Traffic Sign Recognition

Writeup

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

This is the writeup that contains (hopefully) all the rubric points. The project code will be included in the compressed archive in both the ipynb format and HTML format.

Data Set Summary & Exploration

1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

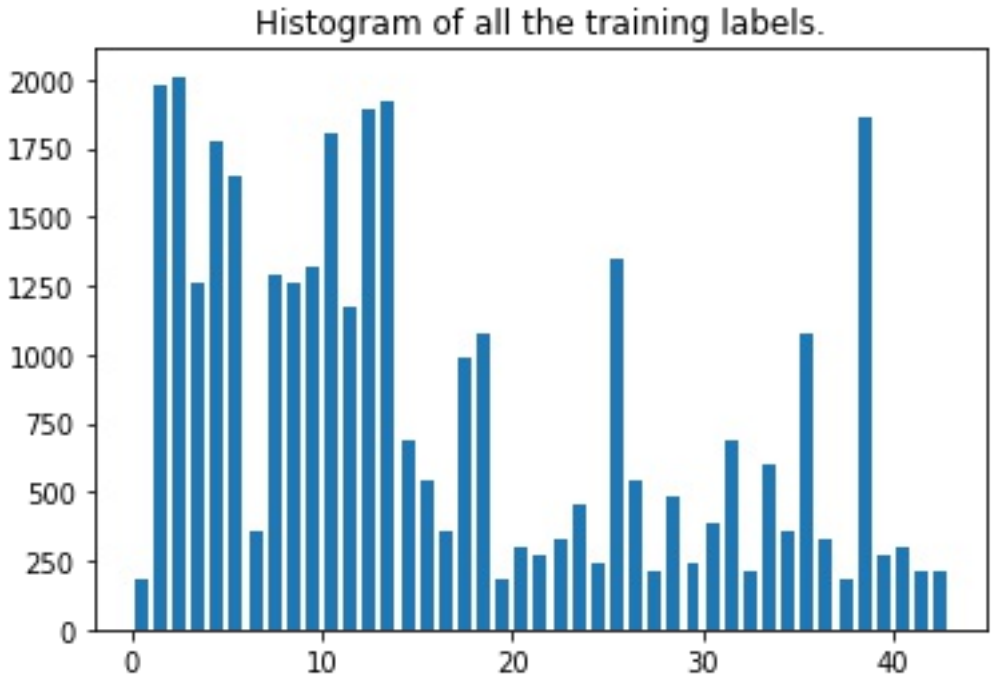
I used the pandas library to calculate summary statistics of the traffic
signs data set:

Before any sort of augmentation:

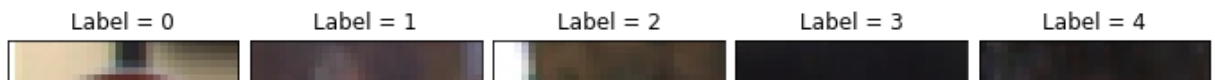
- The size of training set is 34799
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43

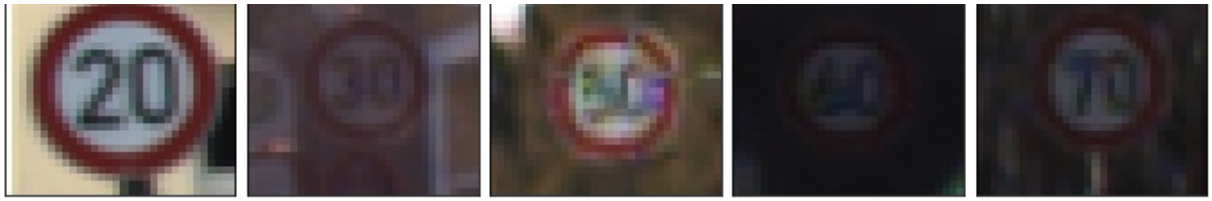
Once augmentation was performed, the validation set seemed a little too small compared to the training set. Hence I decided to splice some off the training set.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data ...



Sample visualization of all the images:





Label = 5 Label = 6 Label = 7 Label = 8 Label = 9

Label = 10 Label = 11 Label = 12 Label = 13 Label = 14

Label = 15 Label = 16 Label = 17 Label = 18 Label = 19

Label = 20 Label = 21 Label = 22 Label = 23 Label = 24

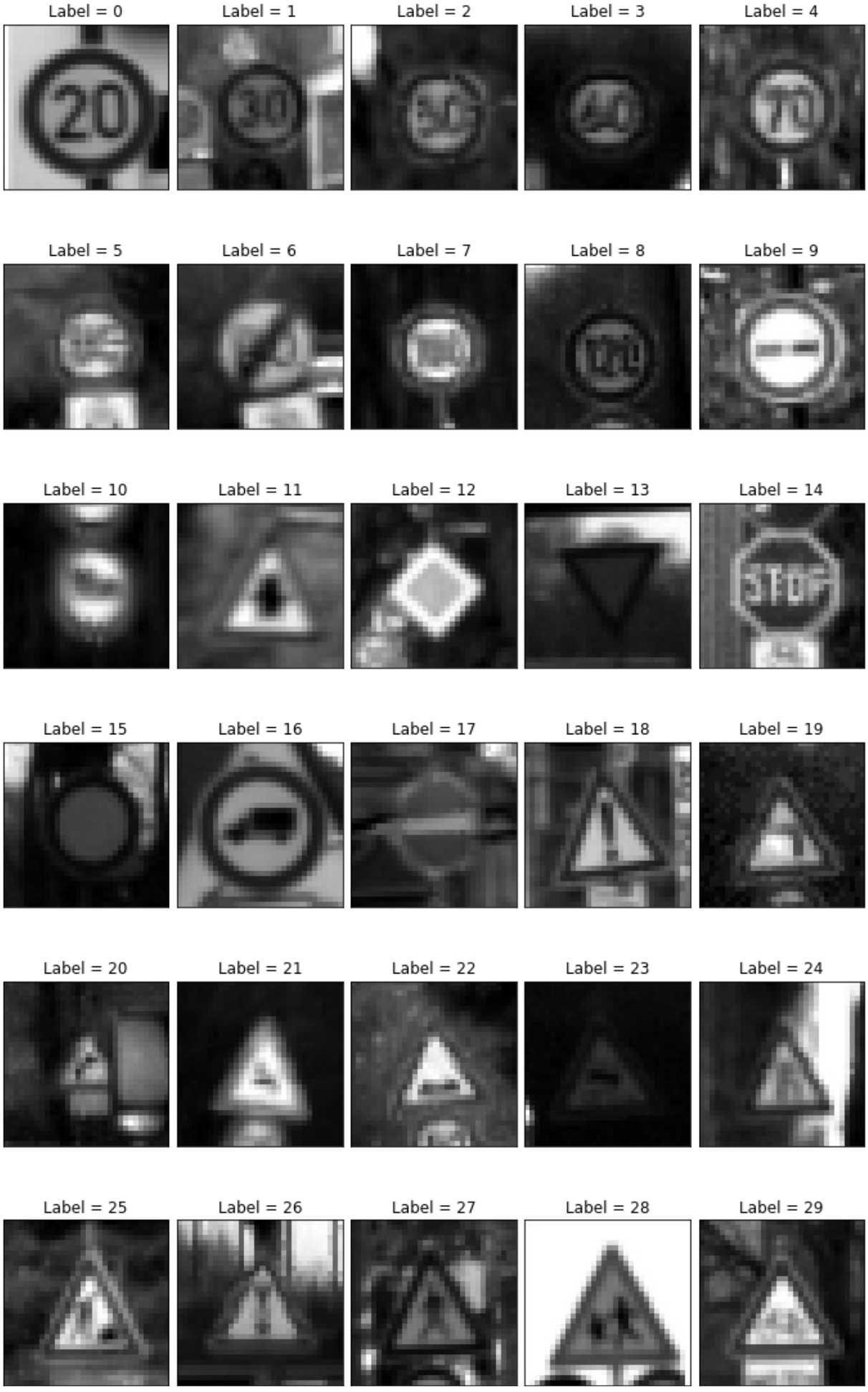
Label = 25 Label = 26 Label = 27 Label = 28 Label = 29

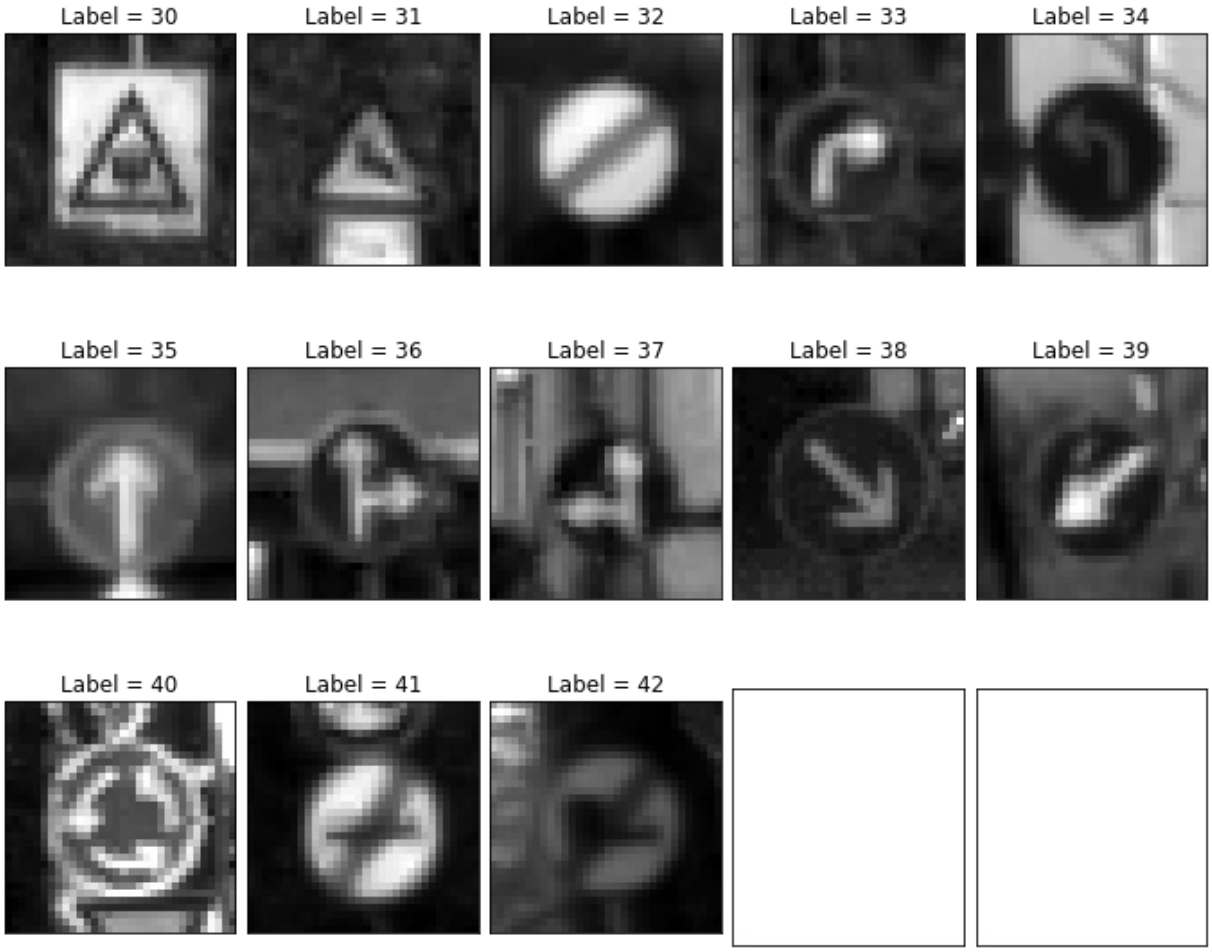
Design and Test a Model Architecture

1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

As the starting step, I first tried to perform normalization on the given data directly and tried to visualize them. For some reason, I was getting a weird output when I tried to visualize the image. Thus, I decided to first apply a grayscale and then perform the suggested normalization.

Sample output:


Label = 0
Label = 1
Label = 2
Label = 3
Label = 4
Label = 5
Label = 6
Label = 7
Label = 8
Label = 9
Label = 10
Label = 11
Label = 12
Label = 13
Label = 14
Label = 15
Label = 16
Label = 17
Label = 18
Label = 19
Label = 20
Label = 21
Label = 22
Label = 23
Label = 24
Label = 25
Label = 26
Label = 27
Label = 28
Label = 29



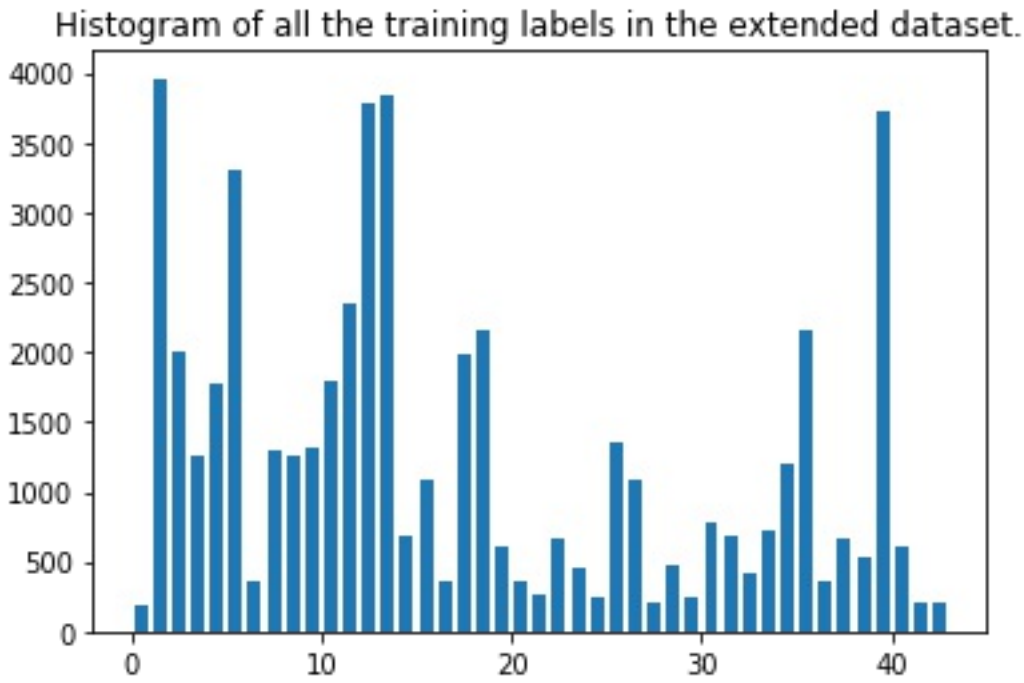
After performing normalization on the above gray scale images, the mean was roughly 0.36:

```
Original mean = 82.6775890369964
Grayscale mean = -0.36004453855662694
```

At first, for the grayscale image conversion, I performed a naive Average mean conversion. This,while at start seemed to be good enough, led me to an alternative luminosity method. Upon performing this type of GrayScale conversion, I was able to get comparitively better results on my training.

After tinkering around with my neural network for quite sometime, I decided to perform some sort of data augmentation to get more training data for the neural network. But, upon running several variations of the neural net, I realized that I was overfitting the results by quite some amount.

The data that I had after I performed my initial round of augmentations:



Once I had plotted this and realized that I was training the neural network incorrectly (after spending a considerable number of hours on it), I realized no amount of modifying the neural net was going to help.

To test out the theory, I removed the augmentation and tested with the exact previous run. The outputs below show that I was indeed performing poorly by over-fitting the data a lot.

Trial 27

```
BatchSize = 128
Epochs = 200
rate = 0.0009
mu = 0
sigma = 0.1
```

EPOCH 200 … Validation Accuracy = 0.850. Took 427.9920265289984 seconds to complete

Trial 28

Without the augmented data

```
BatchSize = 128
Epochs = 200
rate = 0.0009
mu = 0
sigma = 0.1
```

EPOCH 200 ... Validation Accuracy = 0.952. Took 289.76983048200054 seconds to complete

Finally, I augmented the data by performing the flip and then adding fluff data by performing random rotation and translation till a set limit (2000 images).

2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 GrayScale image
Convolution 5x5	1x1 stride, SAME padding, outputs 32x32x32
RELU	
Max pooling	2x2 stride, outputs 16x16x32
DropOut	keep_prob = 0.9
Convolution 5x5	1x1 stride, SAME padding, outputs 16x16x64
RELU	
Max pooling	2x2 stride, outputs 8x8x64
DropOut	keep_prob = 0.8
Convolution 5x5	1x1 stride, SAME padding, outputs 8x8x128
RELU	
Max pooling	2x2 stride, outputs 4x4x128
DropOut	keep_prob = 0.7
Flatten	Neurons = 2048
Fully connected	2048 -> 1024
DropOut	keep_prob = 0.5

Fully connected	1024 -> 512
Fully connected	512 -> 43
Softmax	Output

3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

To train the model, I used an Adam optimizer. This was from the LeNet lesson.

The final settings that I used for the hyperparameters are:

- BatchSize: 128
- Epochs: 200
- Learning Rate: 0.0009
- mu: 0
- sigma: 0.1

For the dropout parameters:

- Layer 1: Keep = 0.9
- Layer 2: Keep = 0.8
- Layer 3: Keep = 0.7
- Fully Connected 1->2: Keep = 0.5

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

My final model results were:

- training set accuracy of ?
- validation set accuracy of ?

- test set accuracy of ?

If an iterative approach was chosen:

- **What was the first architecture that was tried and why was it chosen?**

The first architecture that I chose was the default LeNet architecture that was discussed in the class/lecture. This seemed as a good starting point since the project said that the LeNet architecture should give a validation accuracy of 0.89 if it was built/used as such.

- **What were some problems with the initial architecture?**

Clearly, the validation accuracy is quite low. The architecture was built for simple black on white numbers recognition. It was not built for traffic image sign classification. But, it still serves as a good starting point from which I updated and modified "MyNet" to get a somewhat better solution.

- **How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to overfitting or underfitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.**

The architecure was adjusted in several ways. I moved back and forth between adding more layers and reducing the fully connected layers. I tested both using the VALID padding and the SAME padding provided by the tensorflow conv2d method. Both provided different results over time.

I first started to perform with a depth of 6, then increased it to 16, 32, 64. The variant with the starting depth of 64 (doubles in each convNet layer), didnt seem to produce that good of a result. It actually seemed to perform worse than starting the convNet with a depth of 32. So I decided to stick with the starting depth of 32.

With the SAME padding, I was able to get in 3 convNets whereas with the VALID padding I only got 2. One variant that I tried was to decrease and increase the final fully connected layers in the 3 Layer convNet variant.

Midway through the numerous attempts and modifying the layers (along with changing the Epochs, learning rate and the batch size), I realized that I was overfitting the dataset by quite a bit (can learn more on the top section of preprocessing data). This led me to think on different ways of preprocessing the data for a better result.

Next step that I took was to add in dropout layers. First, I started off with only th first convNet layer, then progressed to all the convNet layers. Finally I stopped at adding the final dropOut layer to the first Fully connected layer that I had after the 3 convNet layers.

- **Which parameters were tuned? How were they adjusted and why?**

All the parameters such as Epochs(tried an epoch of 1000 just for the heck of it), learning rate, batch size, drop out keep probability.

I kept tweaking the values based on what I learnt in class for the learning rate. I kept trying with a lower learning rate till I was satisfied with one. The batch size didnt seem to improve my scores by much. So, I decided to keep the size at 128 and vary the rest of the parameters as much as I could.

- **What are some of the important design choices and why were they chosen?**

Adding more convNets seemed to help with the improvement of the accuracy compared to adding more fully connected layers. When I reduced the number of fully connected layers, at certain parameters, I was seeing better accuracies.

I wanted to try both dropOut layers and the L2_regularization layers. I distincly remembered from the lectures that the addition of dropouts seemed to magially improve the accuracy of the neural network as it makes the network learn more on missing data that are available only at certain times. This reduces the problem of overfitting when the training dataset is spread across properly.