Capstone Project

Arpit Gogia
July 20th 2016

Machine Learning Engineer Nanodegree

Definition

Project Overview

Computers have long been an essential part of our lives. The past few years have seen the rise of computers as extremely powerful computational tools. Machine Learning and Artificial Intelligence domains have flourished in recent times. Computers have gained the power to understand and learn from what we give them to as input. Learning from Images, if achieved to a high accuracy, can be a big advantage of this increased computational power.

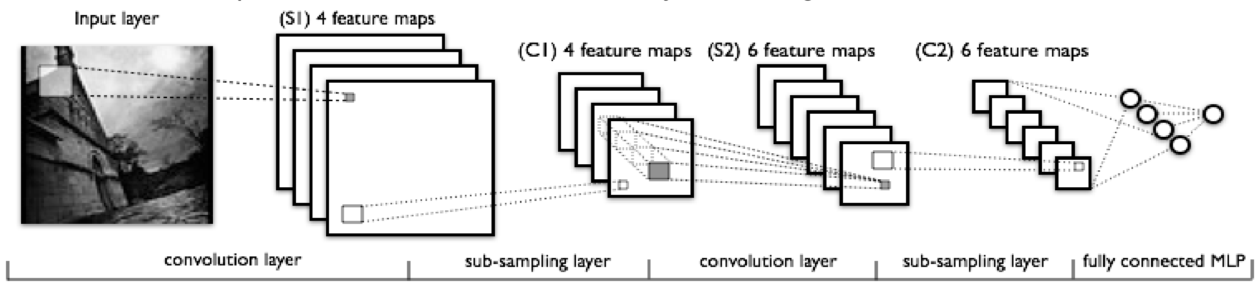
Our motive here is to recognise the Numbers from Street View Images. Accomplishing this would prove to be very useful simply because it would add immensely to the convenience of a user looking for a house/building on a particular street.

Problem Statement

Our aim as mentioned above is to create an agent which can learn from an existing database of number plates of Buildings and hence in the future be able to extract numbers from Google Street View images of houses/buildings to a reasonably good amount of accuracy.
Image Recognition will be achieved using Multi Layer Convolutional Neural Networks, a concept of Deep Learning.

Outline of the tasks to be performed:

1. Read the data (.mat files).
2. Separate the data and class labels as X and Y. The data is a 4D Matrix. The labels are 1D Matrix.
3. Reshape the data to suit our needs. Class Labels are converted to One Hot Encodings.
4. Construct a Neural Network and train the network using the Training Data.
5. Perform Optimizations to increase accuracy on testing data.



Metrics

The accuracy function used by me is defined as follows :

```
def accuracy(predictions, labels):
    return (100.0 * np.sum(np.argmax(predictions, 1) == np.argmax(labels, 1))
          / predictions.shape[0])
```

The predictions are an output of the softmax function, hence all the values are between 0 and 1. The labels are one hot encoded.

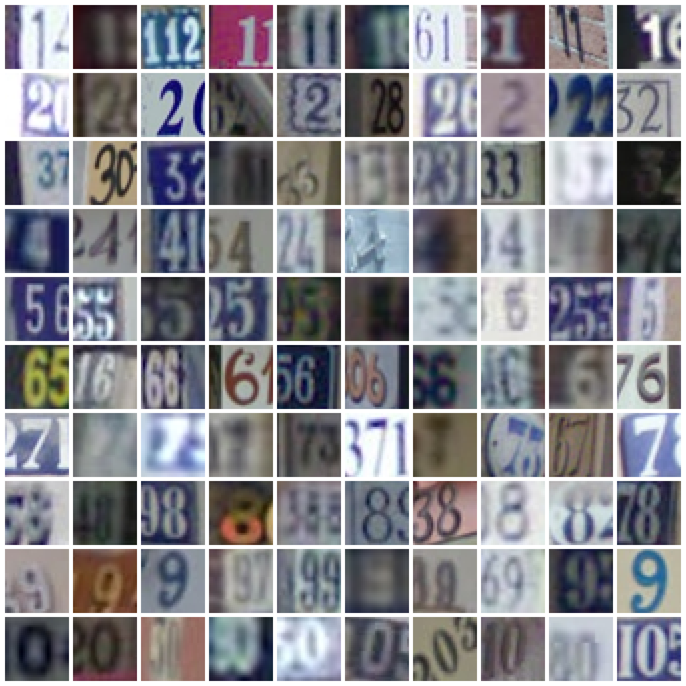
This is the same accuracy function as defined in the Assignment 4 of the Udacity Deep Learning Course.

The given function calculates the argmax for the predictions and the labels, takes the number of predictions where it is equal to the labels and then calculates the percentage accuracy using the number of predictions.

The metric that I have chosen is pretty simple in nature while being able to correctly report the accuracy of the predictions.

Analysis

Data Exploration



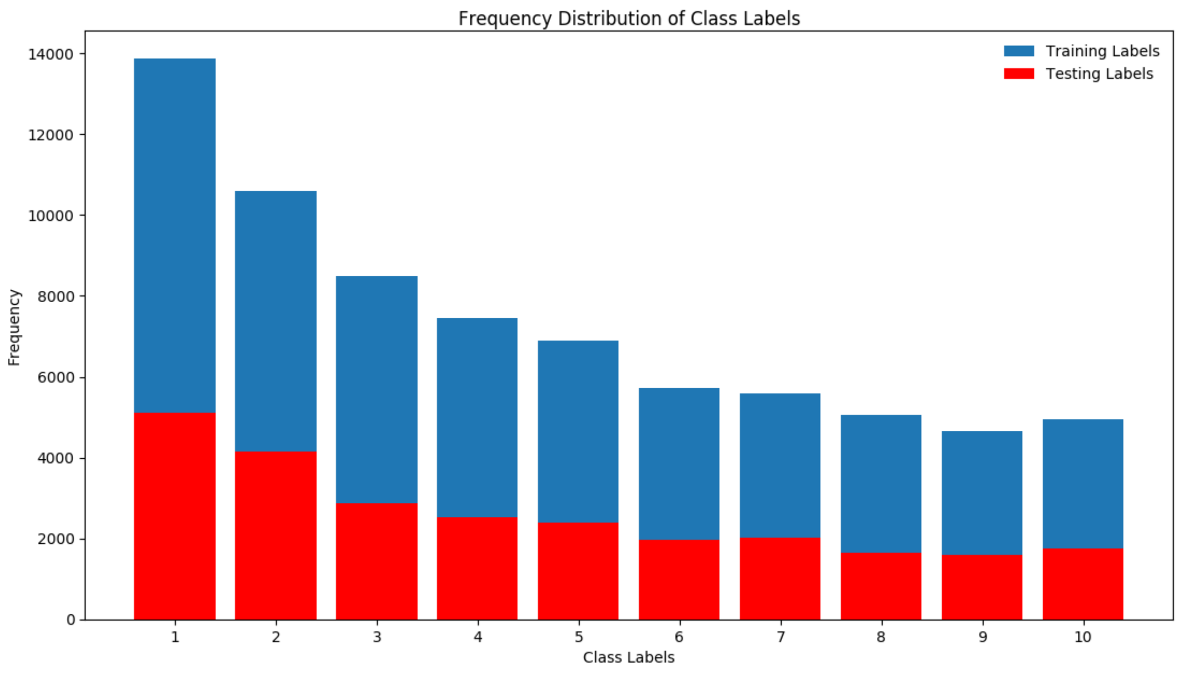
We can observe the following features from the above grid of SVHN Images (32x32):

1. The images are of various colors.
2. The images are taken from multiple angles, in some cases the different digits are at different angles individually.

3. The images although focussed on the number, contain some random objects which are treated as noise.
4. The numbers are available in a number of fonts.
5. Some images are blurry enough to not get recognised even by humans, as shown below.

Exploratory Visualization

I plotted the class labels against their frequency to obtain an idea of how the data is distributed.

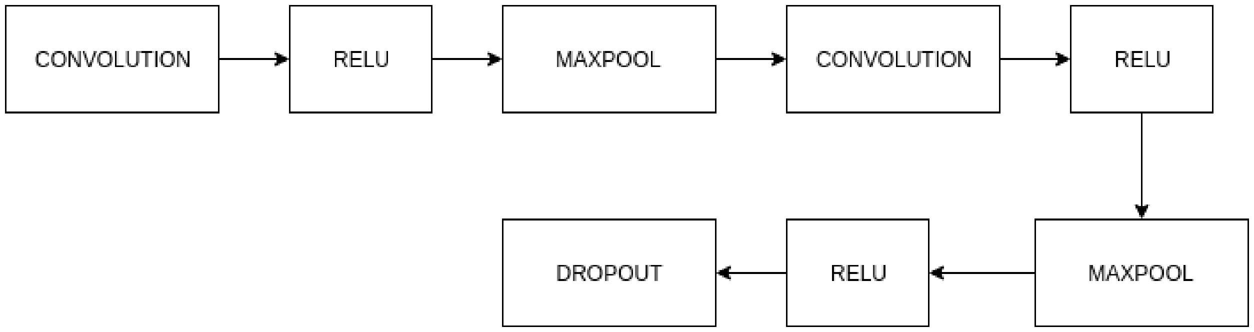


The plot shows that the datasets are considerably skewed towards label ‘1’ or the digit ‘1’.

Algorithms and Techniques

The model used in this project, to carry out supervised learning from images is the **Convolutional Neural Network**. Convolutional Neural Network is computationally more efficient than regular neural networks when used for image processing simply because it requires fewer layers than a regular computational neural network.
Linear Classifiers, that classify the data into one of the ten labels, are also undesirable in this case because they are slow and inaccurate for image processing.
Few important terms:

1. **Convolution:** A convolution operation is carried out in order to detect patterns in an image. It basically amounts to applying filters to an image to gain some information. After applying a convolution, the width and height of the resulting matrix may be lesser/equal to the input image but the depth is increased, depending on the number of filters applied.
2. **ReLu:** Rectified Linear Units or ReLu's are used to apply an elementwise activation function. This doesn't change the dimensions of the output as compared to the input.
3. **Pooling**: A very important layer that allows downsampling the data and attempting to retain the maximum information in reduced dimensions. In this scenario I have used the **Max Pooling** technique which basically selects the most responsive matrix cell under consideration.
4. **Dropout:** A technique used to reduce overfitting in neural networks. A fraction of neural network units are dropped out randomly to prevent overfitting.



Given above is an outline of the model used by me. It includes two convolutional layers, two maxpool layers, a dropout layer and Rectified Linear Units.

Benchmark

In the research paper (Multi Digit Number Recognition from Street View imagery using Deep Convolutional Neural Networks), the authors were able to achieve 96-98% accuracy.
Considering this to be a benchmark, I aim to achieve at least an average accuracy of **85-90% on the training or testing data set.**

Methodology

Data Preprocessing

1. The data is available in MATLAB's .MAT format. Reading of the data is accomplished by scipy.io.loadmat() function.
2. The output of scipy.io.loadmat() is a dictionary. X and Y components of the data are stored as **data and labels respectively.**
3. X is a 4D Matrix of 32x32 images with 3 channels, Red, Green and Blue. The shape of the X data is (32, 32, 3, <*Number of images*>) where number of images is 73257 for Training Data and 26032 for Testing Data.
4. Normalisation is carried out using the formula mentioned in the Udacity Deep Learning Course:

$$\frac{pixel}{128} - 1$$

5. The data needs to be reshaped to suit our Convolutional Network Shape. The final shape is (<*Number of Images*>, 32, 32, 3) for the data and (<*Number of images*>*,* 10) for the training labels. The basic operation carried out is transposing the 4D data matrix and applying one-hot encoding to the training labels. This is one of the major steps in preprocessing the data.

This completes the preprocessing of the data.

Implementation

This project makes use of the Neural Network module of TensorFlow. I have explained the various layers below:

1. The algorithm begins with the first Convolutional Layer with shape 5x5x3x16. This layer employs a 5x5 patch with 'SAME' padding for the convolution. The stride is set to 1x1. This is the first convolution operation on the data.
2. Next comes a REL unit which combines the first convolution output and the first layer biases. The output dimensions remain same. (32x32x16)
3. The max pooling results in an output size of 16x16x16. Max pooling reduces dimensionality.
4. The next convolution layer receives 16x16x16 input. The filters are similar to the first one with shape (5, 5, 16, 16). Another convolution of dimension 5x5 resulting in further reducing the dimensions of the image.
5. Standard Max Pool layer is then applied to get an output of shape 8x8x16.
6. The next layer converts the input into shape (128, 10). This is the fully connected layer which produces a matrix to input into a classifier.
7. The final step includes the last REL unit and a layer for the dropout.
8. After this the output is fit to be supplied to a classifier.

Thus we have in total 4 layers, with a dropout of 0.93.
After the model is built, the dataset is converted to that Model in the TensorFlow Graph.

```
logits = model(tf_train_dataset)
```

The softmax cross entropies are calculated and reduced for the model against the labels using the TensorFlow provided function

```
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits, tf_train_labels))
```

The AdamOptimizer is then applied to minimize the loss

```
optimizer = tf.train.AdamOptimizer(0.001).minimize(loss)
```

Finally the softmax activations are calculated for the training and testing model

```
train_prediction = tf.nn.softmax(logits)
test_prediction = tf.nn.softmax(model(tf_test_dataset))
```

The next step is start the iterations. In each iteration the batch dataset is chosen using an offset dependent on the epoch.

```
offset = (step * batch) % (train_labels.shape[0] - batch)
batch_data = train_data[offset:(offset + batch), :, :, :]
batch_labels = train_labels[offset:(offset + batch), :]
```

All the variables are then supplied to the TensorFlow session and the predictions are obtained

```
_, l, predictions = session.run([optimizer, loss, train_prediction], feed_dict=feed_dict)
```

The feed_dict is the dictionary of the input data for the placeholders in the TensorFlow Graph. I performed 10, 000 epochs/iterations with the best analysed hyperparameters with a batch size of 16. Another 20, 000 epochs were performed with the same hyperparameters and the accuracies have been shown in the table below:

S.No.	Epochs	Batch Size	Training Accuracy	Testing Accuracy
1	10,000	16	74.038125	90.05
2	20,000	16	80.829375	93.2740625

The corresponding output files are ***output_2_10000.txt and output_2_20000.txt***. The information loss is shown in the two graphs in the **Free Form Visualization** section.

Performing more tests to gain more accuracy can be accomplished by performing more number of epochs.

Validation is performed using the accuracy function mentioned in the code and the report before.

```
def accuracy(predictions, labels):
    return (100.0 * np.sum(np.argmax(predictions, 1) == np.argmax(labels, 1))
          / predictions.shape[0])
```

Plotting the accuracy for each epoch can give us a clear visual indication of our model. Information Loss at each instance is obtained by the TensorFlow session

```
_, l, predictions = session.run([optimizer, loss, train_prediction], feed_dict=feed_dict)
```

Where variable **l** is used for storing the loss at each epoch.

Refinement

Refining a Deep Convolutional Neural Network is very tedious but rewarding task. Following was my approach to refinement:

1. The first task is to identify how many layers your Convolutional Network should posses. I was running the Model on a Ubuntu PC with 8GB RAM and 2GB NVidia Mobile GPU. In order to keep the computation less complex and resource free I used 4 layers in all my testing Scenarios.
2. Normalisation of the data points can also affect accuracy. As mentioned before the normalisation formula used was the same as that mentioned in the Udacity Deep Learning Course

$$\frac{pixel}{128} - 1$$

3. The other two parameters I varied in order to test the accuracy are shown in the table below along with the observations. After the below tests I decided to use the **2nd set** of parameters in the final model. Upon searching on Google, I found out about the AdamOptimizer from this link.

S.No.	OPTIMIZER	DROPOUT PROB.	LEARNING RATE	TRAINING ACCURACY	TESTING ACCURACY
1	AdamOptimizer	0.75	0.001	65.31875	84.084375
2	**AdamOptimizer**	**0.93**	**0.001**	**74.038125**	**90.05**
3	AdamOptimizer	0.95	0.001	69.124375	87.568125
4	SGD	0.93	0.001	31.459375	68.24875
5	SGD	0.73	0.0001	14.921875	19.01

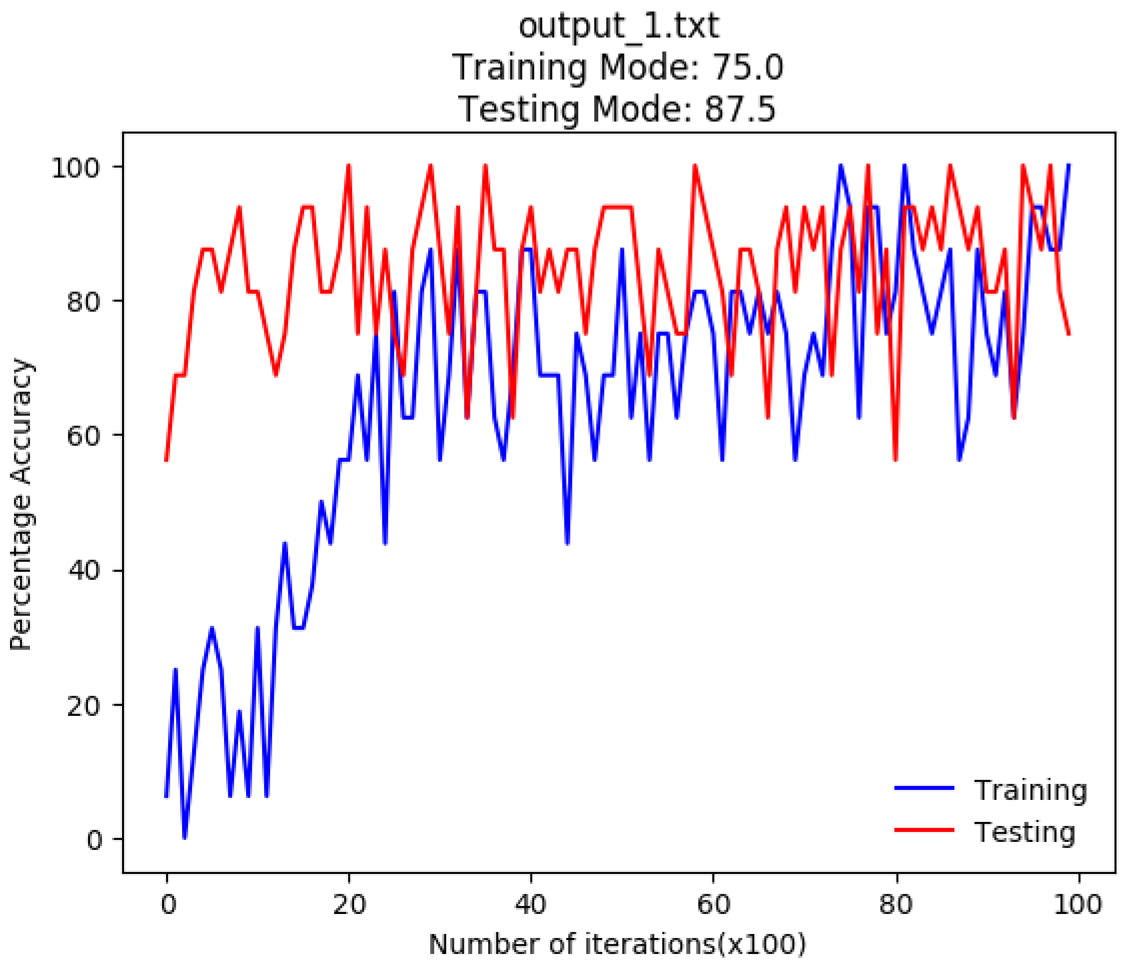
NOTE: Please find the corresponding output files as 'output_<s.no.>.txt'
The Accuracies shown above are Averaged over all iterations
Average Accuracies are shown in the output files

4. Number of steps was also fixed as 10000 to balance the computational load and accuracy.
5. Other Neural Network parameters like Strides and Patch size were kept constant throughout the above tests. The best size for the Patch was 5x5 to gain more information in each iteration.
6. The batch size was set to 16.

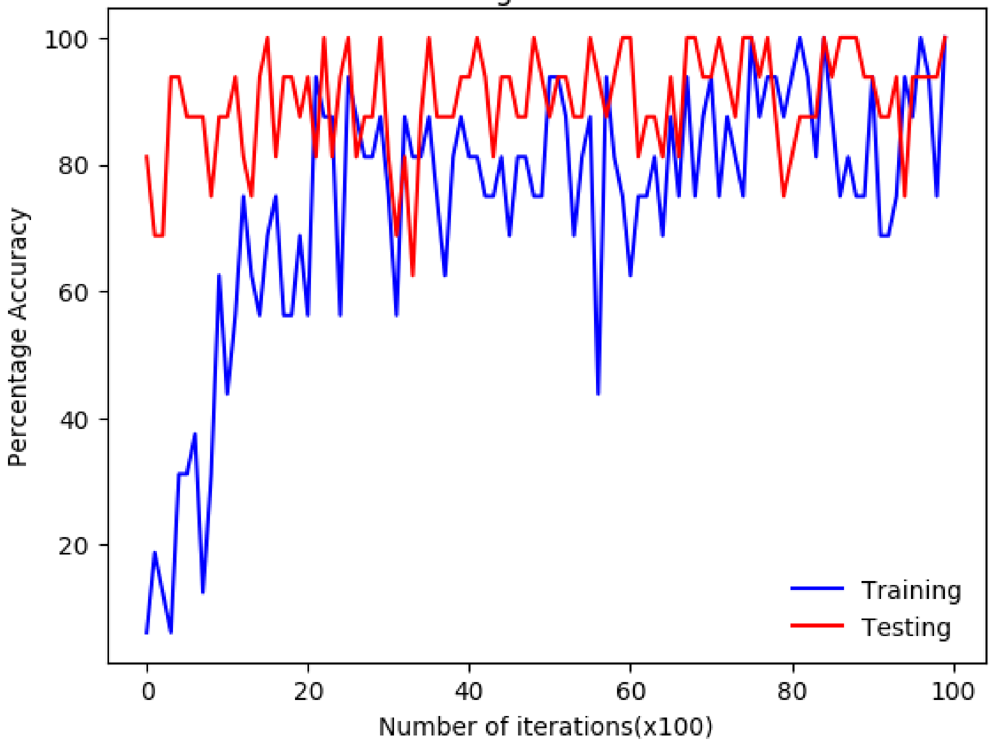
Results

Model Evaluation and Validation

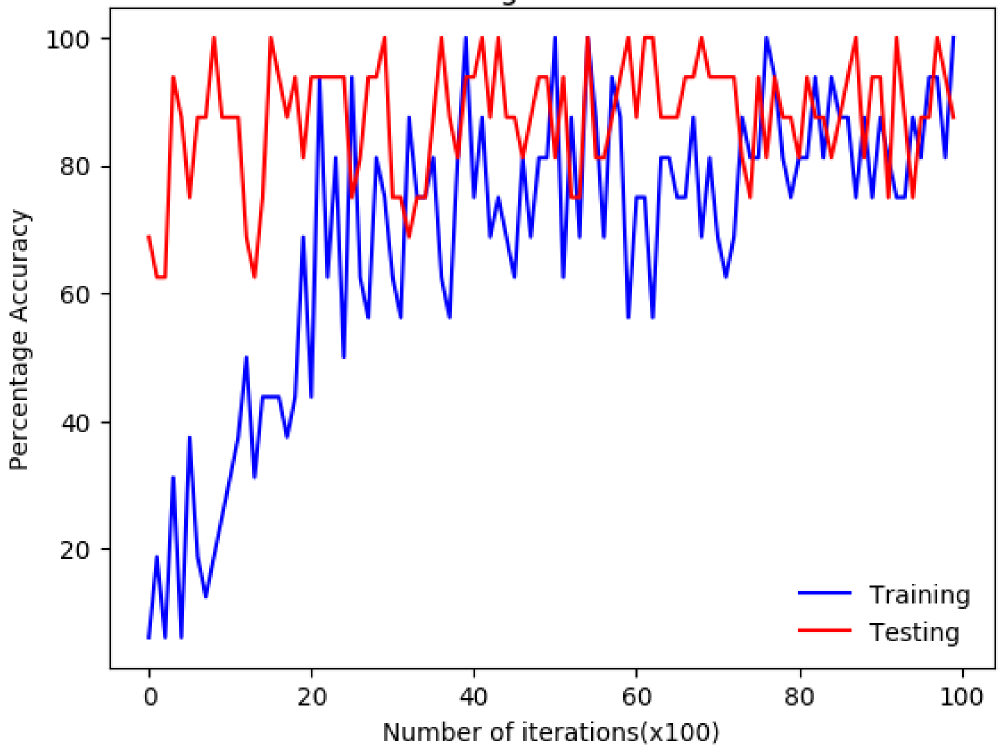
The model was run in batches of 16 for a total of 10000 iterations on testing and training data both. As mentioned above, the output files were analysed and the accuracy at every 100th step was plotted against the number of iterations. Another metric that I calculated was the mode of all the accuracies at every 100th iteration.

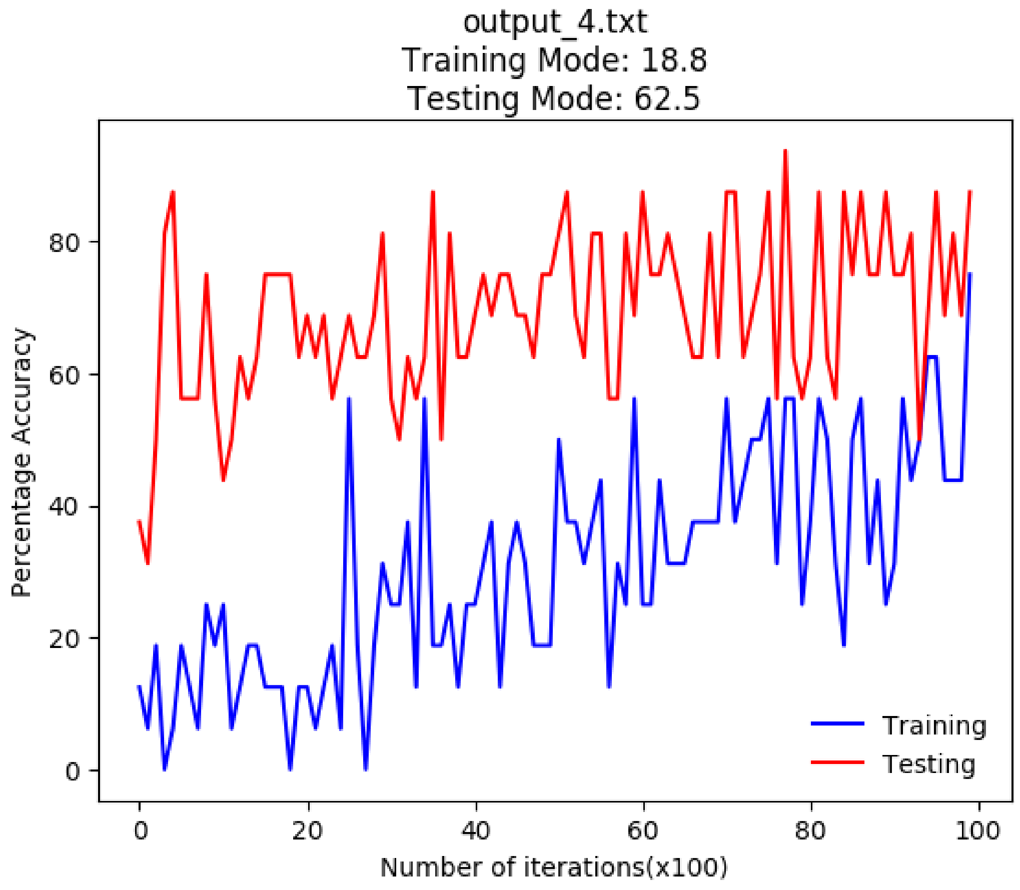


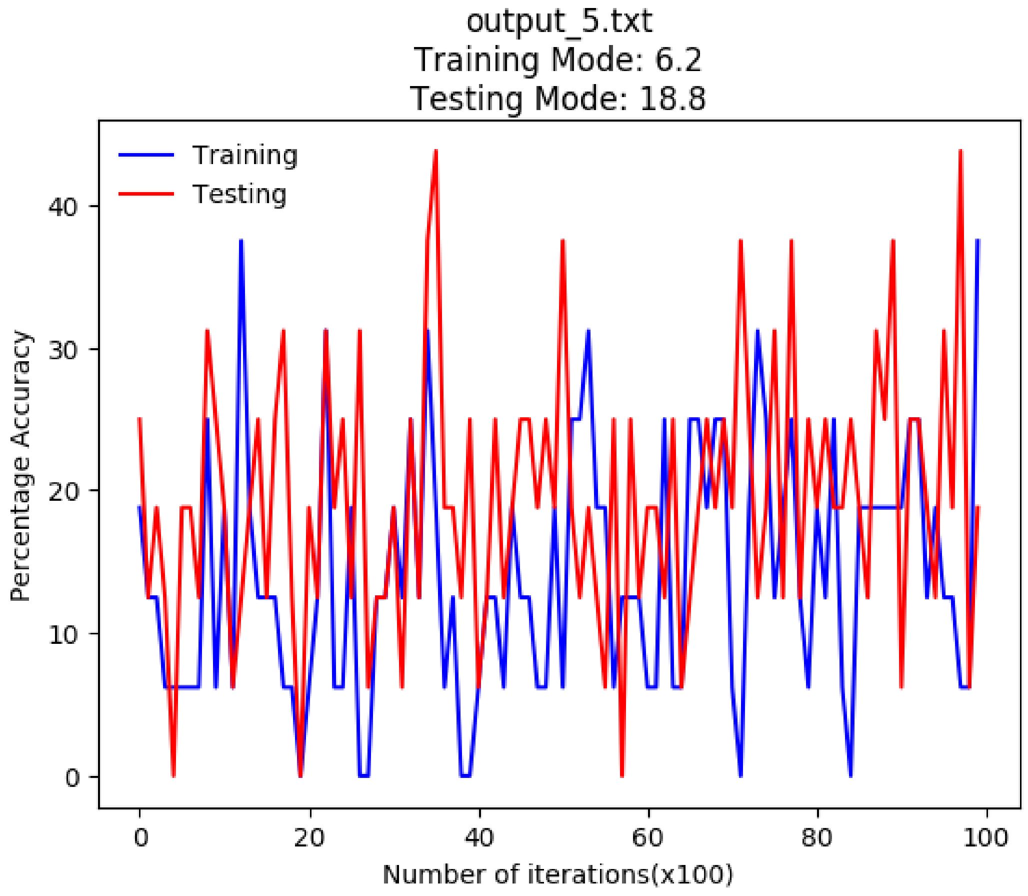
output_2_10000.txt
Training Mode: 75.0
Testing Mode: 93.8



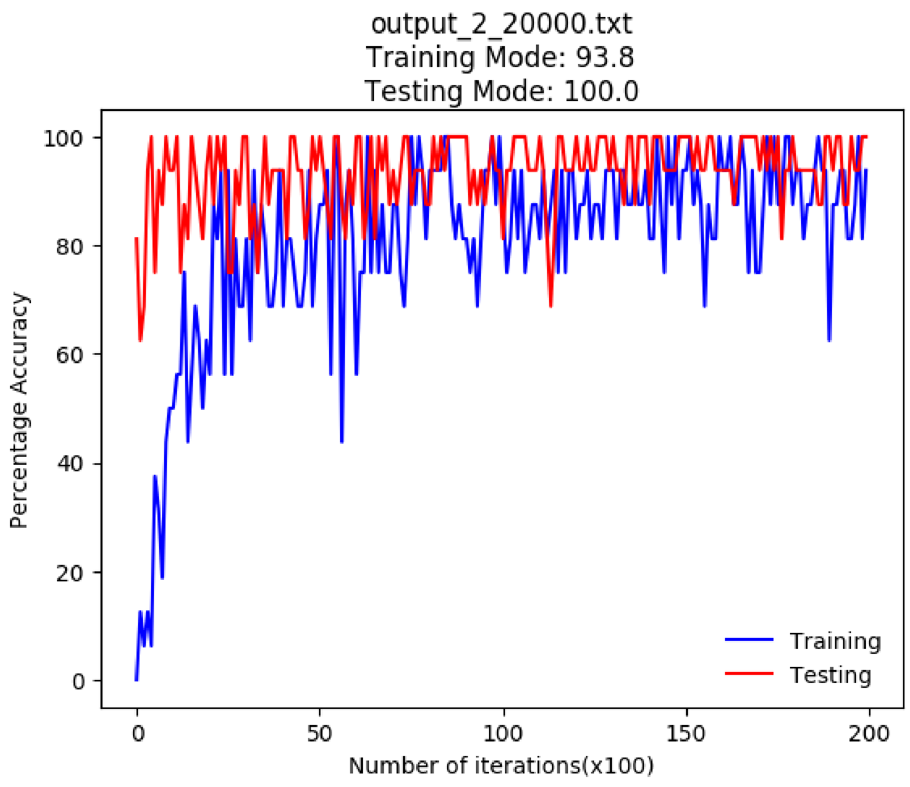
output_3.txt
Training Mode: 81.2
Testing Mode: 93.8




output_4.txt
Training Mode: 18.8
Testing Mode: 62.5
Percentage Accuracy
Number of iterations(x100)
Training
Testing
0
20
40
60
80
100


output_5.txt
Training Mode: 6.2
Testing Mode: 18.8
Percentage Accuracy
Number of iterations(x100)
Training
Testing
0
10
20
30
40
0
20
40
60
80
100

The graphs can show that configuration 5 (as in the table above) is extremely undesirable. Configuration 2 and 3 seem similar and one could even say that 3 is better since it's training mode is higher however this is because this mode isn't calculated over the whole of the training data so it may not give a complete picture. Another observation worth mentioning is the fact that configuration 2 shows higher occurrences of 100% accuracy with negligible information loss.
Configuration 2 was run in two epochs of 10000 and 20000 each. It can be seen that increasing epochs leads to better accuracy in the long run.



The Model corresponding to the second configuration is the most accurate and hence my final choice of the model.

OPTIMIZER	DROPOUT	LEARNING RATE	TRAINING ACC.	TESTING ACC.
AdamOptimizer	**0.93**	**0.001**	**74.038125**	**90.05**

Justification

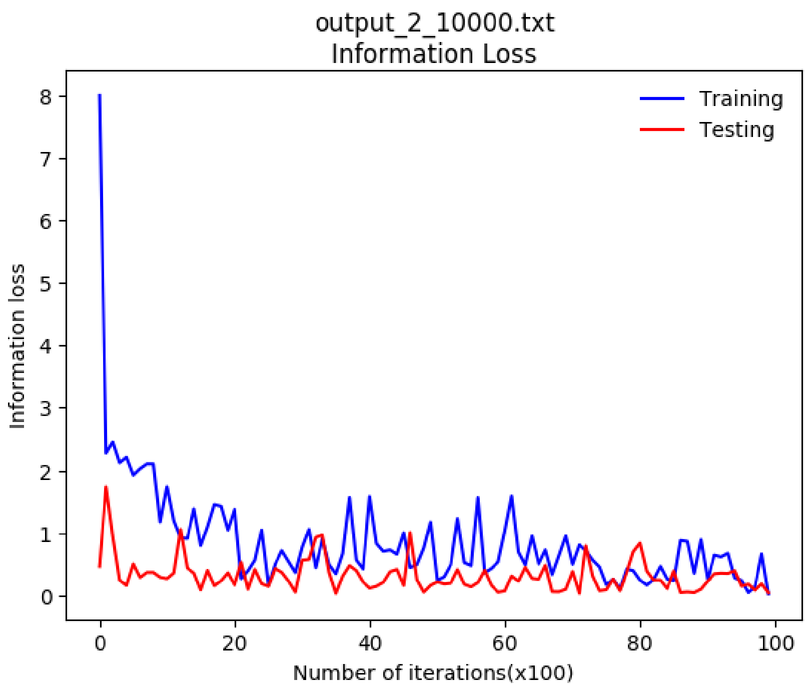
In the benchmark section I mentioned that I would like to have a model with an average accuracy of 85%. I have achieved 90% accuracy with frequent 100% accurate predictions and a 93.8% mode on the testing data and 100% mode with increased number of epochs, which is considerably good. Considering the limited Hardware I have, I may not be able to deepen the Neural Net or increase its complexities by minutely fine tuning the parameters.

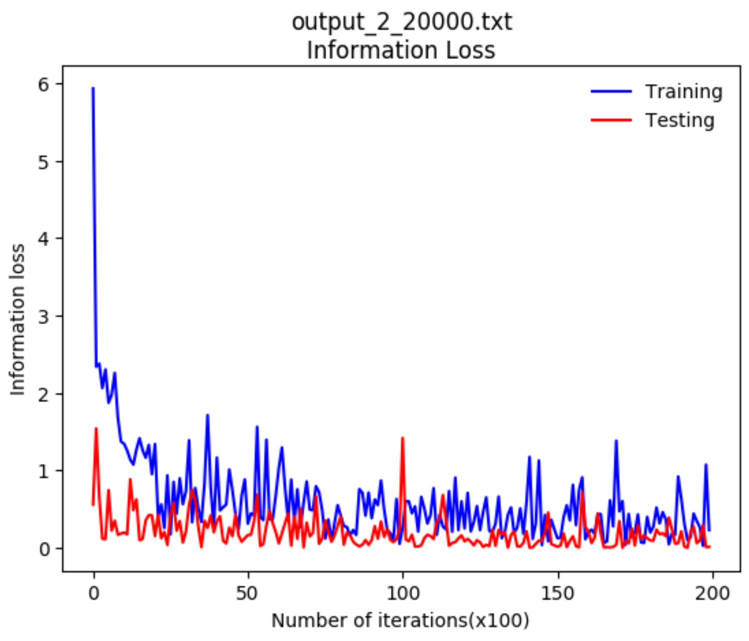
Is the final solution significant enough to have solved the problem?

I believe this model may be able to solve the problem of recognising numbers from Street View Images. It is able to achieve 100% accuracy in many testing cases, and hence by repeated training with increased number of epochs, it can become a robust model that can be deployed in the real world.

Conclusion

Free Form Visualization





The above graphs show the Information Loss for our chosen configuration. It is easily visible that the model learns well from the training data gradually reaching to zero information loss. Another thing that we observe is that the test data starts with very low information loss and overall has comparatively lesser information loss than the training data. This is a representation of our aim with this project, to minimize information loss and increase accuracy.

Reflection

The problem we are dealing with in this project is to learn to recognise numbers in low resolution images captured by Google Street View cameras. The images could have been taken in favourable or adverse conditions, sometimes resulting in photos that are not comprehensible even by the human eye. I aim to solve this problem by harnessing the power of Deep Learning and applying a Convolutional Neural Network with tuned hyperparameters to recognise numbers from such images. This Neural Network must also scalable and have the right balance between accuracy and computational time/complexity.
I would say that the most interesting part of this project is finding the right hyperparameters. There are theoretically infinite possible permutations of the hyper parameters. It would be interesting to find out a combination which would predict the numbers on an image so blurry that even humans can't comprehend it. Identifying when the trade off between accuracy and computation time is the right amount is also crucial to training a CNN Model.

The model discussed in this project can be applied to the real world with more extensive testing to try to improve its accuracy even further.

Improvement

1. **Hardware:** I do not have access to more powerful machines with GPUs resulting in higher computation power. If I do have access to those, the model would have been trained on more batches and data sets than currently.
2. **Depth and HyperParameter Tuning:** One of the most crucial parts of CNN is figuring out the appropriate depth and parameter combinations. I believe if I had tried more combination of parameters or implemented some kind of Grid Search for parameters, I would be able to increase the accuracy even more.

Helpful Links/Papers/Courses

1. http://stats.stackexchange.com/questions/184448/difference-between-gradientdescentoptimizer-and-adamoptimizer-tensorflow
2. Udacity Deep Learning Course
3. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks
4. CS231n Convolutional Neural Networks for Visual Recognition
5. Reading Digits in Natural Images with Unsupervised Feature Learning
6. TensorFlow Convolutions Sample Code