# Verifying Individuality of Handwriting using Deep Learning

Nikhil Shekhar

Sanjana Ramprasad

**Abstract** — This project is to determine an appropriate representation using deep learning to determine whether two samples of handwriting came from the same writer or not for validating the hypothesis that everyone writes differently. The dataset of image contains five samples for the word "and" from people in consideration. The study is built on recent advances in developing machine learning algorithms for recognizing handwriting using deep learning models using TensorFlow. The study consisted of two phases: image representation, and analysis to establish the discriminative power of handwriting. The image representation phase was to crop, merge and create labeled training, validation and test image samples to validate the theory that writing style of one writer can be discriminated from the writing style of another writer.

## I. Introduction

THIS document aims at building a deep neural network model using TensorFlow library using Convolution Neural network, max-pooling and fully connected neural network components. We create the network using the above components and minimize the Cross-entropy cost function using Adam optimizer which uses the Kingma and Ba's Adam algorithm to control the learning rate.

## II. The Dataset and Features

For this project we have a dataset of images, on an average there are five samples of the letter "and" from three pages for each person in consideration. This data was collected from a large sampled population of people from various parts of United States. The entire dataset consists of 15500 grey scale image samples of varying sizes with the horizontal alignment being the same.

## III. Steps for Generating and Testing Model on Dataset

### *A. Pre-process the images to create the feature values and labels from the raw images*

The raw image samples are of different shapes. These images were first converted to a standardized size of 32*64 dimensions and then normalized by subtracting the mean and dividing by standard deviation . After normalization, for each matching pair of samples the images were stacked vertically upon each other to create concatenated image of dimensions 64 * 64. To create the training data, we sampled matched and un-matched images from the all possible combinations possible. The matched training samples was created by generating all possible matching pairs between files such that the first four characters of the filename(representing the person number) were the same. Once the pairs were created, the matching images were segregated into validation and testing. The dataset was partitioned into training data set containing 80% of the data, cross validation data set with 10% of data and remaining 10% is the test data

The same was done for non matching too. To generate non matching pairs, for every image random image files were picked that didn't match the first four letter subset of the filename. The two images were then stacked. The non matching images were split into training and testing the same way as stated above.

This gave us a training sample of 146805 images. Similarly, the validation data comprised of a total of 10226 samples and the test data has a total of 10225 samples.

### *B.File formatting and encoding the data*

The concatenated images obtained as the training, test and validations dataset is encoded into gzipped file using a similar scheme used for encoding and decoding in the MNIST data. The data set is encoded in six separate files, for training images, training labels, testing images and testing labels, validation images and validation labels. All the information is encoded as big endian. The files are in idx format which is ideal for storing vectors and multidimensional matrices dataset.

The idx files are encoded with magic numbers first. The image samples of training and test have a magic number of 2051(encoded as \x00\x00\x08\x03) whereas the labels have a magic number of 2049(\ encoded as x00\x00\x08\x01) The magic number is an integer with the most significant bit first always has the first two bytes as 00 and the remaining bytes represent the type of the data.The file format looks like below.

- Training and test image :

  Magic Number
  Number of images
  Number of rows
  Number of columns
  Pixel
  :
  .
  :
  Pixel

- Training and test label:
  Magic Number
  Number of items
  Labels
  Labels
  :
  .
  :

The pixels are written row wise with values ranging from 0 to 255.

*C.Feeding data into TensorFlow*

The files are read by decoding the gzipped encoding format of the images. The images are read as 32 byte streams. For extracting images, they are reshaped into Number_of_images*height*column*number_of_channels. The number of channels is one since they are all grayscale images. The labels are also decoded using the same decoding scheme. If the one hot encoding parameter is set true. The labels are given out as one hot encoded vectors. In our model we have used labels with one hot encoding.

Images are read in batches and are read in the same order as they are encoded in the gzipped file. However once the index in the epoch to return batches exceeds the number of images in the dataset, it shuffles the entire data set and starts reading the images again in the shuffled order.

*D.Model architecture*

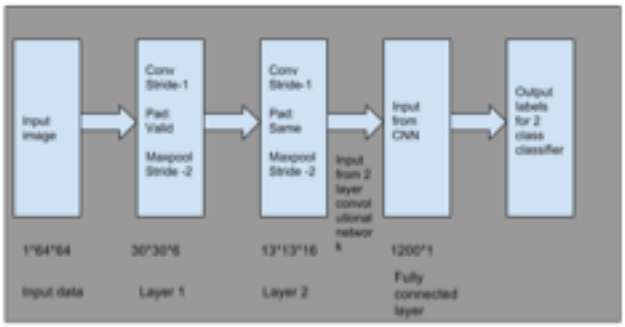


Fig 1.0 - The figure above shows the architecture along with dimensions of each layer.

The final model comprises of a 2 layered Convolution Network followed by a subsampling or pooling layer after every convolution layer , this in turn was connected to a fully connected network. The input images are fed as a 64 * 64 matrix to the deep network The architecture is as described below.

- **Convolution layer 1** - This layer uses a Stride of 1, Filter size is 5* 5. Feature maps of size 60*60*6 are used.
- **Max-pool layer 1** - This layer uses a Stride of 2, output dimension of image is 30*30*6
- A **ReLu** Non-Linear transform is applied to the output of the Convolution.
- **Convolution layer 2** - Stride of 1, Filter size used is 5* 5. Here feature maps of dimension 26*26*16 are used.
- **Max-pool layer 2** - Stride of 2, Output dimension of image is 13*13*16
- **Fully connected layer 1**- The output of the third convolution layer, is transposed and fed into a Fully connected Neural network. The number of features in this layer is 1200.
- **Fully connected layer 2** - The last layer has just two Activation units for a 2-class Classifier. This is the output from the model
- A **dropout** layer is added to prevent overfitting. A dropout value of 0.7 was used.
- **Cross entropy** Logit function is being used as the Cost function.
- **Adam optimizer** used to minimize the Cost function.
- **Batch normalizer** is used after every Convolution layer to ensure that the gradients do not vanish very quick as it would lead to saturation.

*E.Stochastic Gradient Descent Solution*

Calculate the weight vector by using stochastic gradient descent by the use of the equations below

$$w^{(\tau+1)} = w^{(\tau)} + \Delta w^{(\tau)} \qquad \Delta w(\tau) = -\eta(\tau)\nabla E$$

$$\nabla E = \nabla ED + \lambda \nabla EW \ ; \ \nabla EW = w(\tau)$$

$$\nabla ED = -(t(n) - w(\tau)^{\top}\phi(x(n)))\phi(x(n)) \ ;$$

where, $\eta(\tau)$ is the learning rate, and we initialize the weight vector to any random value.There are two weight vectors generated, one for each type of data namely(dw1 and dw2).

*F.Train the hyper-parameters of the model*

For a given group of hyper-parameters such as M, $\mu j$, $\Sigma j$, $\lambda$, $\eta(\tau)$, train the model parameter w on the training set . The parameters $\mu j$ is chosen by using K-means on the training data which returns back the centroid based on the value of M. These centroids are chose as the means for each cluster. M is chosen in way that the ERMS of training data is minimal but taking care of overfitting by plotting ERMS and value of M and choosing M such that the ERMS is stable. The value of optimal M and $\lambda$ is chosen by performing grid search. For this I ran a 2 level nested for loop with varying $\lambda$ and M and finding the combination where the ERMS of the training data is minimum.

*G.Evaluation of the model*

Using the model generated above, calculate the Root mean square error on the validation data to test the model being generated by using the equation below. For this we need to generate the $\varphi$ matrix for the cross validation data sets first and then use the weights calculated in Step E and the same $\mu j$ and $\Sigma j$.

$$E(RMS) = sqrt(2*E(w^*)/N)$$

where, N is the number of size of validation data set.



*H.Tune hyper-parameters of the model*

## IV. Conclusion

*H. Tune hyper-parameters of the model*

Calculate the Root mean square error for the cross-validation data set for both the data sets and check if it is close to the root mean square error of the training data. If the errors are close, then the model is acceptable. But, if not, tune the hyper-parameters by following the steps as in Step F above.

*I. Run the model on test data*

For the real dataset, run the test data through the model generated above and make predictions. The predicted values should be close to the values in the given input file. This determines and validates that the model generated is a good one.

*J. Results*

| Learning Rate | Batch Size | Epochs | C1 feature maps | C2 feature maps | Filter size | Accuracy |
|---|---|---|---|---|---|---|
| 0.01 | 100 | 15 | 6 | 16 | 3x3 | 89.09 |
| **0.01** | **100** | **20** | **6** | **16** | **5x5** | **90.53** |
| 0.01 | 100 | 15 | 4 | 10 | 3x3 | 87.21 |
| 0.01 | 100 | 20 | 4 | 10 | 5x5 | 88.04 |

Fig 1.1 - The table above shows the accuracy achieved based on different filter sizes and feature maps.

The model was tuned with different values of the hyperparameters. It was found that with higher filter sizes and more epochs. The model gave best results for convolutional layers of 60*60*6 and 26*26*16 feature maps with filter sizes of 5x5 and max pooling with strides of 2.

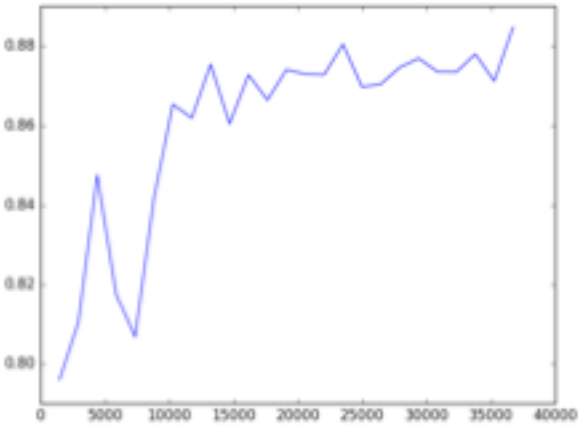


Fig 1.2 - The plot above shows how accuracy changes over time.

## IV. CONCLUSION

Both the Stochastic Gradient Descent and Close form Log Likelihood can be equally useful in creating regression models based on Basis functions. As a general trend, it can be seen that Stochastic Gradient descent takes longer to converge if the hyper parameters are not configured properly, specially the eta parameter.

## V. REFERENCES

[1] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In British Machine Vision Conference, 2014.

[2]Sargur N. Srihari,1 Ph.D.; Sung-Hyuk Cha,2 Ph.D.; Hina Arora,3 M.E.; and Sangjik Lee,4 M.S. Individuality of Handwriting . J Forensic Sci, July 2002, Vol. 47, No. 4.

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, Advances in Neural Information Processing Systems 2 (NIPS*89), Denver, CO, 1990. Morgan Kaufman.