Concurrent Neural Tree and Data Preprocessing AutoML for Image Classification

Anish Thite
Mohan Dodda
Alex Liu
Pulak Agarwal
Georgia Institute of Technology

Jason Zutty
Georgia Tech Research Institute

ABSTRACT

Deep Neural Networks (DNN's) are a widely-used solution for a variety of machine learning problems. However, it is often necessary to invest a significant amount of a data scientist's time to pre-process input data, test different neural network architectures, and tune hyper-parameters for optimal performance. Automated machine learning (autoML) methods automatically search the architecture and hyper-parameter space for optimal neural networks. However, current state-of-the-art (SOTA) methods do not include traditional methods for manipulating input data as part of the algorithmic search space. We adapt the Evolutionary Multi-objective Algorithm Design Engine (EMADE), a multi-objective evolutionary search framework for traditional machine learning methods, to perform neural architecture search. We also integrate EMADE's signal processing and image processing primitives. These primitives allow EMADE to manipulate input data before ingestion into the simultaneously evolved DNN. We show that including these methods as part of the search space shows potential to provide benefits to performance on the CIFAR-10 image classification benchmark dataset.

CCS CONCEPTS

• **Computing methodologies → Computer vision tasks**; **Search methodologies**; **Genetic programming**; **Neural networks**;

KEYWORDS

Deep Neural Networks (DNNs), Automated Machine Learning (autoML), Genetic Programming

1 INTRODUCTION

Deep Learning methods have been shown to improve state-of-the-art performance in a variety of tasks, including Natural Language Processing, Computer Vision, and Speech Processing. This has led to a flurry of activity in the field, with many different task-specific model architectures and hyper-parameters being invented. Due to the wide variety of possible architectures, hyper-parameters, it is often necessary to spend copious amounts of time tracking and experimenting with many different architectures and hyper-parameters for optimal performance. Neural Architecture Search (NAS) methods help automate this process. Methods treat finding the optimal architecture as a search problem and employ a variety of strategies in order to solve it. Hyper-parameter search methodologies also exist, and modern systems incorporate both hyper-parameters and architectures in the same search space.

Optimal performance can be measured with single or multiple objectives. For example, LEAF [10], a multi-objective system optimizes for both accuracy and model size. Since each algorithm is tested on both objectives, end users can select the ideal model based on their desired compute budget.

One major drawback with current systems is a lack of data pre-possessing in the search space. Rather, data pre-processing is hand-designed prior to feeding it into the system. Data pre-processing is arguably one of the most important pieces of a deep learning pipeline, and can dramatically affect the quality of the resulting model.

The Evolutionary Multi-objective Algorithm Design Engine (EMADE) tackles this problem. EMADE, previously introduced as GT-MOEP [19], includes data processing functions in its primitive set, allowing it to evolve both traditional machine learners and data pre-processing algorithms concurrently. Using the Keras API [2] we extend EMADE to include neural network-relevant primitives and terminals such as various types of layers, weight initializers, layer-level parameters, and network-level hyper-parameters. We show that modifying EMADE to evolve using both data pre-processing functions and neural network primitives improves accuracy on CIFAR-10[9], a popular benchmark for NAS frameworks, and can potentially improve on SOTA algorithms.

2 BACKGROUND AND RELATED WORK

Neural Architecture Search automates the process of determining the best combinations of layers and their optimal hyper-parameters for neural networks by treating it as a classical search problem. NAS methods comprise three components: a search space, a search strategy, and a candidate network performance estimation strategy [5].

Evolution-based search algorithms have been used widely to evolve neural networks. NEAT [14] was one of the first genetic

Anish Thite, Mohan Dodda, Alex Liu, Pulak Agarwal, and Jason Zutty

algorithms for neural network topology evolution, created by Stanley Miikkulainen in 2002. NEAT directly encodes neurons, their weights, and neuron connections as genomes. Then NEAT creates a population of genomes which then undergoes a biologically-inspired evolutionary process. Genomes can mutate (where the weights and connection values randomly change) and mate (where portions of two genomes combine to make a new one).

However, direct encoding of neurons to genes is not feasible at larger scales, due to the amount of memory required to store millions of neurons. CoDEEPNEAT [11] and HyperNEAT [13] encode neural networks indirectly by specifying layer parameters and sizes rather than individual neurons. This allows NEAT to be used to evolve larger and deeper neural networks. CoDEEP-NEAT also performs multi-objective optimization, resulting in a list of co-dominant networks (called a Pareto front). This allows CoDEEPNEAT to optimize for metrics such as accuracy and size simultaneously. Experiments on CoDEEPNEAT were run on a variety of datasets [10] including CIFAR-10 [11]. LEAF creates networks from scratch and trains them for 8 epochs during the evolution process. Once the evolution process is complete, the best network is re-trained for 300 epochs. LEAF achieves a 7.3% error rate on CIFAR-10.

EMADE is a automated ML framework based on the DEAP [6] genetic programming framework. In the DEAP framework, algorithms are expressed as trees, with functions expressed as nodes and parameters expressed as terminals. When evaluating the tree, nodes are computed in order of decreasing depth with outputs from child nodes being passed as inputs into their parent nodes. EMADE uses vector based genetic programming, where the data passed between primitives is a vector of elements. EMADE uses high-level primitives such signal processing and computer vision functions along with traditional machine learning functions.

Through the evolutionary process EMADE optimizes combinations of these classifiers and pre-processing methods and their respective hyper-parameters. EMADE's evolutionary process follows a similar algorithm to NEAT and other genetic algorithms. An initial population of individual trees are created and evaluated. Each generation, n trees are selected via a selection function, and are used to create new trees to add to the population. EMADE uses NSGA-II [3] as the selection function, and a variety of mating and mutation operators for tree generation. EMADE has been used for a variety of domains [7, 18, 20], but has never been used on Deep Learning applications.

3 METHODS

3.1 Neural Network Modifications

We extend EMADE to include a neural network-based learner primitive, Layer primitives, and neural network-centered mating and mutation functions.

3.1.1 LayerTree. The LayerTree class is a pre-ordered list representation of the neural network tree. Primitives add Keras layers to this tree, resulting in a tree-structured computational graph.

3.1.2 Layer Primitives. It is necessary to encode the neural architecture in a way visible to the EMADE framework. Therefore, a multitude of neural network layer types from the Keras API are

Table 1: Hyper-parameters for Search Space

Hyper-parameter	Used In	Range
Output Dimension	Core-Layer	int
Activation Function	Core-Layer	[SeLU, ELU, Sigmoid, ReLU, Leaky Relu, Softmax, TanH]
Weight Decay	Core-Layer	float
Kernel Dimension	CNN	int
Stride Size	CNN	int
Padding Style	CNN	Same or Valid
Batch Size	NNLearner	int
Optmizer	NNLearner	[Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax, Nadam, Ftrl]
Dropout Percentage	Dropout	float

added as EMADE primitives. Keras Layers are added as Layer primitives. Each primitive accepts a LayerTree representation of the model built from its child node as a parameter. It creates a Keras layer and adds the layer as the new root, and passes the modified tree on to its parent node. In creating the Keras Layer, EMADE can specify various layer parameters via other primitives and terminals. These parameters are shown in Table 1.

We include Dense and Convolution Layers in the primitive set since CIFAR-10 is an image classification task, and CNNs have performed well on this task. We also add layer primitives including Dropout, BatchNormalization, MaxPooling, GlobalMaxPooling, and GlobalAveragePooling. Concatenate layers are also necessary for EMADE to generate tree-based neural networks. These concatenate a series of previous layers' outputs, and can be used to condense two branches of the tree together. Therefore, we add a Concatenate layer primitive to the set. The primitive takes the output of two child LayerTrees and concatenates them.

3.1.3 Pre-trained Layer Primitives. Many state-of-the-art image classification models show that pre-training on ImageNet yields significant performance gains [4] [17] [16]. Building off this work, we incorporated pre-trained models within the evolutionary process. We add each pre-trained network as a single layer. While it does not allow EMADE to modify the architecture of the pre-trained network, EMADE is easily able to add other primitives before or after these layers. We implement VGG [12], Mobilenet [8], and Inception [15] architectures, which have been shown to perform well for image classification tasks. For all of these layers, we used pre-trained ImageNet weights. However, we do not freeze these weights, allowing them to be updated during the training regime for each tree.

3.1.4 NNLearner. The NNLearner takes in data, batch size, optimizer, and a LayerTree. It creates a neural network from the LayerTree specifications using the Keras Functional API. When evaluating, it trains the network on the train subset of the data, and validates it based on the validation set of the data. NNLearner uses early stopping [1] to determine the optimal epoch to stop training using validation data to determine the early stopping point.

3.2 Image Pre-processing Primitives

EMADE supports over 150 primitives that operate on image data, many leveraged from OpenCV. These include various filters, transformations, and other image processing techniques. EMADE also contains more than 60 signal processing functions leveraged from Python packages such as numpy, scipy, and scikit-learn.

Together, these primitives enable a large host of well-researched techniques to be applied to instances of data during the algorithm

Table 2: Experimental Setup

Parameter	Small Run	Large Run
Seeded Population Size		16
Initial Population Size (with seeded population)	64	512
Walltime		24 hours
Nodes		4
GPU / node		1, 32GB TeslaV100
Evaluations / node		3
Mating - crossover		50%
Mating - crossoverEphemeral		50%
Mating - headlessChicken		10%
Mating - headlessChickenEphemeral		10%
Mutation - insert		5%
Mutation - insert modify		10%
Mutation - ephemeral		25%
Mutation - uniform		5%
Mutation - shrink		5%
Mutation - swap layer		5%
Mutation - remove layer		5%
Mutation - add layer		5%

design process. Primitives in EMADE tend to fall into two categories, those that fit to training instances of data and then apply to those used for scoring (e.g. transforms), and those that apply to each instance independently (e.g. filters).

Some methods that we expect to be useful in pre-processing for image classification tasks include primitives such as: histogram of oriented gradients, hsv histograms, Haralick features, HuMoment features, edge detection filters, blurring filters, gray-scale conversions, normalizations, thresholds, and spectral representations such as Fourier transforms, wavelet transforms, and discrete cosine transforms.

3.3 Mutation Functions

Various mutation functions were added to EMADE's mutation function set in order to aid neural network evolution. These mutation functions include randomly adding a layer to the tree representation, randomly swapping a layer with another layer, and randomly removing a layer. With these three simple mutation functions, EMADE has the potential to reach any architecture in the search space. Specific mutation functions were also added to change the activation function of a layer, the optimizer used, as well as the pre-trained weights used in the embedding layer.

3.4 Mating Functions

Several new mating functions were added to EMADE's existing ones to improve mating between tree-based neural network algorithms. One mating function was a single point crossover that operates on a random point that has an NNLayer output type. This function allows the creation of a child network whose gene composition has its first partition of layers from the beginning of one parent up to the crossover point and the second partition of layers from the crossover point to the end of the other parent (and vice versa for the other child) as shown in ??. A modified version of this function was also added to aid in expanding neural network size. Instead of replacing the latter half of each network, each child network has their original layers in addition to the swapped sub-trees from the parents. Note that this is not a two point crossover.

4 EXPERIMENT

The CIFAR-10 Dataset is a multi-class benchmark dataset for Image Classification. It contains 60,000 32x32 color images in 10 classes

with 6,000 images per class. Classes consist of different vehicle and animal types. We separate the images into a 50,000 image train split and a 10,000 image test split.

We initially seed EMADE's starting population with 16 hand-created NNLearners to bias the evolution process to create neural networks. Each seed is an NNLearner containing a pre-trained network layer, followed by a single Dense layer with a sigmoid activation function to act as an output layer. The pre-trained networks include either a VGG, MobileNet, or Inception architecture. Adam is set as the optimizer, and batch size differs between 1 to 9. The remaining individuals in the population are randomly generated. Using the configurations in Table 2, we performed two separate runs with the same seeded individuals but with different run sizes. Both runs were given approximately the same 288 total compute hours. The small run accomplished 109 generations and evaluated 13,896 individuals. The best individual accuracy was 75.59%. The large run accomplished 28 generations and evaluated 34,989 individuals. The best individual accuracy was 73.84%.
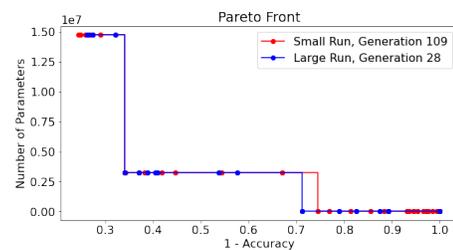
5 RESULTS

Figure 1: Final Pareto Fronts

Final Pareto Fronts are shown in Figure 1. Even though the two runs performed a different number of generations and evaluated a significantly different amount of individuals the resulting Pareto Fronts are nearly identical. As seen in Figure 2, the best accuracy observed in an individual improved over the course of evolution. Note that the best individual accuracy improved the most in the early generations for both runs. Since both runs have the same accuracy/generation profiles, it seems that EMADE does not take advantage of the larger population to optimize more quickly. This similarity may be due to the effects of seeding, since the seeded portion of the starting population is the same.
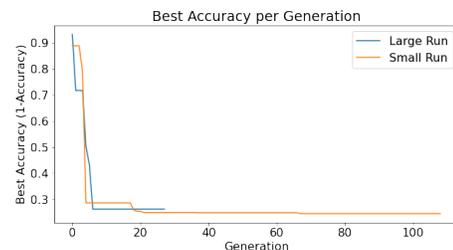

Figure 2: Best Accuracy per Generation

Anish Thite, Mohan Dodda, Alex Liu, Pulak Agarwal, and Jason Zutty

An example individual evolved is shown in Figure 3. It takes advantage of both EMADE pre-processing and NN-based primitives. In this particular example, EMADE optimized the seeded pre-trained MobileNet network by manipulating the batch size, switching the optimizer from Adam to SGD, and applying a cosine window on the data before training/testing on it.
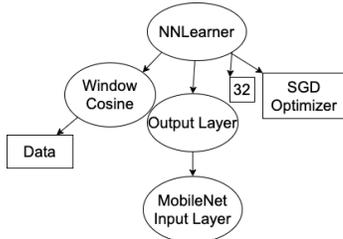

Figure 3: Individual with Pre-processing Primitive(s)

6 CONCLUSION

The paper shows a proof of concept that uses neuroevolution to look through a search space to find optimal neural network architectures across different objectives for image classification tasks. By evolving pre-trained networks and pre-processing techniques with the neural network architectures, we expand on current work to benefit the performance of the CIFAR-10 image classification dataset. Although our current results are not yet comparable with human generated algorithms, it is worth noting that our algorithm was discovered by the EMADE neural network framework on its own using genetic programming. There is a lot of potential for machine generated algorithms to perform at an extremely high level.

We propose this tool to be helpful for other classification tasks where neural networks are used. There is also ample opportunity for future work in improving EMADE's neural architecture search, such as supporting skip connections within nodes in trees, investigating building and preserving tree complexity, and developing ways to more efficiently evaluate individuals.

When EMADE's framework is further developed with more competitive results for the CIFAR-10 dataset, we would like to test our results against other state-of-the-art algorithms and further investigate the benefits of manipulating input data as part of the algorithmic search space. Future research will also include more rigorous testing with other datasets (especially unbalanced ones) and drawing comparisons to other AutoML frameworks.

REFERENCES

[1] [n. d.]. EarlyStopping. https://keras.io/api/callbacks/early_stopping/. ([n. d.]). Accessed: 2021-04-12.

[2] Francois Chollet et al. 2015. Keras. (2015). https://github.com/fchollet/keras

[3] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International conference on parallel problem solving from nature*. Springer, 849–858.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. (2020). arXiv:cs.CV/2010.11929

[5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. (2019). arXiv:stat.ML/1808.05377

[6] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[7] Joel D Griffies, Jason Zutty, Marcel Sarzen, and Stuart Soorholtz. 2018. Wearable sensor shown to specifically quantify pruritic behaviors in dogs. *BMC veterinary research* 14, 1 (2018), 1–10.

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. (2017). http://arxiv.org/abs/1704.04861 cite arxiv:1704.04861.

[9] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[10] Jason Zhi Liang, Elliot Meyerson, Babak Hodjat, Daniel Fink, Karl Mutch, and Risto Miikkulainen. 2019. Evolutionary Neural AutoML for Deep Learning. *CoRR* abs/1902.06827 (2019). arXiv:1902.06827 http://arxiv.org/abs/1902.06827

[11] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 293–312.

[12] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[13] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. 2009. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life* 15, 2 (2009), 185–212. https://doi.org/10.1162/artl.2009.15.2.15202

[14] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. *CoRR* abs/1409.4842 (2014). arXiv:1409.4842 http://arxiv.org/abs/1409.4842

[16] Mingxing Tan and Quoc V. Le. 2021. EfficientNetV2: Smaller Models and Faster Training. (2021). arXiv:cs.CV/2104.00298

[17] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. 2021. CvT: Introducing Convolutions to Vision Transformers. (2021). arXiv:cs.CV/2103.15808

[18] Jason Zutty, Domenic Carr, Rodd Talebi, James Rick, Christopher Valenta, and Gregory Rohling. 2019. Reducing bathymetric-lidar algorithm uncertainty with genetic programming and the evolutionary multi-objective algorithm design engine. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, Vol. 11006. International Society for Optics and Photonics, 110061U.

[19] Jason Zutty, Daniel Long, Heyward Adams, Gisele Bennett, and Christina Baxter. 2015. Multiple objective vector-based genetic programming using human-derived primitives. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 1127–1134.

[20] Jason Zutty, Rodd Talebi, Austin Dunn, Roland Samuelson, and Gregory Rohling. [n. d.]. EMADE for Intelligence Surveillance and Reconnaissance Applications. ([n. d.]).