CIFAR-10 and MNIST Data Classification Performance as a function of Genetic Algorithms in Python for Computer Vision using scikit-learn, Keras, and OpenCV

Matt Fleetwood
ECE 479 Intelligent Robotics II
Maseeh School of Engineering, Portland State University

2018
March

Contents

1 Abstract

The goals of this project are two-fold. One goal is to learn about how the classification performance for a Multi-layer Perceptron (MLP) neural network changes as a function of the architecture parameters (e.g. number of neurons, layers, and activation function). Another goal is to learn how morphological operators in OpenCV change the classification performance for a classifier such as a Convolutional Neural Network (CNN). Here, two genetic algorithms are employed, one written from scratch and another by Matt Harvey, in order to search for solutions to the previous goals. These methods show that by utilizing a genetic algorithm, the training time decreases while the overall accuracy increases for the MLP and CNN. Applications in the context of robotics are discussed.

2 Introduction

In computer vision applications, there are many parameters that configure an artificial neural network. For instance, a Multi-layer Perceptron can contain a variable number of layers as well as a variable number of neurons in each layer. There can also be several different activation functions available to each neuron, such as hyperbolic tangent, sigmoid, softmax, or ReLU. Beyond these options, there are also other methods to optimize a network: training rate, number of training iterations, batch- versus online-training, and other subtle choices such as deciding which optimizer to use (e.g. stochastic gradient descent, RMSProp, Adagrad, Adadelta, and so on). This creates a problem for Machine Learning (ML) or Computer Vision (CV) practitioners because deciding which combination of features yield the best classification results can be a difficult and time-consuming task.

One possible solution to neural network architecture search is to use an evolutionary algorithm, which have for the first time achieved state of the art classification results [2]. This method employs an automatic mechanism to find the most fit architecture or set of configurable parameters. In doing so it can eliminate time and effort spent tailoring neural networks to specific applications. Matt Harvey showcases one implementation for a genetic algorithm that evolves neural networks in order to optimize classification accuracy on his GitHub. According to Harvey, his implementation reduces the training time for an MLP on the MNIST dataset from 63 hours to 7 [4].

Another potential solution to automatically configuring a neural network for better classification accuracy is to meaningfully process the data. One way to manipulate image data is to use morphological operators. Simply put, "it is called morphology because it aims at the analysis of the shape and form of objects. It is mathematical in the sense that the analysis is based on set theory, integral geometry, and lattice algebra"; it "is not only *theory*, but also a powerful image analysis technique" [Morphological Image Analysis: Principles and Applications]. Interestingly, "in industrial vision applications, the operations of mathematical morphology are more useful than the convolution operations employed in signal processing because the morphological operators relate directly to shape" [6]. Put differently, morphology is a powerful mathematical idea for computer vision applications. For example they can help identify objects of interest in image classification such as in medical imaging [5].

2.1 Evolutionary Algorithm from scratch in Python: Morphological Evolver

There are scores of tutorials on-line that describe the use of evolutionary algorithms for various applications. A brief list of these tutorials is given here:

- Was Darwin a Great Computer Scientist?
- Let's evolve a neural network with a genetic algorithm
- DEAP: Distributed Evolutionary Algorithms in Python
- Evolutionary Algorithm
- Introduction to Genetic Algorithms with Python

These tutorials, along with Professor Marek Perkowski's lectures, were the inspiration to build an implementation of an evolutionary algorithm in Python. According to Perkowski, on his lecture slide Steps for a Genetic Algorithm from the *Genetic Programming* lecture, there are three main features to a genetic algorithm: mutation, crossover, and selection. For this project, a chromosome represents a list of morphological operators, which can either be present (indicating the gene is turned on) or absent (indicating the gene

is turned off). During a population update, the best two chromosomes are found. Here, the best chromosome is the one in the population that was used for a CNN that achieved the highest classification accuracy after applying the chromosomes to the input data to the CNN. Once the best two individuals are found, they crossover and create two new children. The children replace the least two fit individuals in the population. Another important note about crossover is that mutation should occur afterwards but only for the children. All code for this project can be found on the author's GitHub.

2.2 Multi-layer Perceptron Networks in scikit-learn

The scikit-learn API was used to test an MLP on the MNIST dataset. In scikit-learn, the MLP is configurable with a number of properties such as the size of the hidden layer, training iterations, solver, and learning rate. The API describes an MLP as "a supervised training algorithm that learns a function ... " [7]. It uses built-in methods to retrieve and operate on the MNIST dataset. As discussed in Professor Marek Perkowski's lecture, *Naive Bayes*, the MLP falls under the second classifier section b as it is a structure that "models the probability of class memberships given input data".

2.3 Convolutional Neural Networks in Keras

The Keras API is an alternative to other ML APIs, such as scikit-learn or Orange, and was used to test CNNs on the CIFAR-10 and MNIST datasets (described below in Test and Training Data). Keras describes itself as "a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano" [3]. Like scikit-learn, Keras also employs ready-to-use methods for retrieving datasets like CIFAR-10 and MNIST. In fact, Keras seemed slightly advantageous to scikit-learn in this regard because scikit-learn did not include a method to retrieve CIFAR-10.

2.4 Test and Training Data using scikit-learn, Keras, and OpenCV

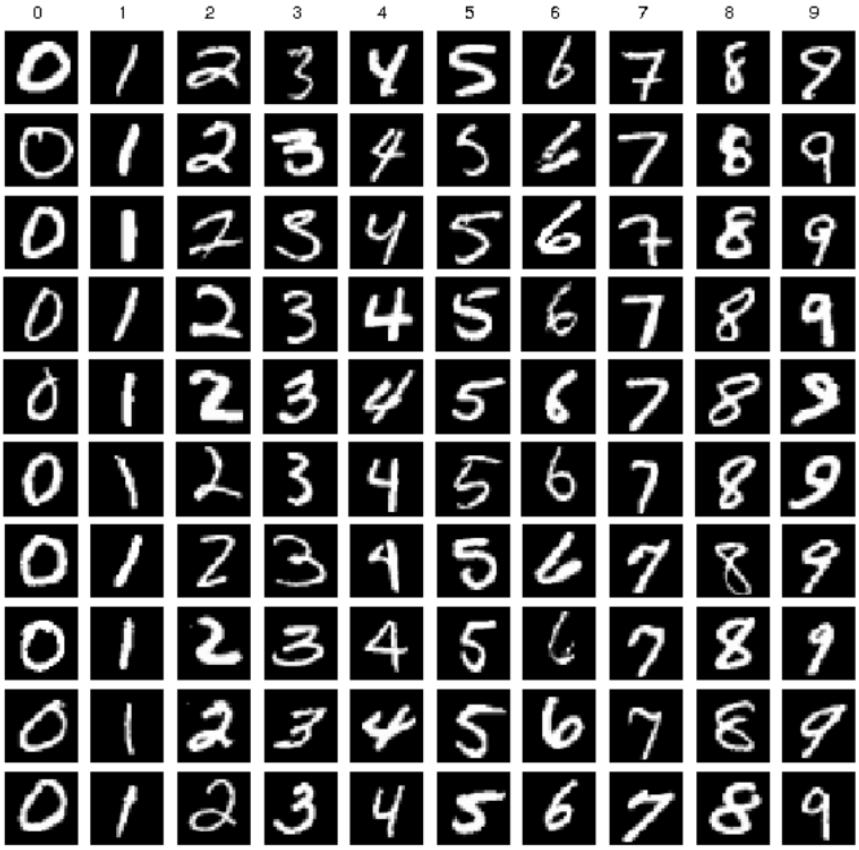

Figure 1: MNIST data from researchgate.

The MNIST dataset represents labeled handwritten images, which contain a number (0 - 9). There are 60,000 examples in the training set and 10,000 in the test set [8]. An MLP in scikit-learn was trained over 10 iterations, with 50 hidden units, using a learning rate of 0.1 and stochastic gradient descent (SGD) following the example here.

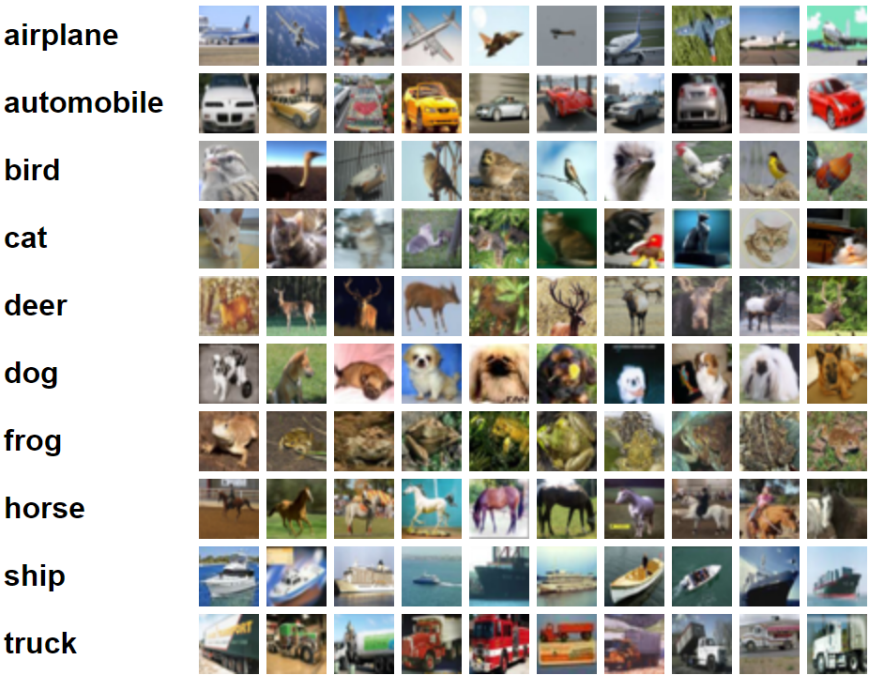

Figure 2: CIFAR-10 data from the University of Toronto.

The CIFAR-10 datasent represents labeled images of ten classes, as shown in Figure 2. It is a "subset of the 80 million tiny images dataset ... collected by Krizhevsky, Nair, and Hinton" [1]. A CNN was trained using the Keras and OpenCV APIs. CNNs were trained using the full dataset, as well as portions of the full dataset such as the first 500 and the first 1000 samples in both the training and testing data. A batch-size of 32 and epochs ranging from [1, 100] were used. OpenCV was utilized for its readily available morphological operators:

- Erosion
- Dilation
- Opening
- Closing
- Gradient

The effects of these operators can be seen on the 1st and 50th training samples below, respectively. The morphological operators were stored in a list in Python, where the two different methods were used. In the first, a fixed list order was created such that the first element in the list represented erosion, the second represented dilation, the third opening, the fourth closing, and the fifth gradient. In the second method, there was no fixed ordering, which meant that the operators could be applied in any order (not simply erosion, dilation, opening, closing, then gradient, as in the first method) and thus needed to be encoded. A base-ten encoding scheme was used such that 0 mapped to no operator (the absence of a gene, in this case a morphological operator), 1 mapped to erosion, 2 mapped to dilation, 3 to opening, 4 to closing, and 5 to gradient. Using these approaches, an implementation of an evolutionary algorithm was created from scratch in order to evolve the most fit combination of morphological operators for an untrained CNN.

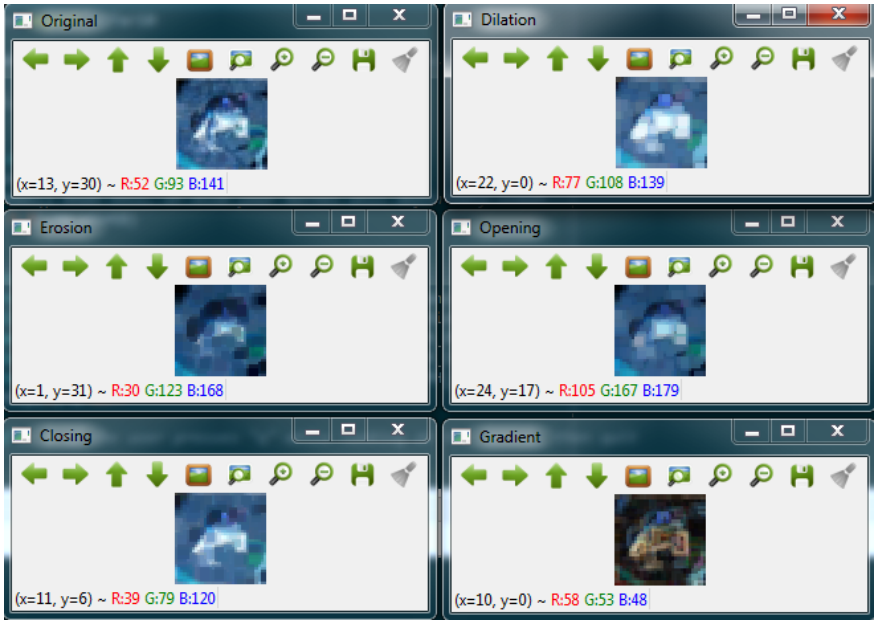

Figure 3: First training sample, a frog, from CIFAR-10 with morphological operators applied.

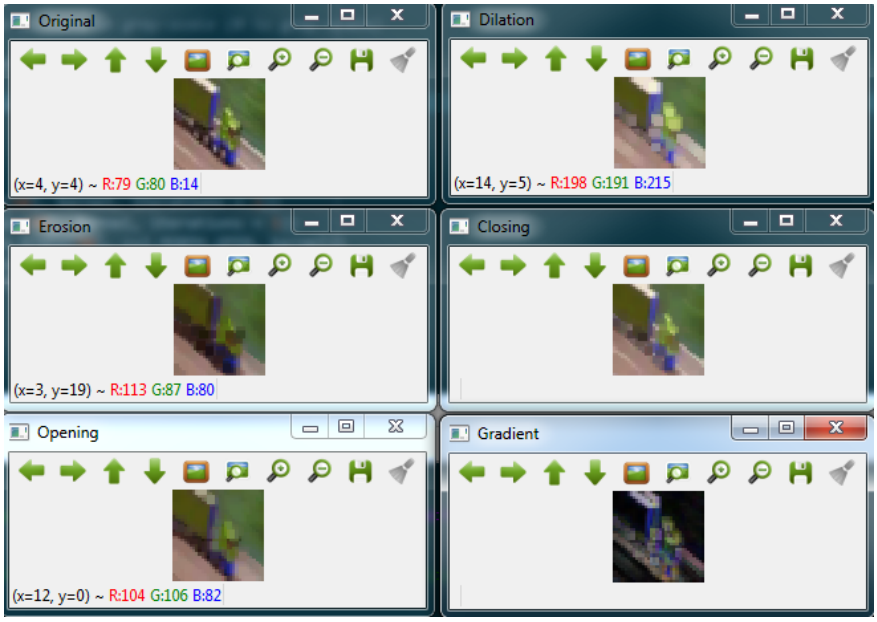

Figure 4: Fiftieth training sample, a truck, from CIFAR-10 with morphological operators applied.

2.5 Results

Table 1: CNN in Keras training results on 500 CIFAR-10 images using a batch-size of 32, 500 epochs, ReLU and softmax activation functions, where closing, dilation, and gradient operators were applied to the data prior to training.

Training Iterations	Loss	Accuracy	Validation Loss	Validation Accuracy
1	2.0219	0.2619	1.7651	0.3801
58	1.1290	0.6152	1.0663	0.6398
65	1.1417	0.6104	1.1086	0.6503
73	1.1559	0.6092	1.0906	0.6376
81	1.1716	0.6063	1.0748	0.6350
102	1.2250	0.5901	1.1964	0.6085
110	1.2656	0.5702	1.2443	0.5712
116	1.2903	0.5691	1.1241	0.6369

Table 2: Morphological evolution results using CNNs that are trained for 10 iterations per every chromosome.

Generations	Average Chromosome Accuracy
1	0.0762
10	0.0966

The MLP in scikit-learn was simple to train on the MNIST dataset. The training accuracy reached over 90 per-cent after 10 iterations, which is significantly faster than the model trained in Keras. On the other hand, Keras achieved the same accuracy percentage on the CIFAR-10 dataset after 100 iterations. It makes sense that the Keras model required more training epochs since the CIFAR-10 dataset is much larger than the MNIST dataset portions used in these experiments.

As shown by Harvey, the Keras model was able to employ his genetic algorithm over just 7 hours to train an equivalent model that doesn't use the genetic algorithm, which took 63 hours. The genetic algorithm build from scratch tended to increases the accuracy for every CNN across each generation. For example, using only 1 training epoch for every CNN, a population size of 10 with 20 generations and 5 genes available to every chromosome, the average accuracy increased from 0.103 to 0.129. It's worth noting that only the first 1000 testing and training images from CIFAR-10 were used. This was necessary in order to reduce the time complexity of the search. For instance, on average each CNN training iteration required 10 sec per every 1000 testing and training images. This means if more accuracy per every set of chromosomes is desired, then the amount of computation time doubles. As a result, only 1000 data samples are used from CIFAR-10 to test the accuracy of the morphological evolver. In fact, using the full CIFAR-10 dataset required approximately 1 hour per every generation of 10 individuals for the CNN training, which only used 1 training epoch. This implies that pre-processing the image data by applying a particular order of morphological operators reduces the training time while increasing the overall accuracy of image classification. The most fit operators tended to be only dilation, or closing followed by erosion then two closing operations.

Clearly, using a genetic algorithm can help reduce the efforts spent on fine-tuning a neural network architecture for classification. This is evident with respect to parameters such as number of layers and neurons, noting Har-

vey's implementation. It is also apparent in the genetic algorithm created from scratch, which focuses on processing the data by using morphological operators instead of architecture parameters such as the activation function. One idea for future experiments would be to combine the results of two separate genetic algorithms: one that searches for optimal neural network parameters (e.g. connectivity, number of neurons or layers, etc.), and another that searches for the most fit data pre-processing method (such as a string of sequentially applied morphological operators as shown here). This combination might potentially achieve greater classification accuracy than either independent solution. Applying another level of abstraction, and keeping these two genetic algorithms in mind, one could employ a third evolutionary algorithm that optimizes over the results of the first two. Considering that nature was able to create the human brain over billions of years of evolution, it seems likely that utilizing genetic algorithms for machine learning optimization solutions could be worthwhile.

2.6 Applications in Computer Vision and Robotics

There are numerous applications in CV and robotics that could take advantage of the experiments discussed above. For instance, a Vikingbot could be used, along with a camera, to detect images. A CNN could be trained in Keras, applying the most fit string of morphological operators found using the morphological evolver. After training, the network can be saved using data persistence methods in Python, such as the pickle method. The Vikingbot would then use the trained data structure to detect and react differently to different classes based on the CIFAR-10 or MNIST datasets. It could theoretically be used to read hand-written digits or react a particular way (e.g. blinking an LED, using an LCD to display a phrase) if it detects one of the ten classes in CIFAR-10, like a dog or a car.

Another application could be used on the Frankenstein robot. Since Frankenstein uses the Microsoft (MS) Kinect for voice and gesture recognition, then this robot could also detect images that represent classes from the CIFAR-10 dataset. Frankenstein already uses reactive software to encode and decode its interactions with the outside world. This makes it trivial to train a CNN and load it onto Frankenstein's ArduinoMega board.

3 References

[1] https://www.cs.toronto.edu/ kriz/cifar.html

[2] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," pp. 114, Mar. 2018.

[3] https://keras.io/

[4]M. Harvey, "Let's evolve a neural network with a genetic algorithm - code included," Coastline Automation, Apr. 2017.

[5] M. Saha and C. Panda, "A Review on Various Segmentation Techniques for Brain Tumor Detection," International Journal of Scientific Research in Computer Science, Engineering, and Information Technology, ol. 3, no. 1, pp. 110, 2018.

[6] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image Analysis Using Mathematical Morphology," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-9, no. 4, pp. 532550, Jul. 1987.

[7] https://scikit-learn.com

[8] http://yann.lecun.com/exdb/mnist/