Neural Networks for Shape Recognition

Dalin Wang (04/25/2017)

1. Definition:

1.1 Project Overview

Training artificial Neural networks has become the de facto method for image recognition problem, which is a very important topic in the machine learning field as it can support lots of interesting applications in many fields including medicine, art and finance. Recent years have seen an explosion of proprietary/open source projects aimed at building a better machine learning platform, such as scikit-learn, tensor-flow, and Teano, to name a few [1]. And as a result, we have seen more and more applications of machine learning in our daily interaction with technology---Alexa voice assistant, Apple photos and self-driving cars. For the task of image recognition, researchers have demonstrated steady progress against ImageNet[2], a benchmark problem for computer vision, and each year at the ImageNet competition, we witness a new construction of neural networks become the state-of-art: QuocNet[3], AlexNet[4], Inception(GoogleLeNet) [5], BN-Inception-v2[6]. The ImageNet is a database of 1000 different classes of images, like "Leopard, container ship and motor scooter".

The paper is divided into 5 sections. Section 2 introduces the dataset and how we prepare and process the dataset. In section 3, I explained the three 3 steps of training a neural network with TensorFlow and the architecture of my CNN. Section 4 discusses the decision on hyperparameters like learning rate and batch size, and the different types of optimization methods. In section 5, I provided the results of my model and discussed some thoughts on aspects of potential improvements.

1.2 Problem Statement

The problem I am trying to solve is to distinguish between 3 basic shapes—rectangle, eclipse, and triangle—in 32 x 32 images with one shape and uniform colors (one for the shape, and one for the background). To accomplish this task, I developed a Convolution Neural Network(CNN) model using TensorFlow on the BabyAIShape Datasets[7]. The motivation for this study is for one, it is a simple task for beginners to learn about artificial neural networks; second, the model can be used in many interesting applications, e.g. an app for teaching children basic intuition of shapes in early education.

1.3 Metrics

The main metrics I am employing to measure the performance of the model is accuracy on the testing dataset, which is defined as $\frac{\Sigma\,\mathrm{TP} + \Sigma\,\mathrm{TN}}{total}$. This metrics is important because it is the measurement to gauge how likely our model is able to predict correctly. In this study, the model consistently achieves an accuracy of 0.995, which meets the expectation given by the small

number of datasets to train the model. I am also calculating precision and recall for each class to further measure the model. Precision measures how many predicted to be class A actually belong to that class A, and recall measures how many predicted to be class A are actually selected among all samples in class A.

2. Analysis

In this study, I have used publicly available dataset BabyAIShapeDatasets. There are several datasets in the link provided, but the particular dataset I have used is the shape2_1cspo_2_3 dataset group, which is composed of 10000 samples of training data, 5000 samples of validation data and 5000 samples of testing data. In the github repository[9], you can find the corresponding python files responsible for generating the *amat* file, which is the file format later used by the CNN model. To generate the amat files, you need to install python[10], and then in the terminal (assuming you are using Mac OS), simply type in the following command:

```
$ python shapeset2_1cspo_2_3.10000.train.py write_formats amat
```

2.1 Data Exploration

I also included the amat files inside the *input* folder for convenience. The *.amat* file is of ascii format. The value separator is the space (ascii code 0x20) It is organized as follows:

- The first line is the number of examples and the number of values per line (1031).
- On each subsequent line, the first 1024 values represent the gray tone of each pixel (or the color, depending on how you interpret it) as a floating point value between 0 and 1, inclusively (32 lines of 32 pixels). The next 7 values are:
- The shape: 0=rectangle, 1=ellipse and 2=triangle
- The color of the shape: this is actually an integer between 0 and 7. Divide by 7 to get the corresponding gray tone.
- The x coordinate of the centroid of the shape, between 0 (leftmost) and 256 (rightmost).
- The y coordinate of the centroid of the shape, between 0 (top) and 256 (bottom).
- The rotation angle of the shape, between 0 (no rotation) and 256 (full circle). This can probably not be learnt reliably because there the reference point is ambiguous (for instance, there is currently no way to know relatively to which side the triangle was rotated).
- The size of the shape, between 0 (a point) and 256 (the whole area). There is a lower bound and an upper bound.
- The elongation of the shape, between 0 (at least twice as wide as tall) and 256 (at least twice as tall as wide).

I gathered some statistics for our three original datasets: training, validation and testing

```
Shape of the original dataframe: (10000, 1031)
training dataset ./input/shapeset2_1cspo_2_3.10000.train.amat has the following characteristics:
shape centroid average: [ 127.6664  127.4526]
shape rotation average: 62.6993
shape size average: 52.3796
shape elogation average 128.7348


Shape of the original dataframe: (5000, 1031)
validation dataset ./input/shapeset2_1cspo_2_3.5000.valid.amat has the following characteristics:
shape centroid average: [ 127.4262  127.4868]
shape rotation average: 63.4374
shape size average: 52.1378
shape elogation average 128.973


Shape of the original dataframe: (5000, 1031)
testing dataset ./input/shapeset2_1cspo_2_3.5000.test.amat has the following characteristics:
shape centroid average: [ 127.185   127.4044]
shape rotation average: 62.5304
shape size average: 52.3488
shape elogation average 128.2326
```

Figure 1 Dataset statistics

The coordinates of the centroid range from 0(leftmost) to 256(rightmost). As you can see, the average centroid for the shapes are in the middle of the image. As for the rotation, the number also ranges from 0(no rotation) to 256(full circle), so the average rotation for the images is about 88 degrees clockwise. The number for size also ranges from 0 to 256(full size), and the average area of the shape occupies about 20% of the image. The elongation of the shape is also between 0(the shape at least twice as wide as tall) and 256 (the shape at least twice as tall as wide), and the average is right in the middle so the average shape is almost as wide as tall.

2.2 Exploratory Visualization

To visualize the data, you can simply type in the following commands:

```
$ python shapeset2_1cspo_2_3.10000.train.py view
```

Figure 1 shows some sample images used for training and testing. As you can see, the colors do not provide info about what the shapes are, and these samples provide some intuition into how we will preprocess our data to reduce the amount of info our model needs to learn.

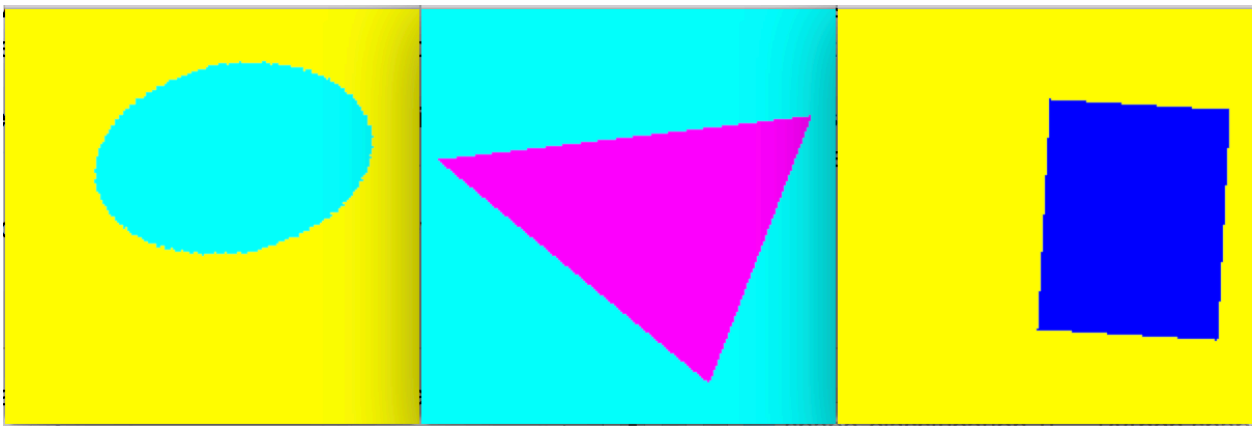

Figure 2 From left to right: eclipse, triangle, rectangle

2.3 Algorithms and techniques (Architecture of CNN)

In this study, we will use Convolutional neural networks(CNN) as our fundamental approach. In general, Convolutional Neural networks(CNN) is a feed-forward artificial neural network represented by a stack of distinct layers that transform input volume into output volume with class scores through a differentiable function.

Though the architecture of CNN is well researched and there is a pattern on how we can implement it, there are still many hyper-parameters we need to decide for our CNN networks. The first one is the number of filters we want to use for convolutional layer. The filter controls the number of output channels we will have. The second is the learning rate and types of optimizer we want to use for gradient descent. Then, we also need to decide the batch size. Last, the number of the hidden layers and the number of convolutional layers.

As you can see from figure 3, we use three main types of layers to build our CNN model: Convolutional Layer, Pooling Layer and Fully-Connected Layer. Specifically, our CNN model consists of two layers of convolutional layer, each followed by a max-pooling layer. The output of the second pooling layer is passed in a fully connected hidden layer, which is followed by an output layer, which is also a fully connected layer. Notice there is also a dropout layer in between the hidden layer and the output layer.

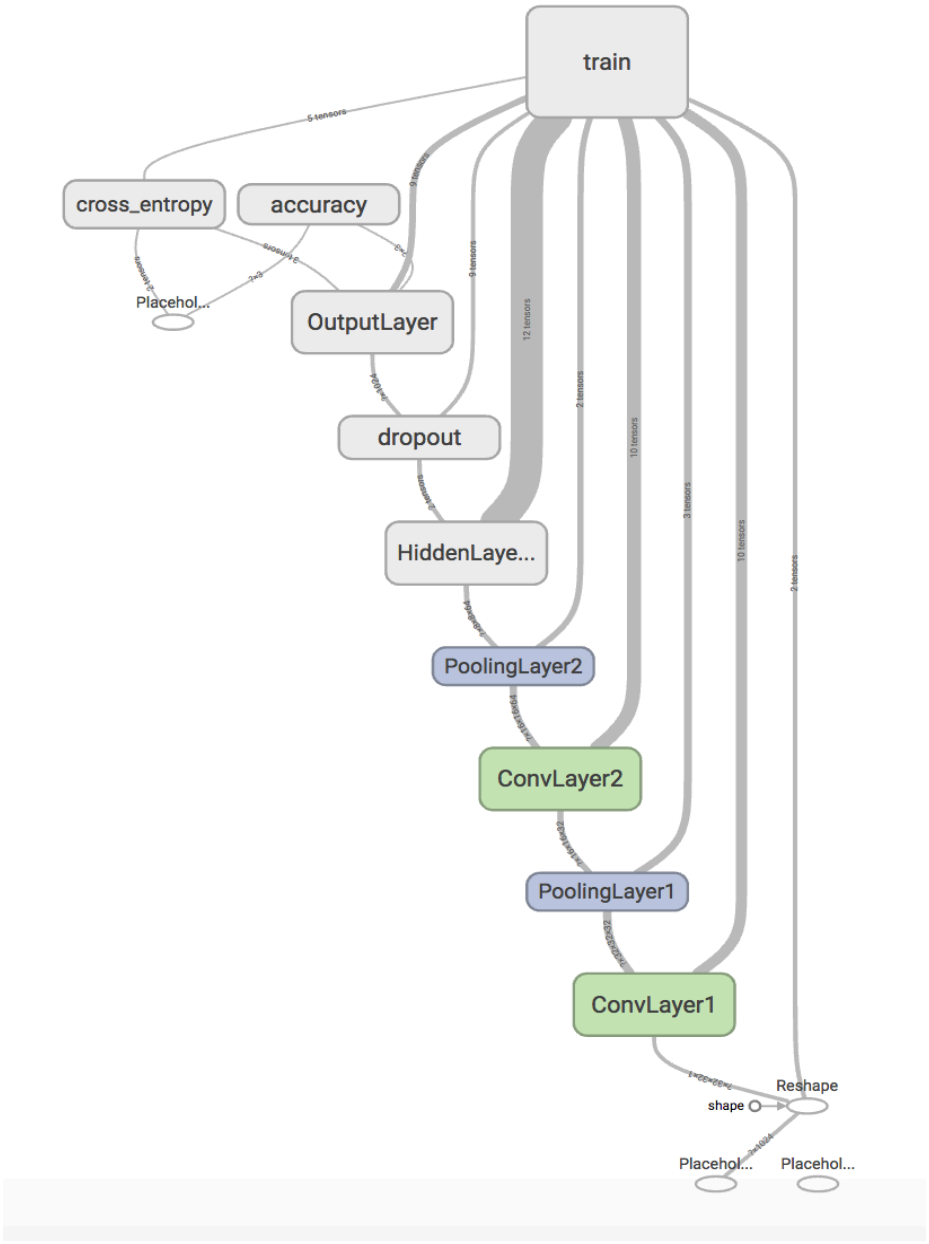

Figure 3 Tensor Graph generated with Tensor board

The convolutional layer is at the forefront of CNN model. It is responsible for creating a model invariant of the shapes' orientations and positions in the image. Recall that the input of the convolutional layer is a batch of [32 x 32 x 1] images of shapes with a unified shape color and background, the default batch size is 50. The convolutional layer will compute the output of the neurons that are connected to local regions in the input by passing the local region of pixels through a filter. Figure 4 is an image I found online that can demonstrate how the convolutional layer filter interacts with the input. The input image here is of size [7x7x3], and the filter is [3x3x3] and they have applied 2 distinct filters to the input volume and the filter stride is 2. In contrast, in my CNN model,

my first convolutional layer used filter of size [5x5] with one input channel and 32 output channels with 0 padding and stride of 1; my second convolutional layer used the same filter size of [5x5] with 32 input channels and 64 output channels with 0 padding and stride of 1.

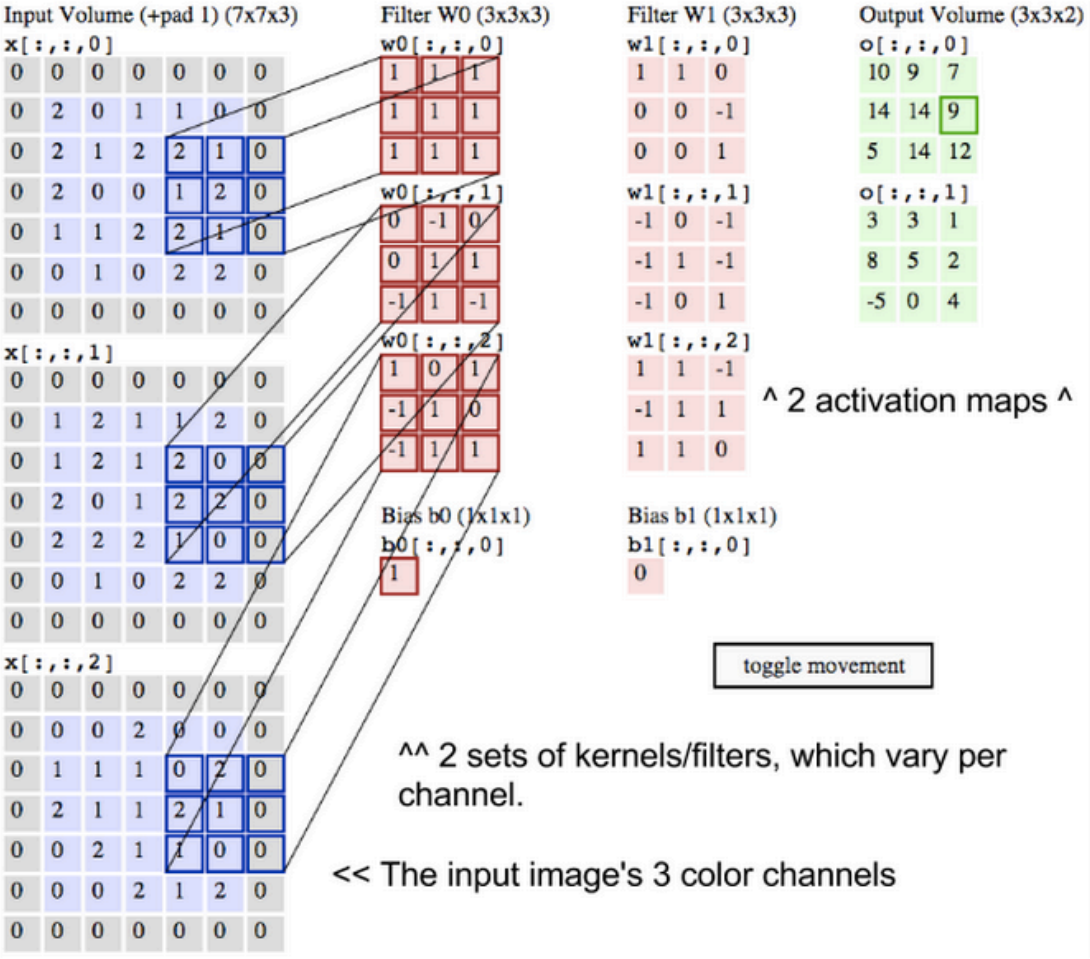

Figure 4 Credit for this excellent animation goes to Andrej Karpathy

Next layer is max-pooling layer. When applied, the max-pooling layer takes the largest value from a block of numbers in the input matrix, and places it to a new matrix next to other max values from other blocks. Lesser information is lost with this operation, however, it has the advantage of avoiding over-fitting and reducing processing time and storage.

The dropout layer is a regularization technique used for reducing over-fiting in neural networks by randomly dropping out some activations. In the model, I chose a dropout rate of 0.8 in the training session.

Last type of layer is fully-connected layer, which just means it has full connections to all activations in the last layer. The hidden layer is responsible for providing more non-linearity and makes it possible for representing more complex functions. The activations in fully-connected layer is only computed with matrix multiplication with a bias offset. As in the case of our model,

we have two fully-connected layers, the hidden layer and the output layer. It's worth noting that the only difference between the fully-connected layer and the convolutional layer is the weights in the convolutional layer is connected to local regions whereas the fully-connected layer's weights are committed to all parameters in the input matrix. As a result, the weights dimension for our hidden layer is [8x8x64, 1024], which is connected to the activations after the second pooling layer which downs the image to [8x8] with 64 feature maps for each parameter in the input matrix. Our output matrix has weights of size [1024, 3], which maps the 1024 features input 3 classification classes.

There are many commonly used activation functions in machine learning. In this project, we chose the ReLU(Rectified Linear Unit) which computes the function f(x) = max(0, x). In other words, the activation function simply thresholds the input at 0. The advantage of activation is that it was found that "it greatly accelerates the convergence of stochastic gradient descent compared to sigmoid/tanh functions due to its linear and non-saturating form"[8].

Last, I used softmax classifier with Adam optimizer for my model:

The softmax classifier is of the form:

$$L_i = -\log\left(\frac{e^{f_{yi}}}{\Sigma_j e^{f_j}}\right)$$

The softmax classifier normalized the class scores to class probabilities that sum to 1 and then applies the results to a cross entropy loss function. Therefore, our model is essentially a softmax classifier that tries to minimize the cross-entropy between true class distributions and estimated class distributions.

Last, I want to introduce the gradient descent method I used for my project: "Adam Optimizer (Adaptive Moment Estimation). Adam is a method that computes the adaptive learning rate for each parameter. It is found that Adam optimizer works well in practice and compares favorably among other optimizers." [9]

2.4 Benchmark

In this study, we used the CNN with one convolutional layer and one pooling layer and one fully connected output layer as our benchmark since it is the simplest form of CNN. The benchmark model is detailed in the file shape3_benchmark.py, and it achieved an accuracy of 0.691 in testing dataset, with a batch size of 50 and with 6000 iterations of training.

3. Methodology

3.1 Preprocessing the data

As we can see from the data exploration section, the color does not help with recognition of shapes, and this noise can be eliminated with some preprocessing. To preprocess the data, I first normalized the data since the color is merely a distraction when training a model to recognize shapes. By iterating over the normalize function, the pixels belong to the shape would have a value of 0.5, and the pixels of the background would have a value of -0.5. Since the training and validation data only has 15000 data points, it is too few to train an accurate model. To solve this problem, I flipped the image along the x-axis and y-axis and then translated the image to the left, right, top and bottom by 3 pixels to gain 24000 more sample data. At the end, I have a total of 30000 training and validation data combined for training.

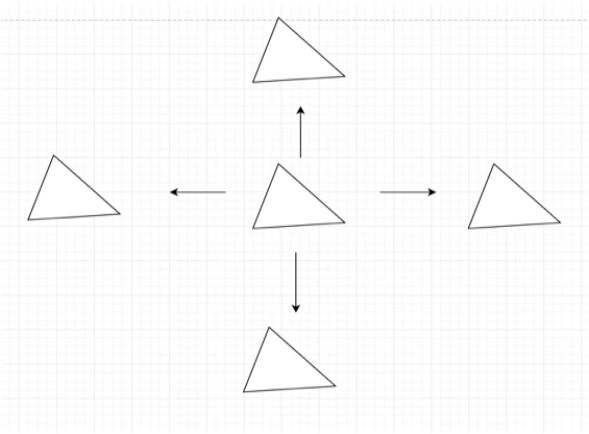

Figure 5Populate The Shape Dataset with translation operations

3.2 Implementation

The algorithms used here are documented in the algorithms section. The implementation of this project is done in python with the tensorflow framework. The central unit of data in Tensorflow is the tensor, which consists of a set of primitive values shaped into an array of any number of dimensions. There are general two steps of implementing the program, first, you need to build the tensor graph, and then run the graph in a session. The tenosorflow API provides different optimizers, like the Adam optimizer we talked about in the algorithms section. The optimizer generally modifies the weights in each tensor according to the magnitude of the derivative of loss with respect to the weights. The complications I have when coding is to find a better way to modularize each layer of tensors, for example, convolutional layer, and fully connected layer.

3.3 Refinement

From benchmark model to our final model, I made several refinements. First, we added one more fully connected layer and one more convolutional/pooling layers' combination. This is because that the benchmark model clearly cannot model the complexity of this problem as it reaches a plateau around 0.65 of training accuracy and 0.7 of cross-entropy. I have also tried with two hidden layers, but it clearly over fitted with a lower testing accuracy 0.694.

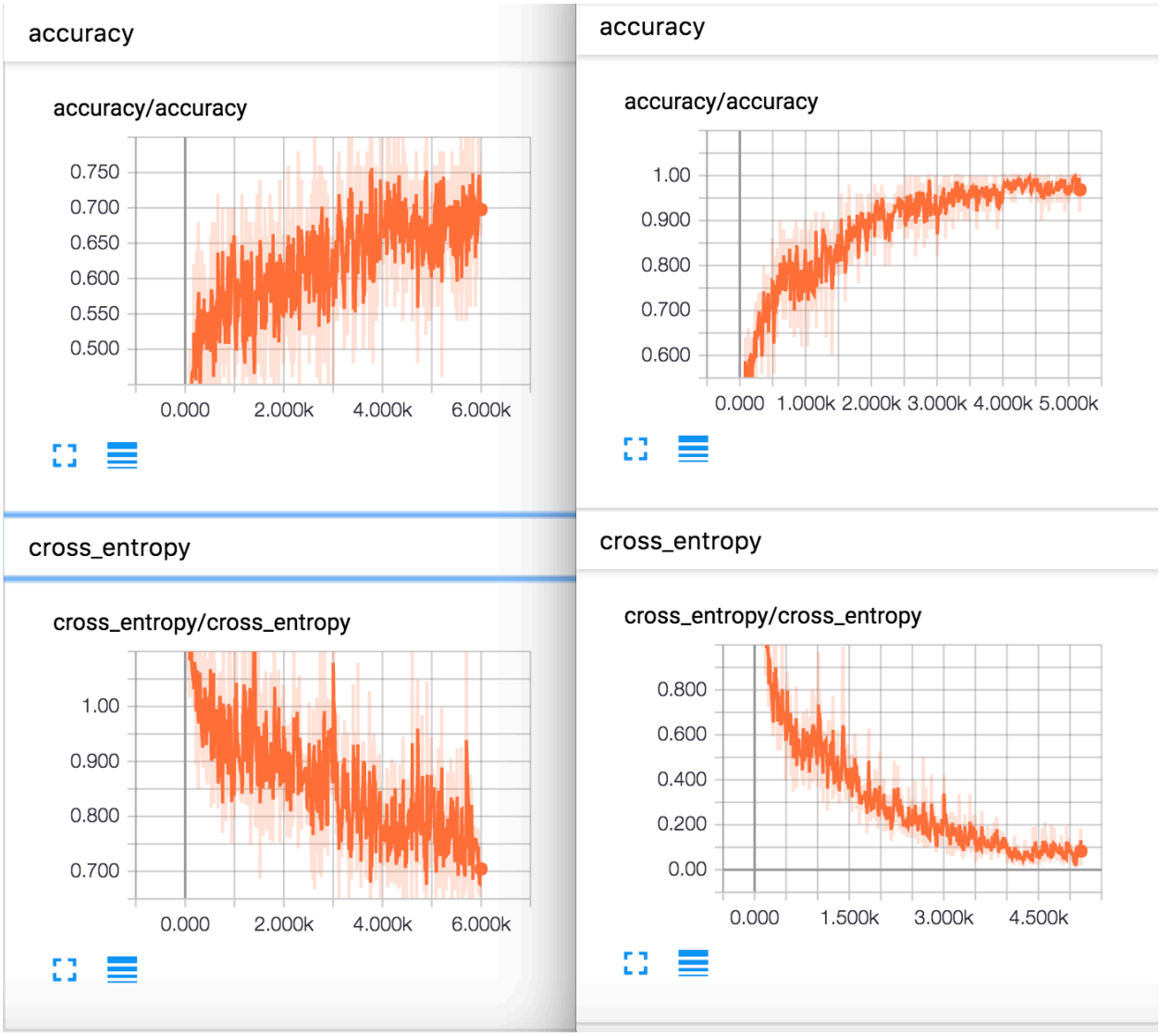

*Figure 6 Left: benchmark model; Right: final model**; The x axis is the number of trials; The y axis is accuracy and cross_entropy respectively***

I have also tried different training rate from 0.1 to $1e^{-4}$, and I found $1e^{-4}$ works best as the other magnitudes often over adjusts the gradients during training.

Last, I experimented with different batch size and found 50 and 100 performs all fairly similarly, however, the constraints on CPU usage is high (about 90% CPU usage at peak) when the batch size is bigger, and so I recommend using 50 so it can train the model in a fairly short time about 15 mins yet not putting too much pressure on CPU.

4. Results and discussions

4.1 Model Evaluation and Validation

After training on training and validation datasets for 6000 iterations on batches of 50 samples per iteration, I ran the model on test dataset of 5000 shape images, and achieved an accuracy of 0.9941on the final model. Here is a snapshot of the accuracy on the testing dataset from tensorboard. The final model is reasonable and aligns with my expectation based on the metrics. The model's robustness is tested with separate testing dataset and can be trusted to use.

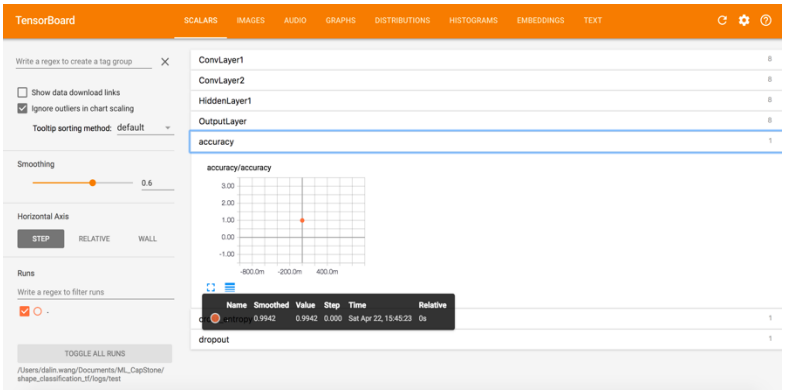

Figure 7 Accuracy on testing dataset

```
confusion_matrix:
                        rectangle ellipse triangle
predicted_rectangle: [[1624         1         1]
predicted_ellipse:    [   6      1771         1]
predicted_triangle:   [   0         3      1593]]
rectangle_precision: 0.9987699877
rectangle_recall: 0.996319018405
ellipse_precision: 0.996062992126
ellipse_recall: 0.997746478873
triangle_precision: 0.998120300752
triangle_recall: 0.998746081505
```

Figure 8 Confusion matrix and precision and recall of each class on testing dataset

As we can see from the statistics of the precision and recall for each class on our testing dataset, our model is very robust regarding to selecting the relevant items for each class. However, it is worth noting that the recall for rectangle is relatively slow with 6 misclassifications where it is predicted as ellipse but actually is rectangle.

4.2 Justification

Compared to the benchmark model, our final model greatly improves the model prediction accuracy on testing dataset by over 30%, from 0.691 to 0.994. Also, the precision and recall for each class is also very high around 0.996 – 0.998 depending on different classes.

5. Conclusion

5.1 Reflection

In this study, I have learned the architecture and methodology of building a CNN, and apply the CNN model to a specific domain. The most difficult part of the project is to have the intuition of preprocessing the data to filter out the color component and populate the dataset by translating the images to different directions. This results in a much bigger dataset than the original dataset and is proved critical in the success of the model. The architecture part is also challenging after I realized that I can make each layer a function and reuse it for with function calls to each layer. This greatly reduces the code length and makes it easier to understand and read. Last, experimentation with the different hyper-parameters is also rewarding as I learned that the choice of learning rate would influence the result a lot, and you cannot just add more depth by adding more number of hidden layers blindly as it would ultimately result in over-fitting your model.

5.2 Future Improvement

For future improvement, we can try experiment with other optimizers and with different size filters for the convolution layers. We can also add explicit rules from our human understanding to aid the algorithm, and it would reduce the training time by a lot.

References:

1. Top 20 Python Machine Learning Open Source Projects in 2016: (*http://www.kdnuggets.com/2016/11/top-20-python-machine-learning-open-source-updated.html*)
2. ImageNet: (http://www.image-net.org/)
3. QuocNet: (http://static.googleusercontent.com/media/research.google.com/en//archive/unsupervised_icml2012.pdf)
4. AlexNet: (http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf)
5. GoogleLeNet: (https://arxiv.org/abs/1409.4842)
6. BN-Inception-v2: (https://arxiv.org/abs/1502.03167)
   Gradient Descent: (*http://cs231n.github.io/optimization-1/*)
7. BabyAIShapeDatasets: (http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/BabyAIShapesDatasets)
8. ImageNet Classifcation: (http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf)
9. Adam Optimizer: (http://sebastianruder.com/optimizing-gradient-descent/index.html#adam)