Hand signs recognition with Convolutional Networks using TensorFlow

Trung Vo
CSE 327 Fall 2017 - Final Project
trung.vo@stonybrook.edu

1. Introduction

Convolutional Neural Networks are similar to the ordinary fully connected Neural Networks, which are constructed by neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the networks.

In this project, a deep convolutional network will be trained to classify 6 categories of hand gestures (0 to 5). So, there will be 5 classes in the last fully connected layer of the network. At the high level, each training image is fed into the network at the first layer, then goes to convolution layer followed by a ReLU activation map, then pooling layer, and so on. The network keeps going deeper till the last pooling layer, then all the hidden units in that layer are flattened out into a single 1D vector that is used to feed into a series of fully connected layers. The softmax function is then applied at the final layer of the network to produce the probabilities of how close the input image is compared to the ground truth or expected output.

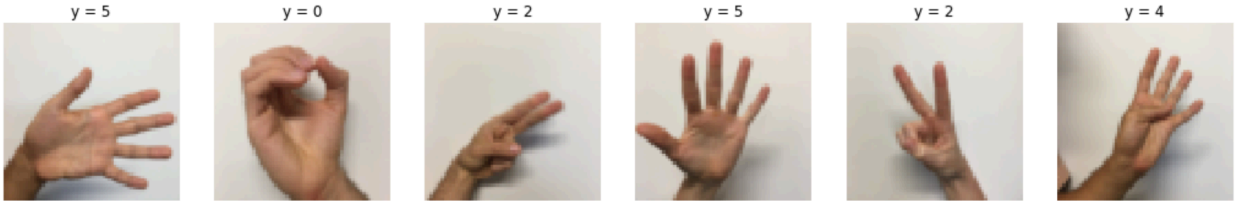
In this project, we will train the network using TensorFlow, a well-known deep learning open source framework for numerical computation using data flow graphs. One of the advantages of TensorFlow is the incorporation of Tensorboard, a friendly interface that is used to visualize our network model as a computational graph, as well as allows the training process become easier with the built-in functionality for visualizing learning curve of loss and accuracy.

2. Dataset Collection and Preprocessing

The dataset includes 1080 training RGB images and 120 testing RGB images of size 64x64x3, with training labels of 6 categories (0 to 5). This dataset can be found at Convolutional Neural Networks Coursera programming assignment 1.

Before training our model, it is important to preprocess the dataset by normalizing our images. Typically, images data will be normalized to zero mean and unit length. To perform this, each image is subtracted by the mean of the training set, this ensures the data to be zero-centered at the origin, which makes training process much easier when perform gradient descent. After that, they are normalized into range between 0 and 1 by dividing each image by 255. This ensures all the input pixel values of the images to be in the same range. Both of these preprocessing steps are important and helps gradient descent converge towards local minima more quickly.

A few images are displayed with their labels as below:



3. Convolutional Network model

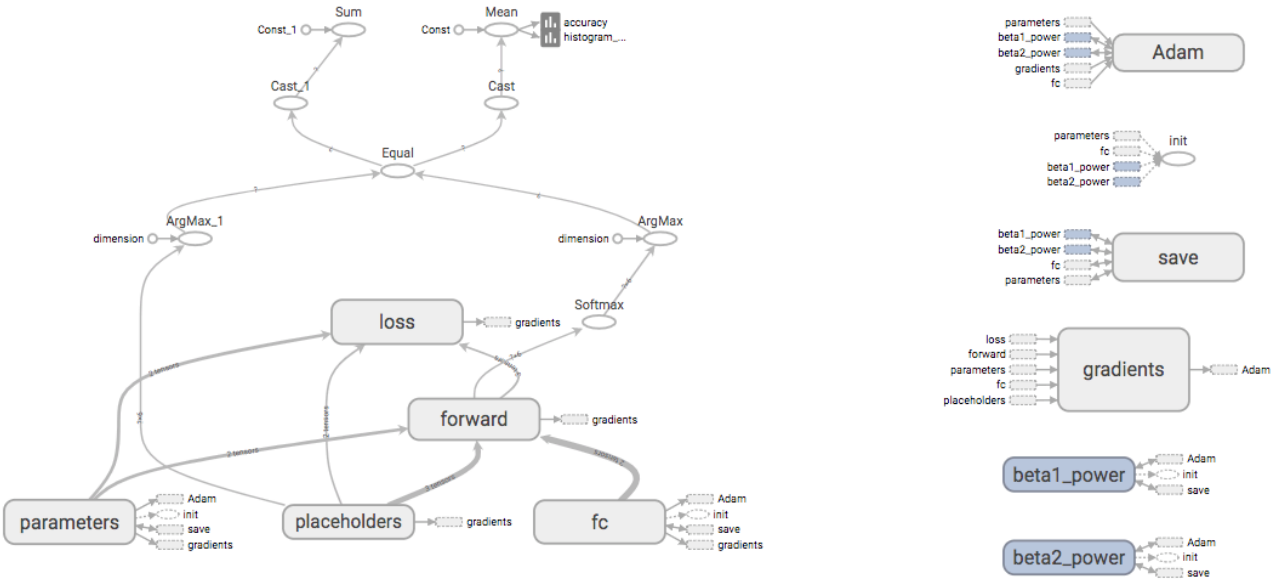
In this project, the CNN model is constructed manually, architected as follow:

Conv1 – ReLU1 – Pool1 – Conv2 – ReLU2 – Pool2 – Flatten – Dropout – FC

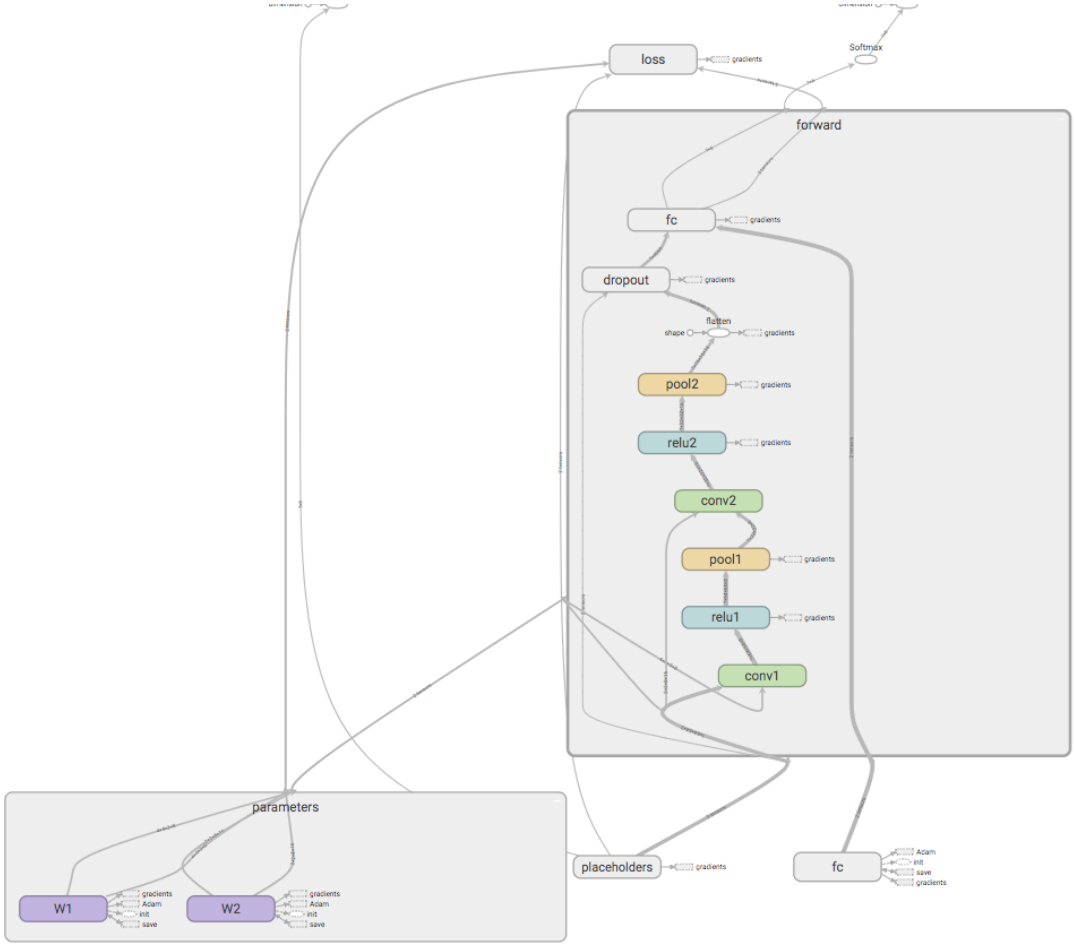
This is similar to the general form of the CNN model.

In particular, input image with size of 64x64x3 will be fed into the first Convolution layer with parameter W1 of size 4x4x3, 0 padding, stride 1 and 8 filters applied. Then the output goes to ReLU activation map. This use of activation is commonly used and produces good result than other activation functions, that is ReLU prevents gradient being "killed off" when network saturates in positive region because of its linearity. Then the output of this activation is fed into the Max Pooling layer with filter size of 2x2, stride 2, which downsamples the volume spatially, independently in each depth slice of the input volume. This helps reduce the spatial size and the number of parameters and computation in the network, and hence controls overfitting as well. Then output from this Pooling is then fed to the series of Conv (W2 = [2,2,8,16]) – ReLU- Pool (filter size 2, stride 2) again. The output of Pool2 is flattened into 1D vector of size 16*16*16 = 4096, fed through a dropout layer and then through the final fully connected layer of output size of 6 (number of classes). This last layer will then later be applied softmax function to produce probabilities of how close the predicted image is to the ground truth image.

By using Tensorflow, which is powerful in generating computational graph for our model, we can easily see our data flow wires up correctly before we start training. Below is our network model visualized in Tensorboard.



Overview of model computational graph



Expand nodes to examine deeper level of our model

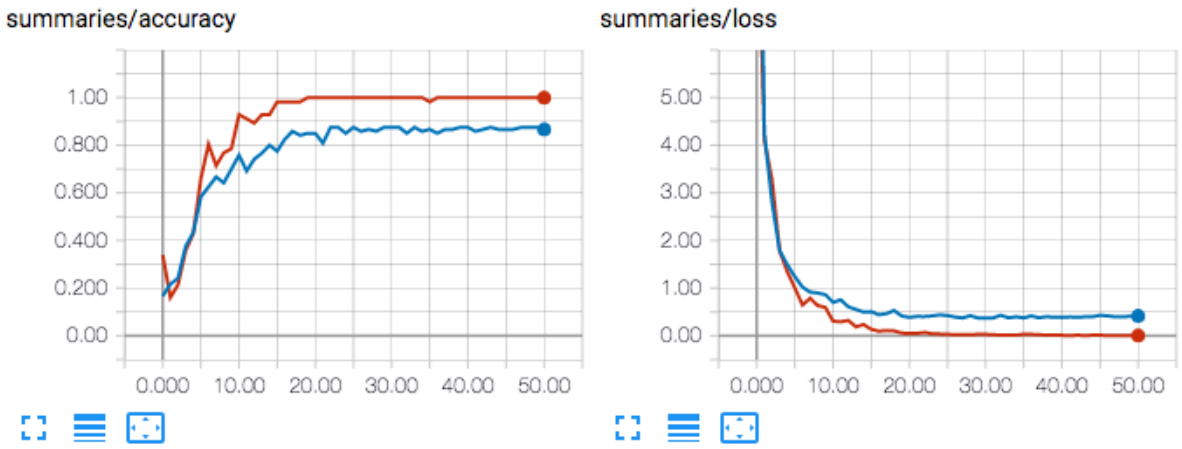
4. Weight Initialization

Before training our network, it's important to initialize our weight parameters to prevent vanishing and exploding gradients from happening. It's because if the weights are not carefully initialized, for example, if they're extremely small, as the network gets deeper, the activation will decrease exponentially and hence the gradient with respect to weights will also decrease exponentially in terms of number of layers. This causes the network to converge very slowly, as weights take many tiny little steps to reach the local minima. This is called vanishing gradients, exploding gradients can happen similarly when the weights are too large. To properly initialize the weights, we can apply Xavier initialization.

5. Training and Evaluating Model

Training model involves breaking the training set into multiple mini batches and train each batch on each iteration. This is called Minibatch Gradient Descent, which is more efficient than Batch Gradient Descent and Stochastic Gradient Descent where all parameters are updated for a subset of the training set. In this project, we use Adam optimizer to train our model. We first train the network with the learning rate of 0.001, 64 mini batches size, 51 epochs. The result is as follow.

Overfitting:

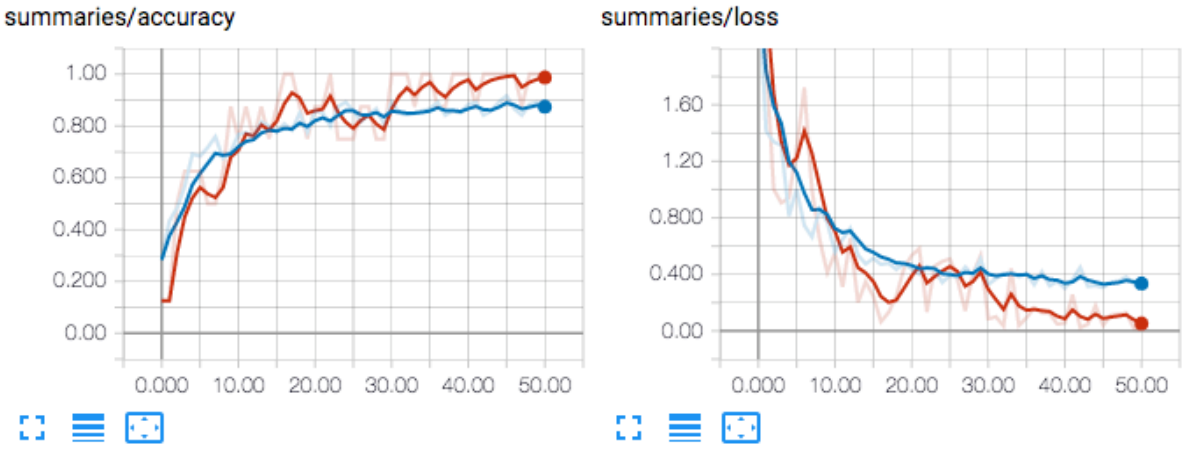


The training curve is orange and test curve is blue. As we obtain from the learning curve, this is overfitting (high variance). The highest training accuracy is 100 % while the highest testing accuracy is only 87.5 %. Therefore, we can use several regularization techniques to improve our model performance, such as data augmentation, dropout, L1/L2 regularization.

- **Data augmentation**

Overfitting can happen because our model doesn't have enough training data. Data augmentation takes each training image and perform several transformations such as random cropping, rotation, horizontal and vertical flip, and so on, to increase training data.

After performing data augmentation, we start training our model again and obtain that the testing accuracy increases to 91.67 %, which is slightly better than previous result. Below are learning curves for training model with data augmentation.



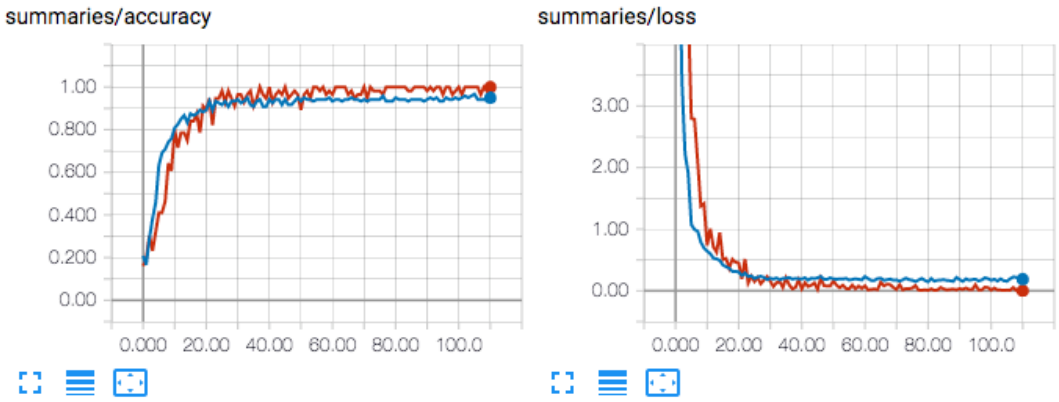
Since data augmentation increases the size of training data, it takes longer to train the model, approximately 18 minutes, with 51 epochs.

- **Dropout regularization**

Another technique we can use to prevent overfitting is adding dropout layer before the last fully connected layer. Here dropout layer is added with a specified keeping probability of 0.6, which essentially keeps 60 % of random neural units of our hidden layer unchanged and turns off the remaining 40 % of units. Dropout is usually added between fully connected layers because these layers occupies most of the parameters, and hence neurons develop co-dependency among each other during training which curbs the individual power of each neuron leading to overfitting data.

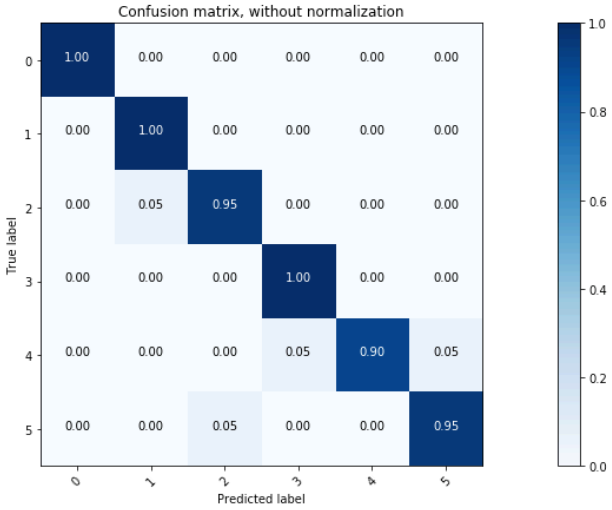
As a result, dropout with keeping probability of 0.6 results in a much better performance. Testing accuracy is 96.67 %, which is much higher than data augmentation approach. Since we randomly turn off some hidden units in the network, it takes more iterations for our gradient descent to converge. Here the training takes approximately 5 minutes with 111 epochs.

Result:

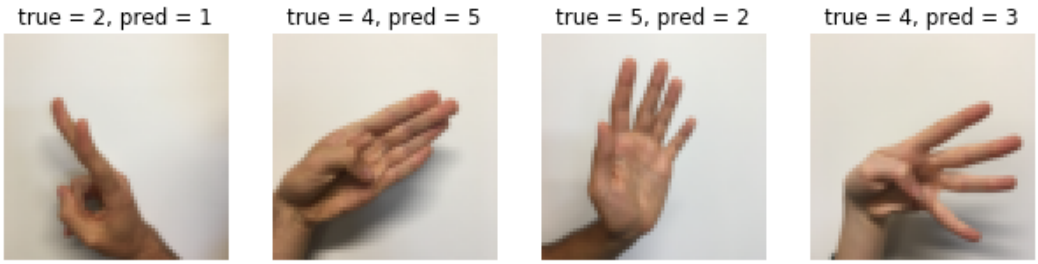


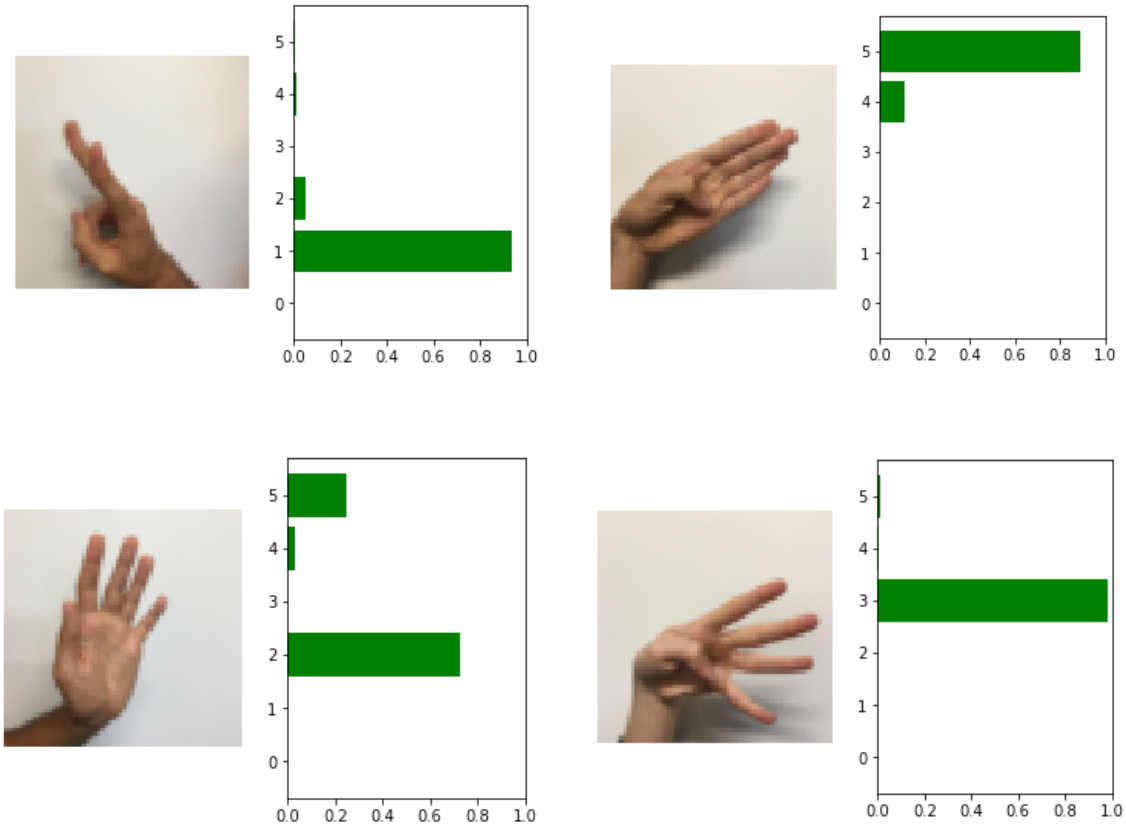
As obtained, the gap between the training and testing curve gets smaller, which results in the best performance for our model of 96.67 % accuracy.

We can evaluate our model performance by using confusion matrix, which essentially tells us the accuracy our model predicts for each class. As obtained, our model predicts very well for class 0, 1, 3. Class 2, 4, 5 are a little off, perhaps because the model might be wrong predicting 2 to be 1 and 4 to be 5 and vice versa.



Display false predictions:





Some of the possible causes for the wrong predictions might be because of the shadow of the hand that makes the training model to hardly recognize correctly. Another cause might be the pose of the hand that makes the model hardly realize the correct prediction, for example in first and second picture.

6. Conclusion

In conclusion, our model performs best by adding dropout layer before the last fully connected layer. This randomly turns off some of the neural units in the hidden layers based on the amount of keeping probability being specified. This technique is common and easy to use in Computer Vision and hence, usually yields the good performance. Furthermore, L1/L2 regularization can also be used to prevent overfitting, in which the loss function consists one extra regularization term, which is the mean of squares of all the weights in the network times lambda (a tunable parameter that controls our model from overfitting). As lambda gets higher, the weights become smaller and make the network become more simple and hence, helps the network become high bias and overcomes overfitting. Data augmentation is also easy to use. However, since the number of training data increases, it might take longer time to train, which is one of the drawbacks of this approach.