Maggie Cao
23245277
1738 63rd Street 2nd FL
Brooklyn, NY 11204
(212) 380-3142
12-18-2017
CISC 4900 - Fall 2017
Paula Whitlock
Brooklyn College
Final Report: Simple Shapes Using Convolutional Neural Networks

Introduction:

Pattern classification and recognition, which is a field of machine learning, has been one of the most challenging tasks for a computer. This is because computers do not learn the same way as humans. Humans can generalize and have temporary memory, making it difficult for machines to be hard-coded in the natural technicalities of the human brain. However, neural networks embody a similar organization of how neurons interact with each other.
Each weight of single neuron in the neural network represents how strong the neuron's knowledge is about the data input. The neurons are connected to each other in each layer and in between layers, which is also called a fully connected layer. As each input data is propagated forward through the hidden layers of the neural network, there are activation functions which acts as a summation of the input values multiplied by the weights for the learning features to be feed into the next layer. As it reaches the last layer output, which is the main classifier because it determines the probability of the class, the data is backpropagated using sophisticated multivariable calculus to update the weights.

Without the weight updates, the neural network would not be able to learn. Each epoch describes a single forward and backward pass. Since the project is about classifying images of simple shapes, which includes circle, triangle, rectangle and square, a convolutional neural network will be implemented. A convolutional neural network do not need feature extraction in the preprocessing of the input images. The layers of the convolutional neural network act as a feature detector by extracting actual pixels from the data image, gathering features from filters and maxpooling. Then, the convolutional neural network will be connected to a traditional fully connected neural network to perform image classification.

Problem, Solution and Enhancements:

There are a lot of problems that need to be solved in the simple shapes image classification project. The hardest part of this simple shapes image classification problem is to be able to get the convolutional neural network to generalize well without overfitting or underfitting the network model. It is hard to generalize because the input data is never perfect, and it is redundant for the convolutional neural network to process background data without the class object. The input data is created by using Python's Cairo module, where single shapes of each class is drawn on random locations, rotations and scales on the 300 pixels by 300 pixels image. The final solution to deal with the unnecessary background data is to use the Sobel algorithm to detect edges, crop the image, and paste it on the 200 pixels by 200 pixels canvas with the same background color. This must be done separately before the main convolutional neural network loader file because I need to manually check if data image is good. Good data means that the square and rectangles are not shifted to the side, the triangles are not so small and that the circles do not overextend the boundary, making the image indecipherable. Then, the 200 pixels by 200 pixels cropped image will be feed into the convolutional neural network without the extra preprocessing function in the Keras' Image Generator.

A typical CPU will not be able to execute a convolutional neural network. The convolutional neural network will be trained in Google Cloud's virtual machine because a typical job submitted to the Google Cloud gives a memory error if the data input is around 8000 images for training. Increasing the number of GPUs for the job would not work because each GPU is assigned a fixed memory limit for a Google Cloud job. A virtual machine will allow the user to manually install Tensorflow from source. This is because it is very time consuming to train a convolutional neural network. It usually takes about two hours for training. Tensorflow is installed from source, which enables AWX instructions with GPU CUDA support for the Tesla K80. This speeds up the computation of the convolutional neural network by three times as without installing Tensorflow from source. Hyperparameter tuning of the dropout layers is an option for the virtual machine to be implemented with Google Cloud's storage bucket of the project.

Scope of the Work and System Documentation:

The scope of the actual work completed is in the next pages with detailed explanations. The project logs are on the website https://mahgieeee.github.io/.

Summary:

During training of the convolutional neural network, the network is not able to go below a loss of 0.38 or reach an accuracy higher than 80% for any epoch. Based on the documentation, the network looks like it is being overfitted because the validation accuracy is lower than the training accuracy. I find that very peculiar and originally thought it was because I was using an incorrect network architecture and loss optimizer for the loss function. I tested on many versions of different network architectures, activation functions and optimizers, but the network still won't go below 0.38. The major mistake I made was that I assumed I had perfect data images since I made my own data. I didn't check if the data images are reliable. I had to manually check every training and validation image data to ensure that the data is clear for the neural network to process. After fixing the data images, the training of the convolutional neural network is being underfitted because the validation accuracy is twice as much as the training accuracy and the validation loss is half of the training loss as the first several epochs. This is shown in training output documentation:

208/207 [==============================] - 62s - loss: 0.8218 - acc: 0.6593 - val_loss: 0.4303 - val_acc: 0.8694

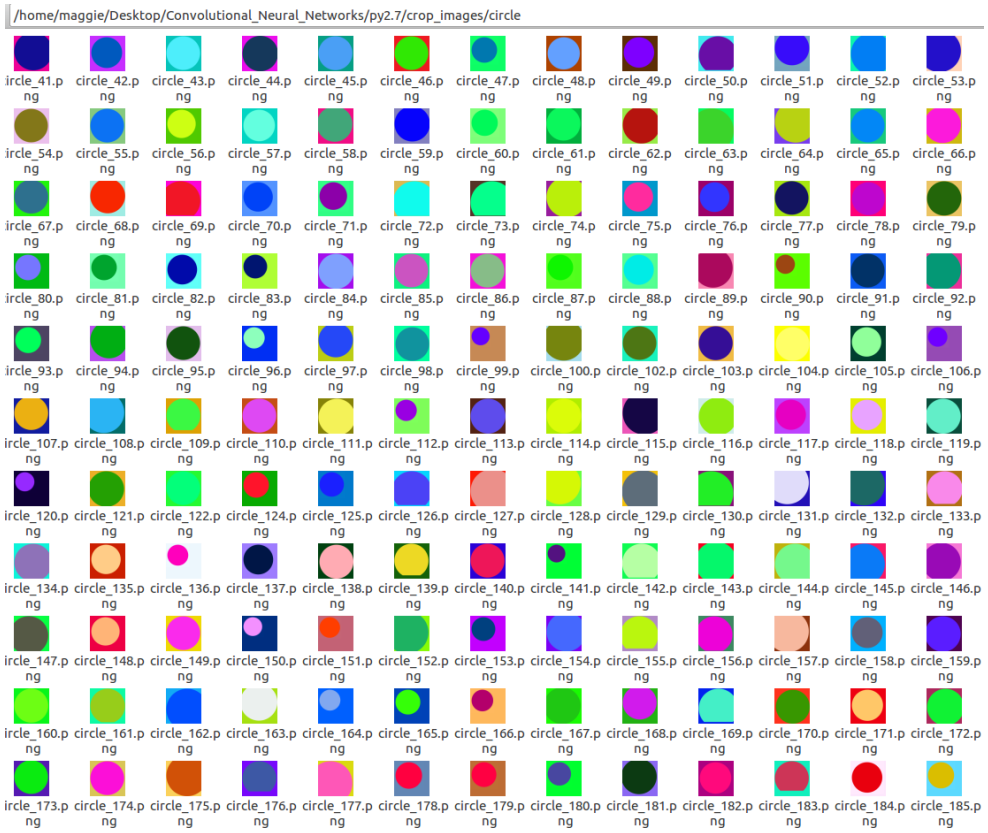
I tried increasing the layers of the convolutional neural network and reducing the dropout and maxpool layers and it seems to work. The current convolutional neural network architecture is described in the below chart. It also trains in PNG data instead of jpg because the compressed pickled files are much smaller (65MB).

Adam optimizer of 200 by 200 images (16,000 for training, 1,500 for validation

Num of Filters	64	64		64			256	256	256	4
Layer Type	CONV_2D	CONV_2D	MAXPOOL	CONV_2D	MAXPOOL	Flatten()	Dense()	Dense()	Dense()	Dense()
Conv. Size	(3,3)	(6,6)	(6,6)	(6,6)	(6,6)			Dropout (0.15)	Dropout (0.15)	
Padding	valid	valid		valid						
activation	relu	relu		relu			relu	relu	relu	softmax

Complete Description of tasks mentioned in the proposal but not accomplished with reasons:

Image classification and detection are two separate things in terms of getting the code to work. I assumed that they have buildable and very similar architectures, but they are completely different. This project only does image classification not image detection since image detection requires a recurrent convolutional neural network to be implemented, such as the FAST CNN. I am also not able to get the model to generalize well and I am still confused about why the model couldn't learn well. The final code also does not account for intersections between shapes as stated in the interim status reports. This is because the loss of the convolutional neural network during training wouldn't go down, so I had to simplify my dataset to consist of a single shape per image data instead of two



variations of the same shape in an image. This is the simplified version of the current dataset and the cropped version of it, which is the actual input data for the convolutional neural network.

Evaluation:

I think I did considerably well considering that I did not have any prior knowledge or experience in machine learning and convolutional neural networks. I had to learn a new programming language Python 2.7 and Keras from scratch. I created a program to draw simple shapes, my own pickle files to store numpy arrays, merge and load these arrays to prepare for the training of the main convolutional neural network file. I did all of this by myself without anybody helping me with the code. One of my friends just offered me advice on multiplying 255 by the float value for the Image module to get the correct background color of the image. My friend who does Tensorflow gave me a link for a tutorial on compressing image data using save from pickle. I had to use joblib to compress the data since it does a great job dumping huge lists of numpy image data arrays. That code is not copied, it is a reference within the save part, which is roughly five lines of code. In the main convolutional neural network file, I had to also use a tutorial on how to prepare the file to be executed in the cloud. The tutorial is in https://github.com/clintonreece/keras-cloud-ml-engine.

I learned that is extremely difficult to get the convolutional neural network to generalize well during training. The network is overfitted and underfitted, but that balance is hard to get a grasp of. This is because they are so many factors to consider, which are specifically convolutional neural network architectures, image input data, loss functions, activation functions, number of filters and neurons per layer. I also got a bit sidetracked by trying to do much in the project, classifying multi-output labels of intersections between shapes. I should use programs to analyze the image data, but the network is too complicated to get important information. This is because it is difficult to get the gradients of the hidden layers.