

**Digit recognition in Street View House Number Recognition using Deep Convolutional Neural Networks**

**Project Report**

ANLY 535 - Machine Learning II

**GROUP 4**

Priya Singh

Aniket Naik Desai

Aby Koshy

Aaina Malhotra

Pooja Chopra

Contents

[Introduction 3](#_Toc32570841)

[Related Works: 4](#_Toc32570842)

[Data: 5](#_Toc32570843)

[Technical Approach: 10](#_Toc32570844)

[Test & Evaluation: 14](#_Toc32570845)

[Conclusion: 20](#_Toc32570846)

[References 20](#_Toc32570847)

Introduction  
  
Identifying multi-digit numbers in pictures taken at street-level is an essential part of modern-day map building. A perfect example of a corpus of such street-level photographs is Google’s Street View imagery, constituted of millions of geo-positioned 360-degree images. The capability to automatically reproduce a home number from a geo-positioned bit of pixels and correlate the reproduced number with a known address helps to distinguish, with a high level of precision, the position of the building signifies. Arbitrary multi-character text recognition in pictures is yet very challenging. This difﬁculty occurs due to the broad variability in the presentation of the text in a wide array of fonts, colors, styles, orientations, and character arrangements. The identification problem is considerably hampered by environmental factors such as brightness, shadows, secularity, and occlusions as well as by image addition factors such as resolution, motion, and focus blurs.

This project focused on recognizing single digits in a real application by reading house numbers from street level photos using a convolutional neural network with multiple layers. Our approach was fully supervised, whereas the previous studies used unsupervised learning methods also. We found the best accuracy by increasing the number of convolution and pooling layers (until it starts decreasing). We have used greyscale and non-greyscale images in order to compare the accuracy between the two types of images and measure the accuracy. The Keras and Tensorflow were used as two frameworks. We also varied the number of layers and the number of epochs to improve accuracy. We have also tested the accuracy of the model by using the logistic regression,SVM, and random forest.

Related Works:

Understanding various objects in images has been one of the significant goals of machine learning. Possibly the most popular method to the image-based analysis of number series involves coupling a sliding window detector with a number classifier (Wang et al., 2012; Jaderberg et al., 2014b). The detector and the classifier are typically trained individually, using different loss functions. The seminal work on ConvNets of LeCun et al. (1998) introduced a graph transformer network architecture for understanding a sequence of numbers when reading checks and also explained how the whole system could be trained end-to-end. That system, however, still relied on various ad-hoc elements for obtaining candidate locations.

Lately, ConvNets working on cropped series of characters have achieved state-of-the-art performance on house number identification (Goodfellow et al., 2013) and natural scene text recognition (Jaderberg et al., 2014a). Goodfellow et al. (2013) trained a separate ConvNets classifier for each character position in a house number with all measurements except for the output layer shared among the classifiers. Jaderberg et al. (2014a) showed that synthetically created images of text could be used to train ConvNets classifiers that deliver state-of-the-art text recognition performance on real-world images of cropped text.

Improvements in the availability of computational sources, progress in the extent of accessible training sets, and algorithmic advancements such as the use of piecewise linear units (Jarrett et al., 2009; Glorot et al., 2011; Goodfellow et al., 2013) and dropout training (Hinton et al., 2012) have resulted in many new successes using deep convolutional neural networks. Krizhevsky et al. (2012) obtained dramatic developments in state of the art in object recognition. Zeiler and Fergus (2013) later improved upon these results.

Convolutional neural networks have earlier been used principally for purposes such as recognition of individual objects in the input image. In some cases, they have been accepted as elements of systems that explain more complex tasks. Girshick et al. (2013) use convolutional neural networks as feature extractors for a system that performs object detection and localization. However, the system as a whole is more extensive than the neural network portion trained with backdrop and has unique code for managing enough of the mechanics such as introducing candidate object regions. Szegedy et al. (2013) pointed out that a neural network could learn to output a heatmap that could be post-processed to resolve the object localization problem. In our study, we use a related strategy, but with more limited post-processing and with the added requirement that the output is an ordered sequence rather than an unordered list of detected objects. There is no need for a separate component of the system to offer candidate segmentations or provide a more significant level model of the image.

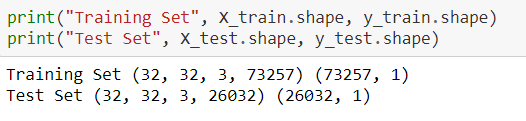
Data:

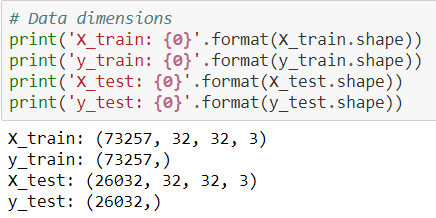
SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with the minimal requirement on data preprocessing and formatting. The data is built from house numbers extracted from house numbers in Google Street View images. It consists of little over 73K images for the train and 26K images for the test. The data contains label data with color information with various natural backgrounds. The data is divided into three datasets - train, test, and extra. We are going to use a train and test datasets in our project. The link to the dataset is<http://ufldl.stanford.edu/housenumbers/>.

1. **Attribute Details:**

* 32x32 cropped samples from the classification task of the SVHN dataset.
* Total 10 Classes, 1 for each digit. i.e., each sample is assigned only a single-digit label (0-9)
* Seventy-three thousand two hundred fifty-seven digits for training, 26032 digits for testing.
* Comes in two formats:
* Original images with character level bounding boxes.
* 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides).

1. **Data Preprocessing:** Once the data is loaded in the jupyter, the train and test data will be pre-processed to remove any inconsistency in the data. The data is split into X train, y train, x test and y test from the train and test data. Below is the shape of the training and test data. The data is then transposed into image arrays to convert the pixels values of each image in all three channels (RGB). Moreover, the total number of images is 99289 which is calculated from the X train and X test.





The plot images function is defined for plotting the images after running the model. The plot images function takes nrows and ncols images as input. Here are some of the training and test set images.



Figure 1 – Training set images

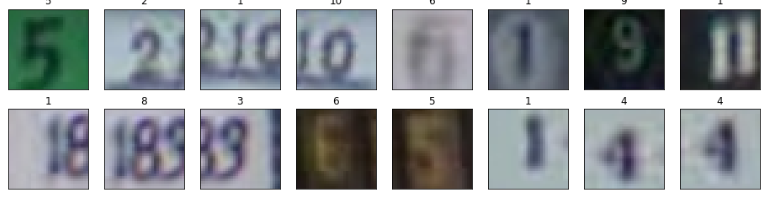


Figure 2 – Test set images

There are unique labels in the train data. In the next step, the label ten is converted to 0 in both train and test data. Now, the unique values are from 0 to 9. The training data is split into train and validation set by using train\_test\_split function. The distribution is shown in figure 3. The train, test and validation data are now converted into the greyscale image from RGB given in figure 5. In the next step, images are converted to float32 data type as it helps to rescale their value. In the next step, the data is normalized by calculating mean and standard deviation on the training data which was converted to greyscale. The greyscale inputs which train greyscale, test greyscale and validation greyscale are subtracted from the mean and divided by the standard deviation. The next step is using the one-hot label encoding that transforms the label values to a one-hot encoding images. The data is stored using the h5py package which stores the large amount of numerical data which is easy to manipulate from numPy. The data saved in the h5py form is loaded again for the modeling.

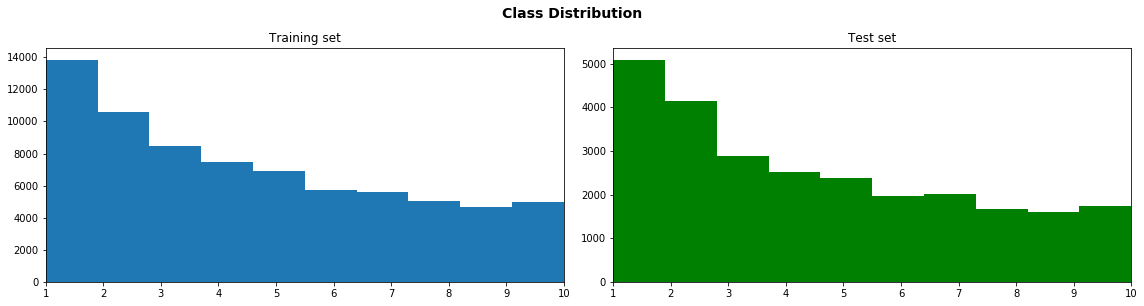


Figure 3 – Distribution of train and test data

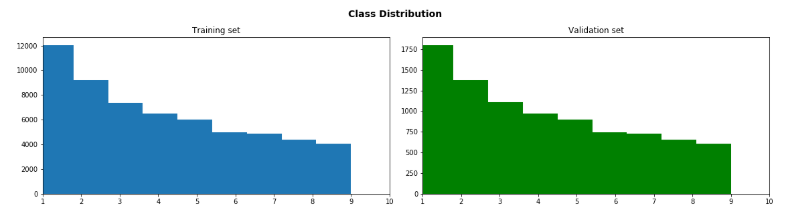


Figure 4 – Distribution of train and validation data

Figure 5 is the class distribution of training and test set. It shows that there are a greater number of "1” and “2” class distribution in both train and test data. The number of data in each set is y shape = 63733, y validation = 9524, and y test = 26032.

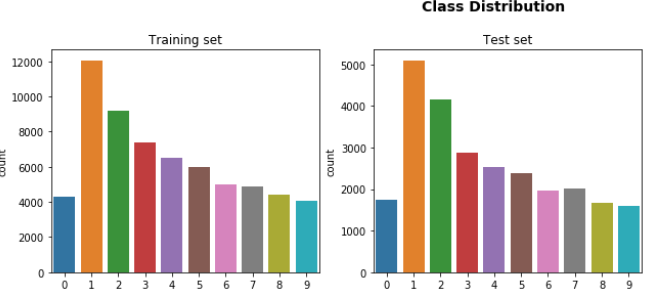


Figure 5 – Class distribution

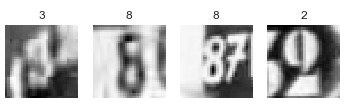


Figure 5 – Greyscale images

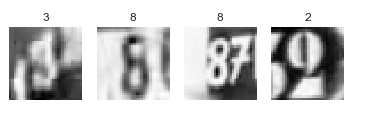


Figure 5 – Normalized train data

Technical Approach:

The project is a combination of image and digit processing of street view house numbers. The images consisted of many digits of house numbers from 1 to 6 digits. The study used cropped images, which were 32 x 32 which had single or double digits in the cropped images. The study used a convolutional neural network, which is mainly used for digit recognition in images. Initially, the train.mat and test.mat data was loaded using the **scipy.io** **import** load mat. We have used two frameworks for our model:

* Tensorflow
* Keras

1. **Tensorflow:**

The TensorFlow was run with the help of GPU in google colab. The helper function is defined for plotting the results when we are predicting the labels. It randomly selects the images. This function defines the nrow and ncols. When the prediction is not passed, it gives True and if prediction is passed then it displays the prediction and labels of images. The get batch is defined which is the subset size of the training sample and it is used to train the network. Our convolutional neural network was built using the following parameters. There were two Convolution layers followed by the max-pooling layer after each convolution.

* Initializing the configurations of the CNN and the data dimensions
  + Convolution Layer 1: Weight Size 5 x5, Feature maps output: 32 plus relu function as the activation
  + Max Pooling layer: pool window size 2 x 2, stride 2
  + Convolution Layer 2: Weight Size 5 x5, Stride: 1, Feature maps output: 64 plus relu function as activation
  + Max Pooling layer: pool window 2 x 2, stride 2
  + Flatten Max Pool output
  + Dense layer 1: Weight shape: Units 256 plus relu as the activation function
  + Dropout:
  + Logits layer: Weight shape: [dropout layer x 10]

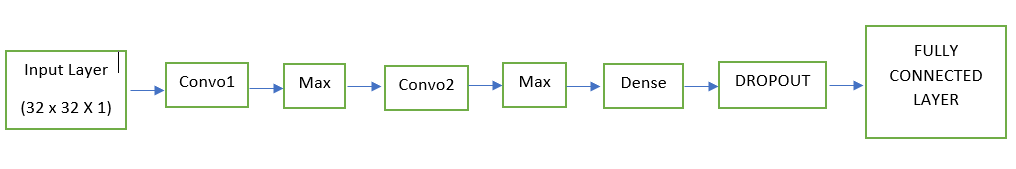


Figure 6 – Deep convolutional neural network

Create placeholder variables (input variables, dropout): The placeholder variable is initialized with the float32 data type.

* Create model architecture / computational graph: The model architecture is defined in figure 5. The
* Define loss function and optimization: The loss function is calculated for each iteration using the prediction from cnn\_modelfn(x) we defined in the model. Next, the adam optimizer is defined by using the calculated loss function.
* Define evaluation metric: To evaluate the performance of the model, the accuracy can check whether the index of the maximum value of the predicted index is equal to the labeled image and both will be a column vector. The accuracy is calculated across all the images and averaging them out
* Initialization of TensorFlow: The TF graphs have been created, the TensorFlow session is used to execute the graph.
* The checkpoints are initialized. Saver object is created, which is used for storing and retrieving the variables from the TensorFlow graph.
* Model Training
  + Initialize neural network hyperparameters## No of example in each batch for updating weights with a batch\_size = 512
* Batch Data Generator
  + In each iteration, a new batch of data is selected from the training set: In this model new batch of 512 images will be selected
  + feed\_dict function (Used in training, evaluating, and making predictions): This will return a batch of the data based on batch\_size & step. And, at every epoch (when the step is 0) it will shuffle the data
  + evaluate\_batch function (Used in training) This function will split the validation and test set into batches and calculates the accuracy over all the batches.

1. **Keras:**

We choose Keras as another framework for building the models and test the neural network with minimal lines of code. We performed a convolutional neural network on RGB images. In this mode, we also have not used a validation data set. Since we have performed a model on greyscale images. The loaded data was converted to float32 and normalized our data value to the range [0,1]. The test and train data are then transposed. Our model used a batch size of 1288 and 20 epochs

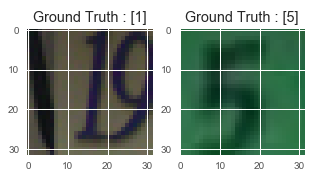


Figure 7: RGB image of train and test data

Our convolutional neural network was built using the following parameters.

* + Convolution Layer 1: Weight Size 3 x3, Feature maps output: 32 plus relu function as activation
  + Max Pooling layer: pool window size 2 x 2,
  + Dropout: 0.25
  + Convolution Layer 2: Weight Size 3 x3, Stride: 2, Feature maps output: 64 plus relu function as an activation
  + Max Pooling layer: pool window 2 x 2,
  + Dropout: 0.25
  + Convolution Layer 3: Weight Size 3 x3, Feature maps output: 64 plus relu function as an activation
  + Dense layer with units 64 and relu as an activation
  + Dropout: 0.2
  + Dense layer with units 64 and relu as an activation
  + Dropout: 0.5
  + Dense layer with units 10 and softmax as an activation

Test & Evaluation:

* Testing and validity: The approach was tested by using the confusion matrix and calculating the test accuracy of the model. Moreover, by plotting the misclassified and correctly classified images which will predict whether the image of the digit is predicted correctly shown in figures 8 and 9.
* The approach used is valid when looking at the methods used in previous studies.
* To evaluate our models, we have done test set performance, which was done by predicting the test set and calculate accuracy based on predictions.

**Model 1:** Modeling in TensorFlow framework using greyscale images

In figure 8, the five labels have a high accuracy of 93% than other digits and label 2 has 92%. In figure 9, we can see that their digits like 3 and 8 are incorrectly classified mostly. Moreover, in figure 10, it shows correctly classified digits which are mostly 2 and 6.

In figure 11, the train and test loss look decreasing, and it shows that it is perfectly fitting. The final test accuracy reported in the model is 87% which seems to be low as compared to previous studies.

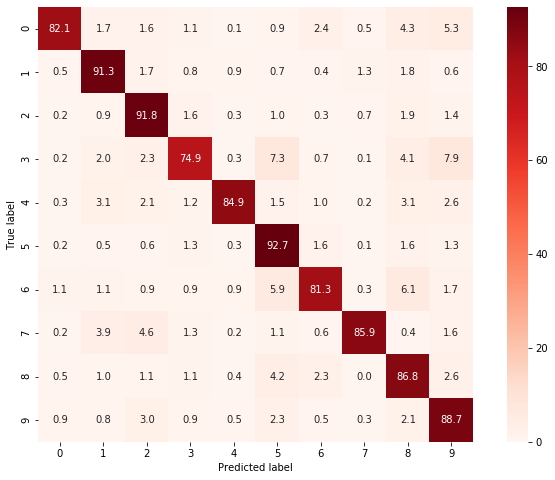


Figure 8: Confusion Matrix

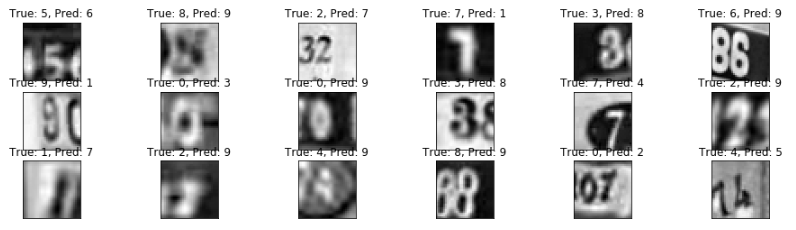


Figure 9: Incorrectly classified images

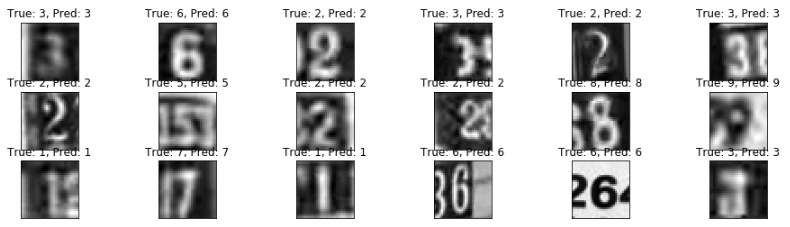


Figure 10: Correctly classified images

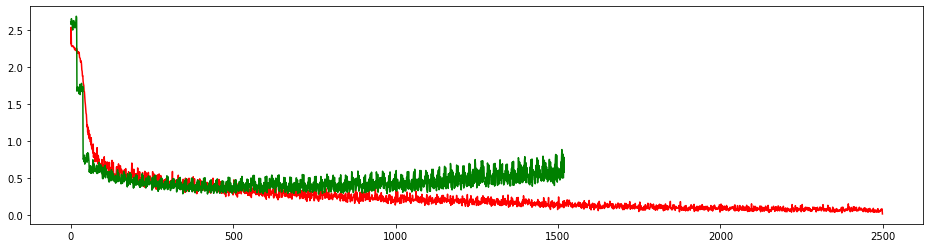


Figure 11: Train loss versus valid loss (Red = Train loss and green = Valid loss)

**Model 2:** Modeling using TensorFlow framework using non grayscale images

The CNN architecture remains the same and accuracy of this model was 88% which is slightly better than the greyscale images

**Model 3:** Modeling using Keras framework using grayscale images

The model in the Keras framework had an accuracy of 92.19% and Loss is 28% without batch normalization for greyscale images. The CNN architecture used had three convolutional layers, two filters, two dropouts with max-pooling of 2 x 2 I.e. taking the max from 2 x 2 layers of weights I.e. dense layer The below figure 12 and Fig 13 shows the training/validation accuracy and training/validation loss. The training and validation accuracy is increasing the number of epochs. Moreover, training and validation loss is decreasing with the number of epochs.

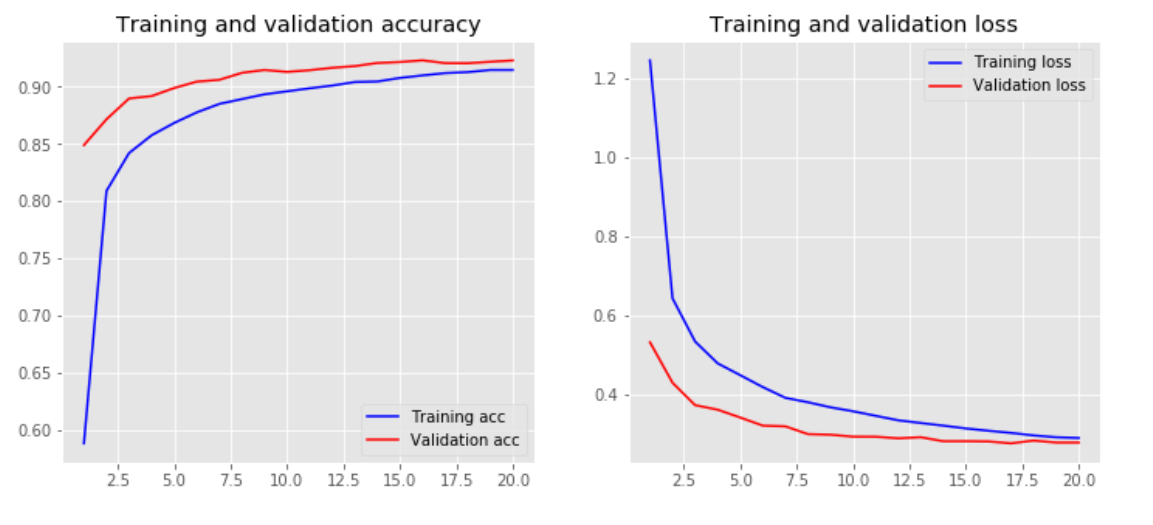


Figure 12: Training and validation accuracy Figure 13: Training and validation loss

**Model 4**: Modeling using Keras framework using non-grayscale images

In model 4, we have used Keras as a framework and applied algorithm on non-greyscale images. Figure 12 is the image from the training set. To visualize the image with the label, we have initialized the y with the train data to extract the images and labels from the data. The axes and set titles are set for labeling 0 to 9 digits shown in figure 15. Our model used three filters;3 drops out with Max pooling of 2,2 pixel i.e. taking the max from 2 col & 2 rows and two layers of weights i.e. Dense Layer. We used the below parameters for our model:

* batch\_size = 128
* nb\_classes = 10
* nb\_epoch = 20

The model accuracy was 89% and Validation loss was 38% . The below figure 15 shows the training/validation accuracy and training/validation loss. The training accuracy is increasing more than the validation accuracy with the number of epochs. The training loss is greater than the validation loss. The model seems to be perfectly fitting.

**Model 5:** Modeling on non-grayscale images by using batch regularization

By using batch regularization, it helped in improving the accuracy by standardizing the inputs to a layer for each mini-batch**.** Accuracy with no grey images was Accuracy: 89.6% and validation loss was 38%. After using Batch normalization accuracy increased to 93% and validation loss decreased to 30%. Figure 17 shows the digits predicted from the data. It was able to predict many digits correctly. Figure 18 shows the confusion matrix which helps in identifying which class was misclassified incorrectly minimum and maximum times.

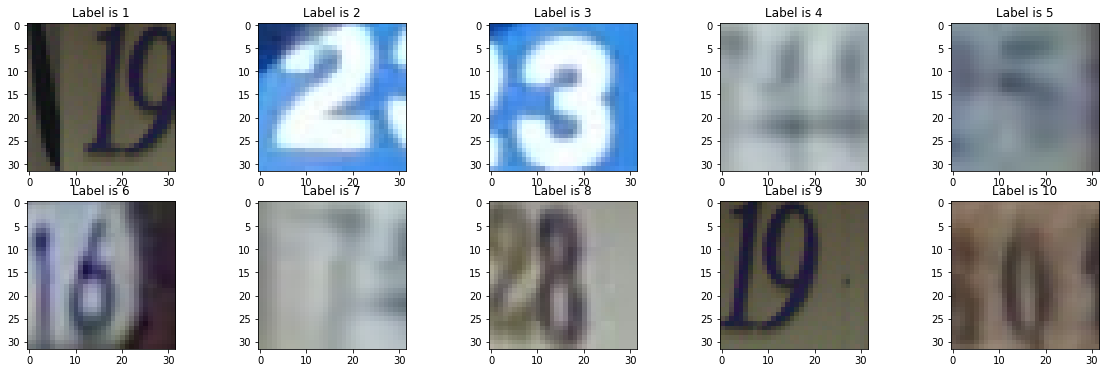


Figure 14: RGB image

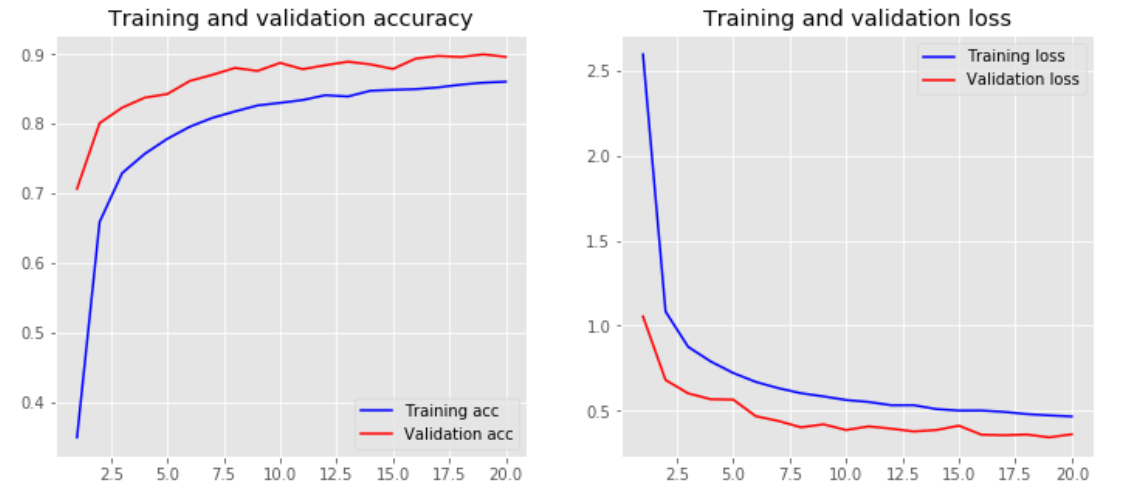


Figure 15: Training and validation accuracy Figure 16: Training and validation loss

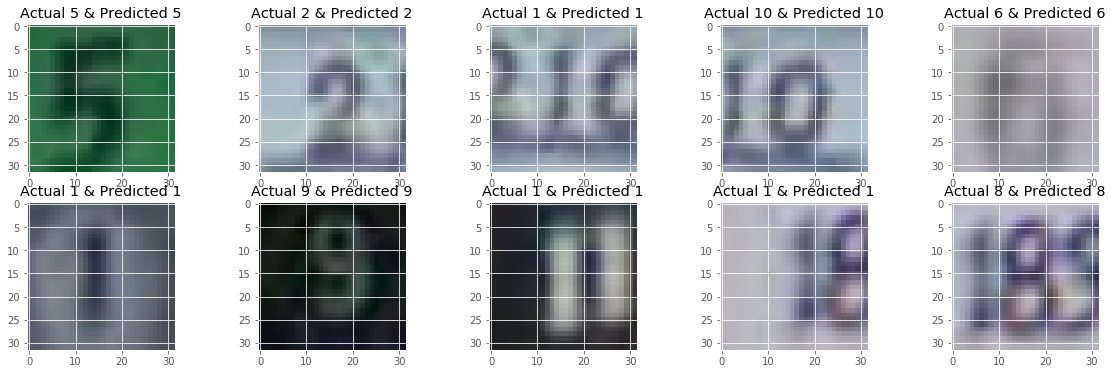


Figure 17: Digit recognition

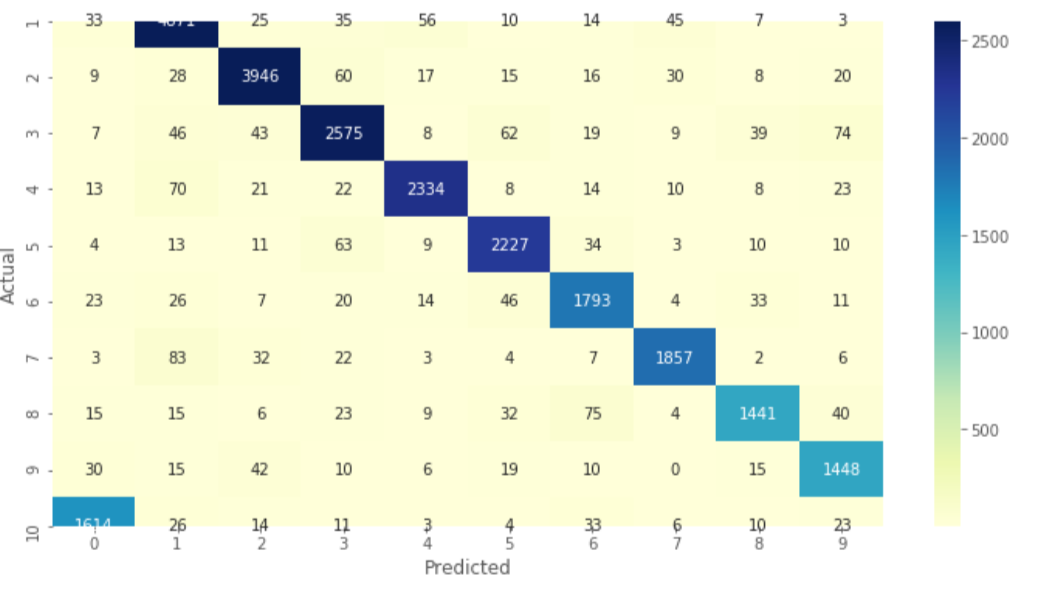


Figure 18: Confusion matrix

**Classification algorithms:** In classification algorithms, we have converted the four-dimensional data to two-dimension data for the PCA to compute on x train data.

* Random forest classifier: The accuracy of the random forest was 51%
* K – nearest neighbors: The accuracy of knn is .48
* Logistic regression is .24

The approach used in this project helped calculate the accuracy by using different frameworks and using greyscale and RGB images. This helped in understanding whether the greyscale image helped in recognizing the digits efficiently. Future work, to be focused on using the weights from MNIST data and use those on SVHM data. Exploring more architectures will help in classifying the original images and using different frameworks like PyTorch or caffee will help run the model.

Conclusion:

* Model – I/P- {C-P-C-P-FLAT-FC-FC}-O/P on Color images with Adam optimizer, batch normalization, and dropout gives accuracy of 91%
* Model – I/P- {C-P-C-P-FLAT-FC-FC}-O/P on Grey images with Adam optimize and dropout gives highest accuracy of 92%%
* A higher number of layers increase the accuracy
* Gray scaling does improve the accuracy of the model
* Batch normalization and dropout improve the accuracy of the model

|  |  |  |
| --- | --- | --- |
| MODELS | Greyscale Images | Non-Greyscale images |
| Convolutional neural network using TensorFlow | 87% | 88% |
| Convolutional neural network using Keras | 92% (Using 3 convolutional layers) | 89% (Using 3 convolutional layers) |
| Convolutional neural network using Keras and add batch regularization | - | 91% |

Table 1

References

* Jaderberg, M., Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Deep structured output learning for unconstrained text recognition. *arXiv preprint arXiv:1412.5903*.
* Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., & Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*.
* Felzenszwalb, P., Girshick, R., McAllester, D., & Ramanan, D. (2013). Visual object detection with deformable part models. *Communications of the ACM*, *56*(9), 97-105.
* Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.