeatureNET relies on a meta-model of deep neural networks, consisting of generic configurable entities. Then, it uses tools developed in the context of software product lines to generate diverse (maximize the differences between the generated) DL models. The models are translated to Keras and can be integrated into typical machine learning pipelines. FeatureNET allows researchers to generate seamlessly a large variety of models. Thereby, it helps choosing appropriate DL models and performing experiments with diverse models (mitigating potential threats to validity). As a NAS method, FeatureNET successfully generates models performing equally well with handcrafted models.

CCS CONCEPTS

• **Computing methodologies → Neural networks**.

KEYWORDS

Configuration search, NAS, Neural Architecture Search, AutoML

ACM Reference Format:
Salah Ghamizi, Maxime Cordy, Mike Papadakis, Yves Le Traon. 2020. FeatureNET: Diversity-Driven Generation of Deep Learning Models. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3377812.3382153

1 INTRODUCTION

Deep Learning (DL) systems can solve complex tasks in an increasing variety of domains, including safety-critical areas like self-driving cars and computer-aided health systems. They rely on a model – a neural network – which goes through a computationally-expensive training, based on known data, to learn how to perform the aimed task.

To facilitate the deployment of DL systems, pre-trained models have been made available in order to be used for solving the different tasks they were trained for, with a complementary training. This transferability property, however, does not universally hold, as shown by recent empirical studies [9]. Different models are often needed for different tasks.

Software diversity is a known approach for building robust, secure and reliable software [1]. The key idea is that systems with diverse functions and elements are robust to many kinds of unanticipated events, failures and inputs. In the context of machine learning, diversity may also help diminishing security and overfitting threats.

Moreover, the proliferation of DL systems make essential the study of additional requirements, such as correctness, privacy, robustness and interpretability. As a result, researchers design a growing number of Quality Assurance (QA) techniques to assess and improve DL systems wrt such requirements. However, a common pitfall is that these techniques are often evaluated on a small and similar set of DL models [11], often retrieved in a pre-trained form. Therefore, empirical results may not generalise to different models, especially to those deployed in the field.

To reduce this validity threat, it is essential to provide researchers with a wide spectrum of models. We fill this gap and present FeatureNET, a tool to generate, train and deploy large and diverse sets of feed-forward neural networks.

FeatureNET implements a Neural Architecture Search (NAS) method which involves a much larger space of DL models than established approaches. It relies on PLEDGE – a search-based algorithm initially developed for software product line testing [4] – that maximises the diversity of the generated models. Hence, the generated set contains models with diverse structure, attributes and performance indicators (e.g. accuracy, robustness, etc). This is in contrast to state-of-the-art NAS methods that target the best performing models.

Researchers can use FeatureNET to generate diverse models seamlessly, without requiring in-depth knowledge and experience in DL system engineering. Our tool can support multiple use cases, can be integrated with existing datasets and is open for extension.

Overall, FeatureNET provides the following functionalities:

(1) **Model search and generation:** FeatureNET implements a meta-model for feed-forward neural network architectures based on two levels of configurable entities (named Blocks and Cells), originally presented in [3]. The tool samples configurations of those entities by maximizing their diversity.
(2) **Model compilation and training:** FeatureNET parses the sampled configurations, translates them into Keras models [2] and launches the training on the chosen dataset and for the specified number of epochs.
(3) **Model assessment:** FeatureNET evaluates every generated model based on predefined metrics. It also features an API through which researchers can plug their experiments and

[1] A video of the tool is available at: youtu.be/63nRH9SryDM

[2] https://www.keras.io/

ICSE '20 Companion, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7122-3/20/05.
https://doi.org/10.1145/3377812.3382153

tailored evaluation metrics, and run these experiments on each of the trained models.

The approach underlying FeatureNET was originally presented in [3]. We also reported evaluation results based on an early prototype of the tool. Since then, we have enhanced the functionalities of FeatureNET and developed user interfaces to allow its broad use by the research community. FeatureNET is open source (under MIT license) and available on Github[3]. The repository lists the system requirements (Python, JDK for Pledge, NodeJs for the web interface, etc.) and provides installation instructions.

2 FEATURENET PIPELINE

Figure 1 shows the pipeline implemented in FeatureNET. It covers the full process of generating, training and evaluating DL models according to the user's metrics and experiments. Users can interact with FeatureNET through a web application with a Graphical User Interface (GUI), a Command-Line Interface (CLI), or an Application Programming Interface (API). The API includes the core functionalities that are also used by both the GUI and the CLI.

2.1 Parameterize the Generation

The first functionality is the parameterization of the generation process. Figure 2 shows our GUI. Mandatory parameters include the maximum number of blocks and cells that a model configuration can have, the number of configurations to sample, the dataset to train and test on, and the number of training epochs. During this step, the meta-model can also be specialized to restrict the generation to a subset of the architectures (e.g., LeNet architectures). This is achieved through the GUI or by tuning a dedicated XML file.

The FeatureNET meta-model abstracts DL model architectures as a sequence of blocks and cells (see Fig 3 for an illustration). Each block and cell has multiple attributes, which roughly correspond to the types of layer one can find in feed-forward neural networks, their parameters and internal computations.

2.2 Launch Model Generation and Training

After the parameterisation step, the user can launch a new model generation and training process. FeatureNET first parses the parameters and derives constraints under which the models will be generated. These constrained are fed into PLEDGE[4, 5], which then generates a diverse set of architecture configurations satisfying the input constraints.

Next, FeatureNET translates all configurations produced by PLEDGE into a Keras models and trains them on a specified dataset for a specified number of epochs. By default, the Cifar-10 [6] and MNIST [7] image classification datasets are considered. Though, users can choose other datasets through the CLI and API. Once a model is trained, the model is evaluated wrt chosen metrics. Users can either rely on predefined metrics like accuracy and empirical robustness [8], or plug-in tailored evaluation procedures via the API.

[3]https://github.com/yamizi/FeatureNet

Table 1: Test accuracy, size, and efficiency of LeNet5 and the best model produced by FeatureNET (out of 1,000) with a 12-epoch training, on MNIST dataset.

Architecture	Accuracy	Size	Efficiency
LeNet5	97.14%	545546	1.78
FeatureNET (best model)	97.74%	365194	2.68

2.3 Retrieve Generated Models and Evaluation Results

Once a configuration is assessed, the trained model is exported in *Keras HDF5 format*. A graphical representation of the model is exported for visualisation purpose. Finally a log file in JSON format reports the quality metrics computed on each model.

The GUI also provides visual feedback on the progress of the different steps of the pipeline. It also gives access to a visual report of all generated models and the metrics that have been assessed.

3 EVALUATION

We conduct preliminary experiments to evaluate FeatureNET. Here we report the main results, while the extended study is available in the paper presenting the FeatureNET approach [3].

Our first objective is to achieve diversity in the generated set of models. Thus, we ask:

RQ1: *Does FeatureNET generates a diverse set of models?*

To answer this, we use FeatureNET to generate 1,000 DL models and train them on the MNIST dataset for 12 epochs. Then, we measure the size of the models and assess their accuracy on the test set of MNIST.

Results are shown in Figure 4. By sampling a diverse set of architectures, our technique generates a wide range of sizes, from a few thousands to millions of weights. Moreover, we observe that high accuracy models are not necessarily the largest ones and that models performing poorly are of all sizes.

Since FeatureNET is a NAS method, we also measure its capability to produce high-performing models. Hence, we investigate:

RQ2: *Can FeatureNET generate models with high accuracy and high efficiency (i.e. accuracy over size)?*

Table 1 shows the performance achieved on MNIST after 12 training epochs by (a) a standard LeNet5 architecture and (b) the best of the 1,000 DL models generated by FeatureNET in the previous experiments. We observe that the best generated model outperforms LeNet5 both in accuracy and efficiency, which tends to confirm the viability of our NAS approach.

We also perform additional experiments to compare FeatureNET with NAS tools from literature. Interestingly, FeatureNET has competitive performance with open-source solutions like AutoKeras but falls behind commercial tools such as Google AutoML. One reason is that such tools rely on architecture and training enhancing methods (e.g. data augmentation), while FeatureNET works only on the architecture of the models.

4 EXAMPLE OF USE CASES

We illustrate the use of FeatureNET through 3 use cases. These are experiments we recently conducted as part of our research in DL.

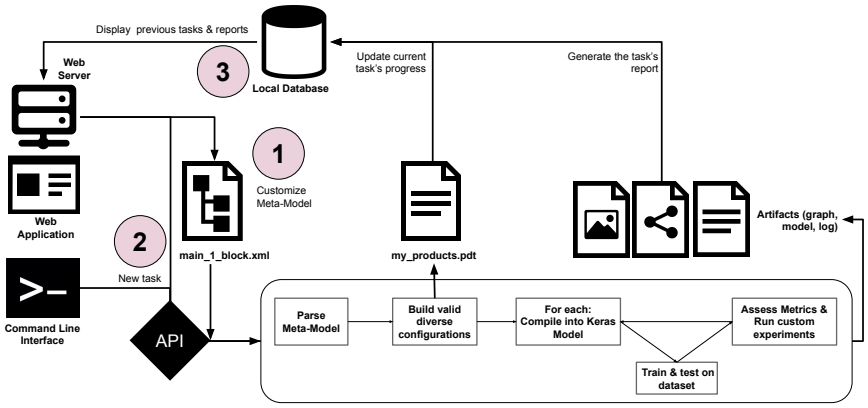

Figure 1: The FeatureNET pipline. The 3 main user interactions are denoted by red chips.

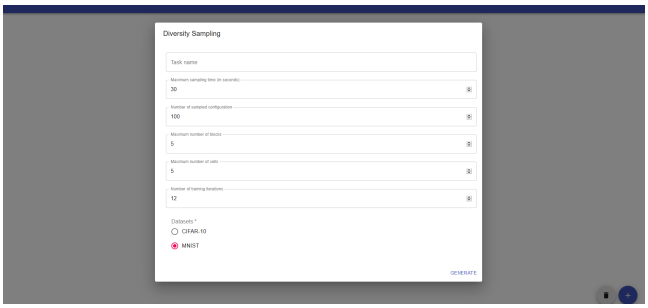

Figure 2: FeatureNET GUI: To schedule a new task, we set some parameters like the maximum number of blocks and cells per model, the number of models to train, and for how many epochs.

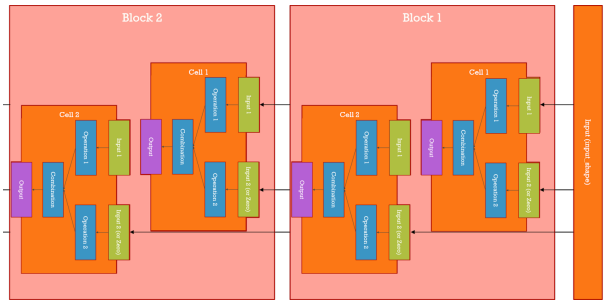

Figure 3: The DL architecture meta-model implemented by FeatureNET. A block is made of multiple cells and every component of a cell is a layer with different roles. For instance, *Input* components handle logical operations, while *Operation* components handle matrix transformation and regularization.

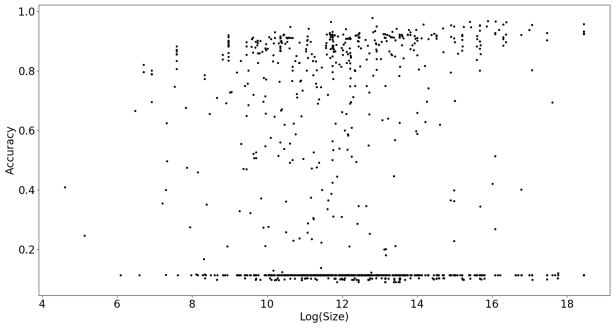

Figure 4: Distribution of the size (log scale) and the accuracy (on MNIST) of 1,000 generated models. The size is given in terms of number of trainable weights of the models.

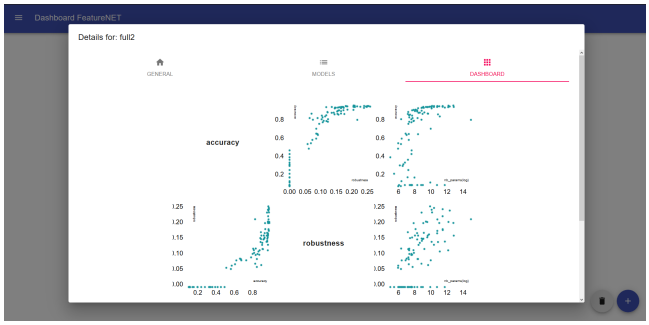

Figure 5: FeatureNET GUI results dashboard: Once the task is over, the dashboard provides an overview of the metrics that have been recorded. In the top center graph, the accuracy and empirical robustness of 100 models are plotted.

4.1 Study the Accuracy and Robustness of DL models

Our first use case concerns the link between model accuracy and robustness to adversarial attacks. The original study of Tsipras et al. [10] tends to indicate the existence of a trade-off between these two metrics. To complement their findings, we use FeatureNET to generate 100 diverse models and compute their accuracy and empirical robustness (two metrics implemented by default in FeatureNET) on the MNIST dataset.

```
1  from run import build_meta_model, build_models
2  from tensorflow_generator import TensorflowGenerator as TG
3
4  dataset, training_epochs  = 'cifar10', 50
5  nb_base_products=(5,5,1000)
6  meta_model = build_meta_model(nb_base_products)
7
8  def my_experiment(keras_model):
9   layers = keras_model.layers
10  first_layer_weights = layers[0].get_weights()[0]
11  return np.corrcoef(first_layer_weights)
12 TG.add_metric('corr_weight',my_experiment)
13
14 build_models(metamodel=meta_model, dataset=dataset,
   ↪  nb_base_products=nb_base_products[2],
   ↪  training_epochs=training_epochs)
15
16 for product in TG.products:
17  layer_w_metric = product.metrics.get('corr_weight')
18  print('Layer1 mean weights correlation:{}'.format(layer_w_metric))
```

Figure 6: Use of FeatureNET's API for custom experiments.

Once the models have been generated, trained and evaluated, we can observe the results in FeatureNET's GUI. Figure 5 shows the "Details" view of a task and the dashboard where we get plots of the metrics recorded for each model.

We see in the top center figure that accuracy and robustness improve together until the accuracy reaches a plateau. Robust models are also the most accurate. These preliminary results somehow contradict Tsipras et al.'s and pave the way for extended studies.

4.2 Steganography based on Diverse DL Models

In a recent work, we proposed a new steganography method (named adversarial embedding) to hide messages within images by applying adversarial attack algorithms on a DL model [2]. In this method, the model acts as a private key and decodes the message by classifying the received adversarial images. To preserve the confidentiality of the embedded information at large scale, it is essential to use a diverse set of models. The model generation method of FeatureNET inherently satisfies this requirement.

Moreover, a potential threat lies in the use of other models to recover the information. That is, if a model classifies the adversarial images in the same classes than the model used to produce them (the private key), a malicious third party can decode the message.

To evaluate this threat, we produced the adversarial images by applying the PGD adversarial attack algorithm on a ResNet model and the Cifar-10 dataset. Then, we designed an experimental pipeline integrating FeatureNET (via its CLI) to generate 100 models. We used those models to attempt decoding the hidden message. It turned out that none of the models succeeded, which indicates that our method is secure.

In another evaluation, we assessed different qualities of our steganography method (e.g., resilience to image tampering). FeatureNET allowed us to experiment on many different models, thereby increasing the validity of our conclusions. We also generated better models (w.r.t. these tailored quality metrics) than handcrafted models available in the literature.

4.3 Tailored Analysis of DL Models

Our last use case illustrates the use of FeatutreNET's python API to perform tailored analyses of generated models. For instance, we consider an artificial experiment where we compute the correlation between the weights of the first layer of trained DL models.

We provide a python code snippet in Fig. 6. At Lines 4-5, we specify that we want to generate 1,000 models with maximum 5 blocks and 5 cells each, trained on the Cifar-10 dataset for 50 epochs. Lines 8-12 set up the experiment to compute the correlation between the weights of the first layer of each model. Finally, We build the models in line 14 and print the experiments results in lines 16-18.

5 CONCLUSION

FeatureNET is a NAS tool that generates diverse DL models. We designed the tool so that it is extensible and easy to integrate with external experimental pipelines. We make FeatureNET available with the hope that it will support experimental studies at using large and diverse sets of models, thereby increasing the confidence in the validity of experimental studies.

In future we plan to extend FeatureNET with additional search methods, use enhanced DL methods, like data augmentation, and extend the expressiveness of the meta-model to support a broader range of models (e.g. recurrent and bayesian neural networks).

6 ACKNOWLEDGMENT

Mike Papadakis is supported by the Luxembourg National Research Funds (FNR) C17/IS/11686509/CODEMATES.

REFERENCES

[1] Benoit Baudry and Martin Monperrus. 2015. The Multiple Facets of Software Diversity: Recent Developments in Year 2000 and Beyond. *ACM Comput. Surv.* 48, 1 (2015), 16:1–16:26. https://doi.org/10.1145/2807593

[2] Salah Ghamizi, Maxime Cordy, Mike Papadakis, and Yves Le Traon. 2019. Adversarial Embedding: A robust and elusive Steganography and Watermarking technique. (2019). arXiv:1912.01487 [cs.CR]

[3] Salah Ghamizi, Maxime Cordy, Mike Papadakis, and Yves Le Traon. 2019. Automated Search for Configurations of Convolutional Neural Network Architectures. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 119–130. https://doi.org/10.1145/3336294.3336306

[4] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. 2014. Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize T-Wise Test Configurations for Software Product Lines. *IEEE Trans. Software Eng.* 40, 7 (2014), 650–670. https://doi.org/10.1109/TSE.2014.2327020

[5] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. 2013. PLEDGE: A Product Line Editor and Test Generation Tool. In *Proceedings of the 17th International Software Product Line Conference Co-located Workshops* (Tokyo, Japan) *(SPLC '13 Workshops)*. ACM, New York, NY, USA, 126–129. https://doi.org/10.1145/2499777.2499778

[6] Alex Krizhevsky. 2012. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (05 2012).

[7] Yann LeCun and Corinna Cortes. 2005. The mnist database of handwritten digits. (2005).

[8] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2015. DeepFool: a simple and accurate method to fool deep neural networks. (2015). arXiv:1511.04599 [cs.LG]

[9] Thilo Stadelmann, Vasily Tolkachev, Beate Sick, Jan Stampfli, and Oliver Dürr. 2019. Beyond ImageNet: Deep Learning in Industrial Practice. In *Applied Data Science*.

[10] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. 2018. Robustness May Be at Odds with Accuracy. (2018). arXiv:1805.12152 [stat.ML]

[11] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2019. Machine Learning Testing: Survey, Landscapes and Horizons. arXiv:1906.10742 [cs.LG]