Project - Letter Classification

Project Team Name- Deep Divers
Project Team Members-
Abraham Kong, Akanksha Rawat,
Cory Randolph, Karishma Kuria

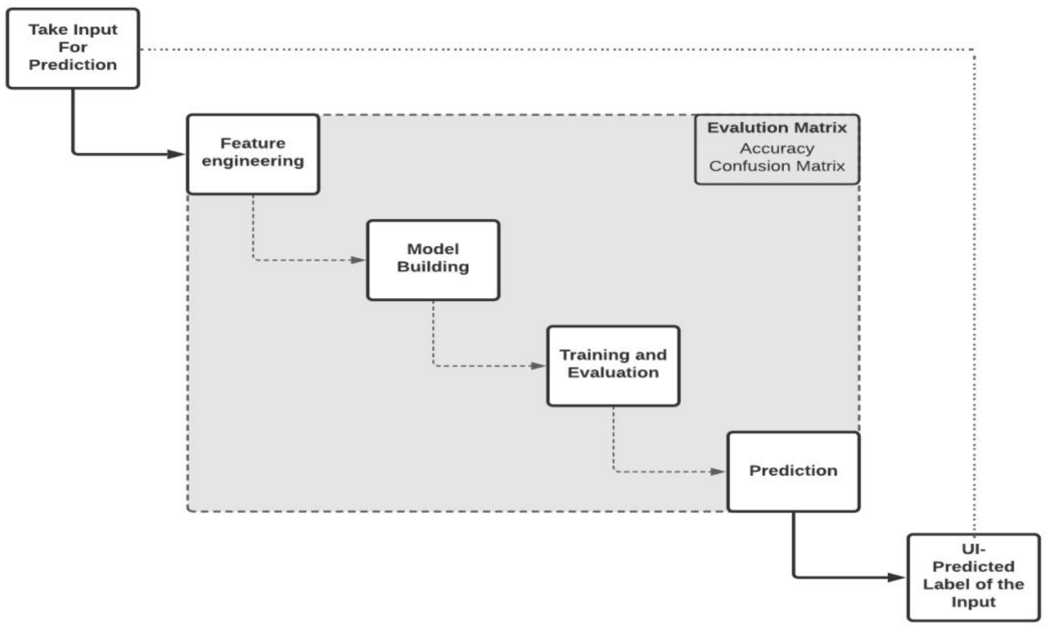
CMPE 255 - Fall, 2021
12/10/2021

1 Abstract:

People spend more and more time on their electronic devices each day, but some still prefer the old fashion way of writing.We are going to build a web application that allows users to write their letters on the whiteboard, and through Machine Learning Models, the application will acknowledge the handwriting and transfer it into a typed-out paragraph. So now the user will be able to enjoy the movement of handwriting while also enjoying the convenience of their electronic device.

2 Introduction:

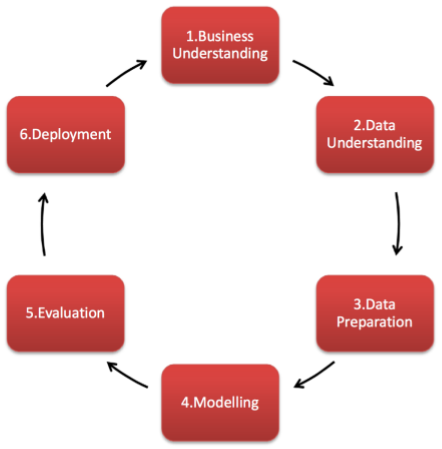
1. Build a whiteboard on a front end UI
2. Take user drawn had written letters
3. Apply Machine Learning Model to classify drawing
4. Return a prediction based on the user's drawn letter



3 Related Work:

TensorFlow.js Crash Course - Machine Learning For The Web - Handwriting Recognition (Link in the reference section), is one of the videos that provides a lot of insight to our project. In contrast to the lecture of only use image processing to include the prediction for numbers (0-9), we go further to includes the alphabet letter (a-z), also, since the video only go through the idea of the model, our project builds an interactive website for the user, so the user can play with the letter detection directly.

4 Methodology



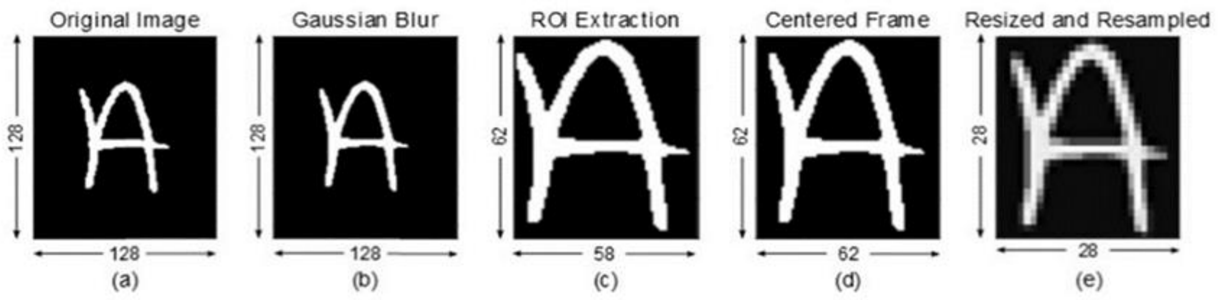
Source

4.1 Business understanding:

Interactive prototype to convert handwritten alphabet to digital alphabet, potential application in the future: transfer hand-written articles to digital typed-out articles.

4.2 Data understanding:

We have imported EMNIST and extracted the balanced dataset. This paper introduced a suite of datasets, known as Extended Modified NIST (EMNIST). The EMNIST dataset has a set of handwritten digits (0–9), (a-z), and (A-Z) being converted into 28x28 pixel pictures. These datasets are intended to represent a more challenging classification task for neural networks and learning systems.
We will use the “Balanced dataset” as our data which has 47 classes.
EMNIST Balanced: 131,600 characters. 47 balanced classes.



source

4.3 Data preparation:

Since the dataset has so many features and all features cannot be used for model training because it reduces the accuracy of the model. In this stage we have used several dimensionality reduction techniques such as PCA, SVD which are described in the later section of the report.
Used certain supervised classifiers and divide the data into test and train. Converted certain numerical labels into letters.
Then did data normalization to make the data fit for our model, here we have divided the data by 255 based on the RGB codes.

4.4 Modeling:

A great way to use deep learning to classify images is to build a convolutional neural network (CNN) based model and The Keras library in Python helps to build this.

The model type that we will be using is Sequential.
We use the ‘add()’ function to add layers to our model.

4.4.1 Model construction

Our model is based on neural networks architecture and built using Keras- high level API of tensorflow framework in multiple layers.

- begin with model type= Sequential()

  model = Sequential()

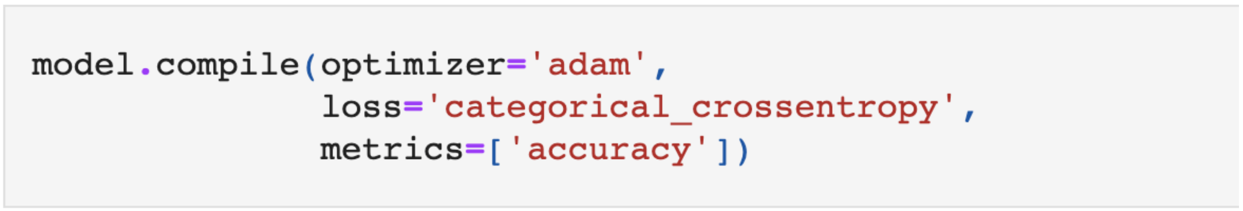
- then consist of layers with their types: model.add(type_of_layer())

  model.add(Conv2D(32,kernel_size=3,activation='relu',input_shape=(28,28,1)))

- Our first layer is the Conv2D layer. These are convolution layers that will deal with our input images, which are seen as 2-dimensional matrices.
- 32 is the number of nodes in the layer.
- Kernel size is the size of the filter matrix for our convolution. So a kernel size of 3 means we will have a 3x3 filter matrix.
- Activation is the activation function for the layer. The activation function we will be using for our first 2 layers is the ReLU, or Rectified Linear Activation.
- Our first layer also takes an input shape. This is the shape of each input image, 28,28,1 as seen earlier on, with the 1 signifying that the images are grayscale.
- ‘Dense’ is the layer type we will use in our output layer. There is a ‘Flatten’ layer. Flatten serves as a connection between the convolution and dense layers.
- The activation is ‘softmax’. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.
- Dropout(0.4) will prevent our network from overfitting thus helping our network generalize better.
- Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1

4.4.2 Compiling the model

After constructing we complied our model and used these below parameters [optimizer, Loss, Metrics]



```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

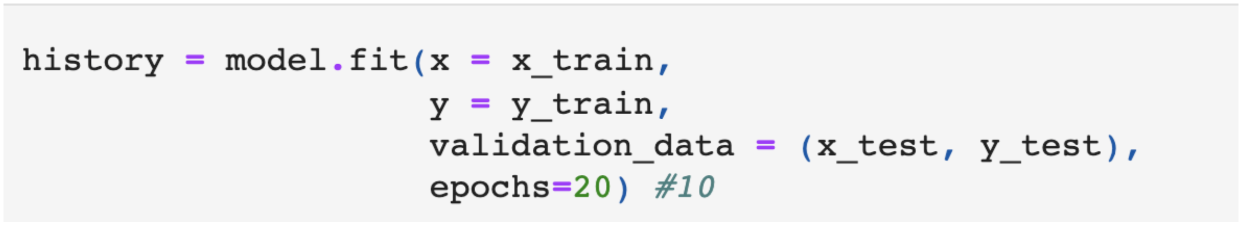
After adding a sufficient number of layers the model is compiled. At this moment Keras communicates with TensorFlow for construction of the model.

4.4.3 Training the model

To train our model, we used the ‘fit()’ function on our model with the following parameters: training data ,target data , validation data, and the number of epochs.

For our validation data, we have used the test set provided in our dataset, which we have split into x_test and y_test.

The number of epochs is the number of times the model will cycle through the data, we haved 20 here.

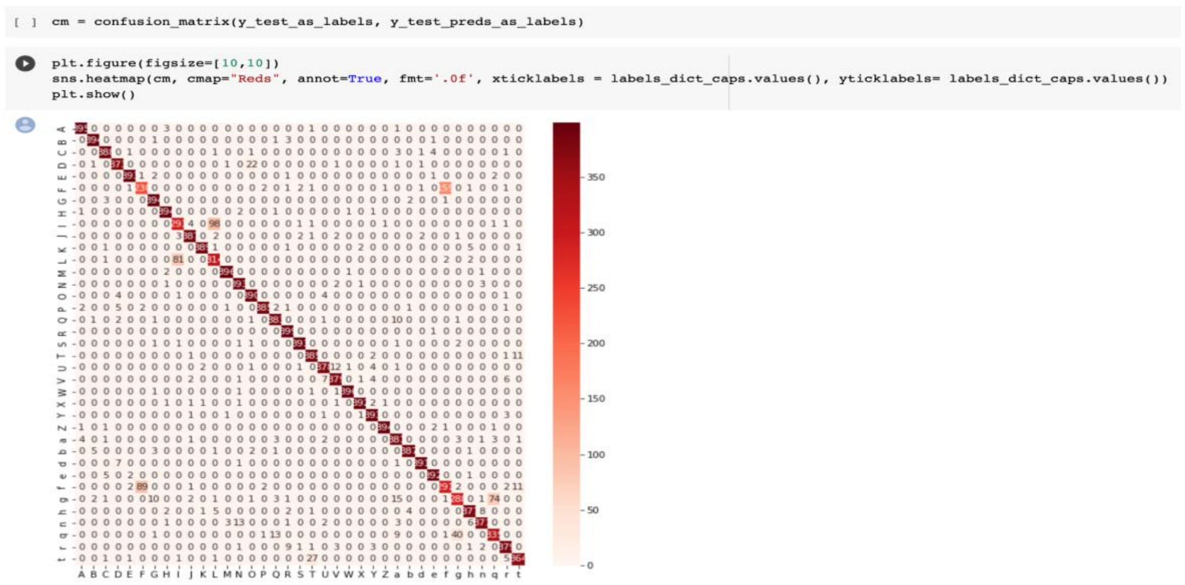


```
history = model.fit(x = x_train,
                    y = y_train,
                    validation_data = (x_test, y_test),
                    epochs=20) #10
```

4.5 Evaluation:

We have evaluated on a test data set using below measures:

- Confusion Matrix
- Accuracy
- Loss
- Precision
- Recall
- f1-score



```
[ ] cm = confusion_matrix(y_test_as_labels, y_test_preds_as_labels)
```

```
plt.figure(figsize=[10,10])
sns.heatmap(cm, cmap="Reds", annot=True, fmt='.0f', xticklabels = labels_dict_caps.values(), yticklabels= labels_dict_caps.values())
plt.show()
```

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b d e f g h n q r t
350
300
250
200
150
100
50
0

4.6 Deployment:

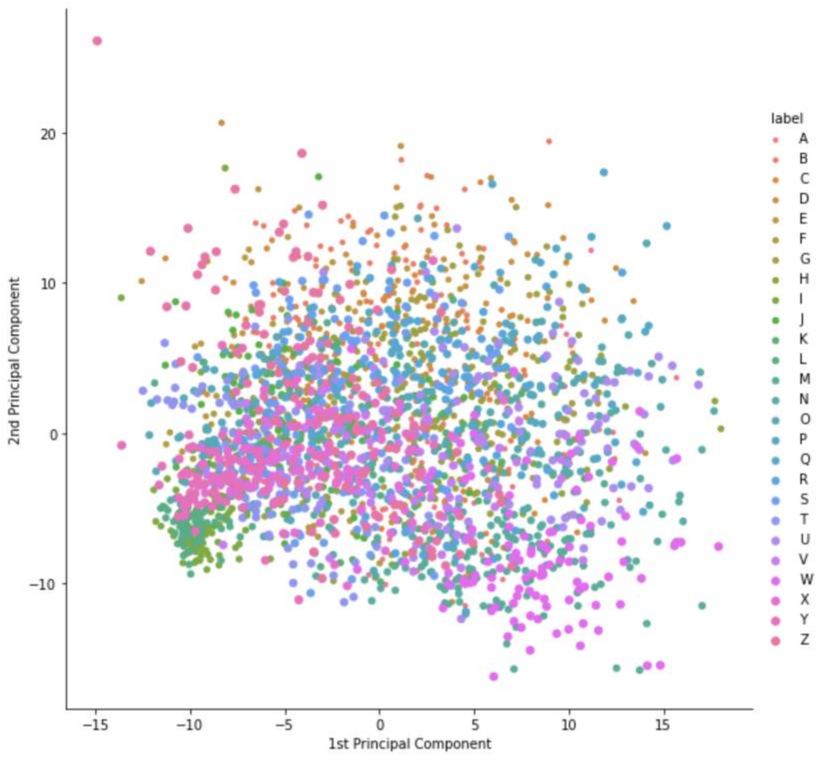
The first step in deployment was to create a simple user interface that would allow users to interact with our project and model by drawing their own letters and then receiving live predictions based on the trained model.
The second step was to deploy this model as a working web application, through PaaS Heroku. We chose to do this through Heroku since it integrates well with Github and Streamlit (front end software).

5 Methods:

Following Dimensionality reduction techniques are used:

Principal Component Analysis (PCA):

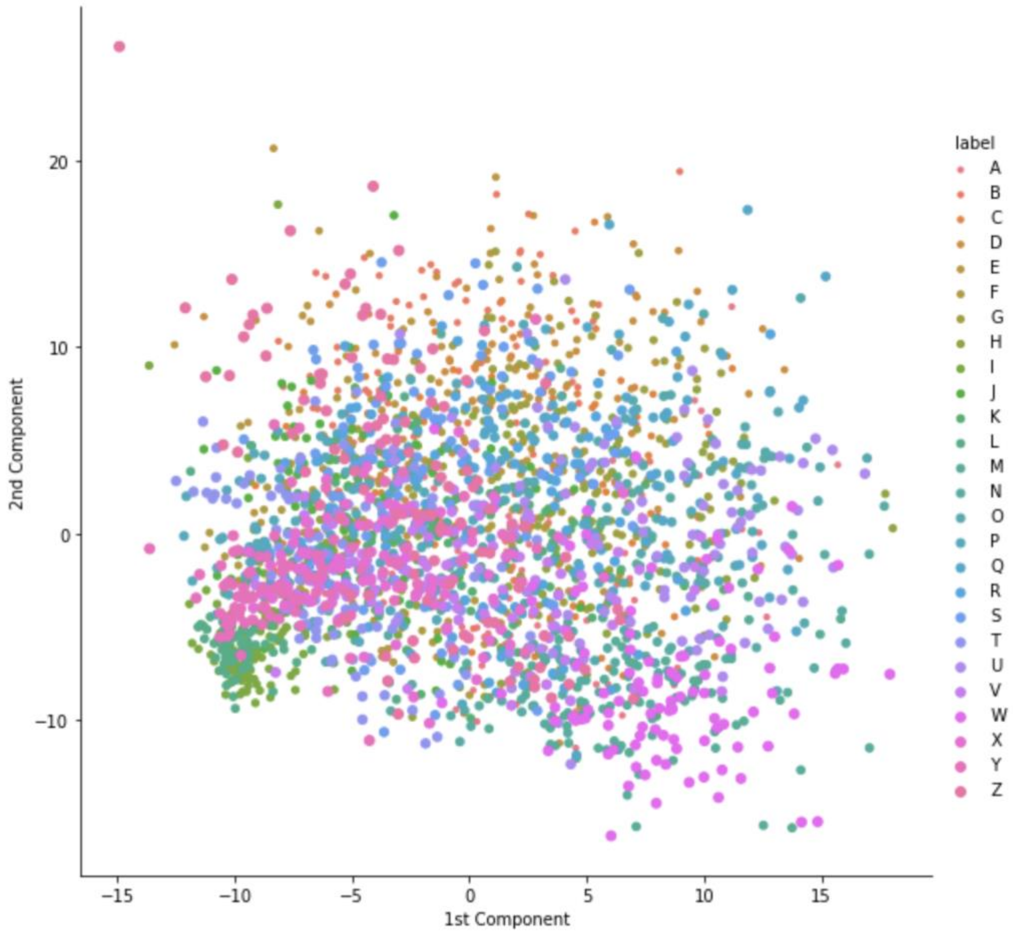
PCA is an unsupervised linear dimensionality
reduction technique that helps us to identify patterns in data based on the correlation between the features.
PCA aims to find the directions of the maximum variance in high dimensional data and project it onto a lower dimensional feature space. The below plot shows the result of PCA.



The above plot shows a lot of overlapping between classes. Very few classes are distinguishable from the cluster and most of them are mixed. So this clarifies that PCA is not very good for high dimensional datasets.

Singular Value Decomposition (SVD):

SVD is also a linear dimensionality reduction technique. Very similar to PCA, but it doesn't center the data before computing the singular value decomposition. This means it can work with sparse matrices efficiently. It can be explained as a projection method, where data with ***m*** features can be projected into an ***m*** or less dimensional space while retaining the complete essence of the data in the original dataset.



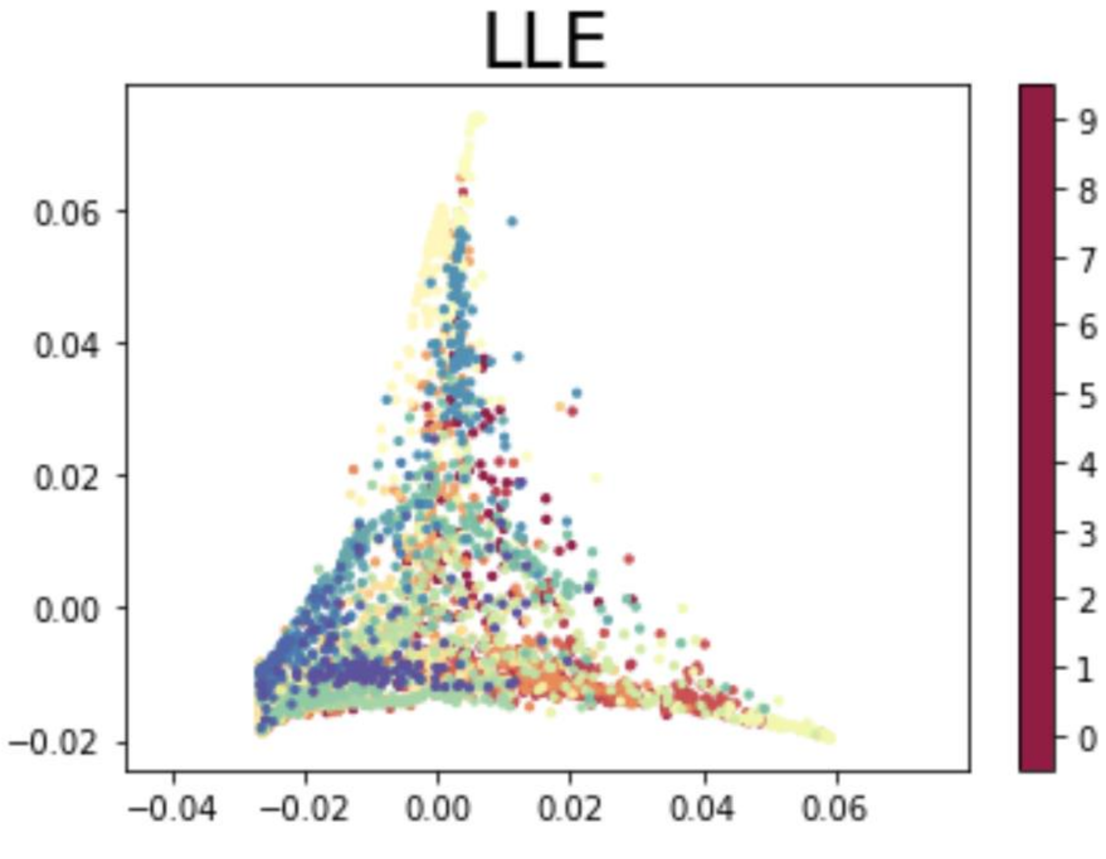
The plot above shows the result of SVD. It’s clear from the plot that SVD has performed better as compared to PCA but yet again the classes are not so clearly separated and still contain too much overlapping. Since both PCA and SVD are linear projection techniques, they cannot properly capture nonlinear data dependencies such as in case of MNIST letters.

Locally Linear Embedding(LLE):

LLE is an unsupervised and non-linear dimensionality reduction technique that creates an embedding of the dataset and tries to preserve the relationships between all the neighborhoods in the dataset.

LLE can be thought of as performing a series of local PCAs that are globally compared to find the best non-linear embedding. When the data is primarily very high dimensional it cannot be described linearly with just a few components.

With these cases, nonlinear manifold embeddings like LLE and Isomap can be helpful.

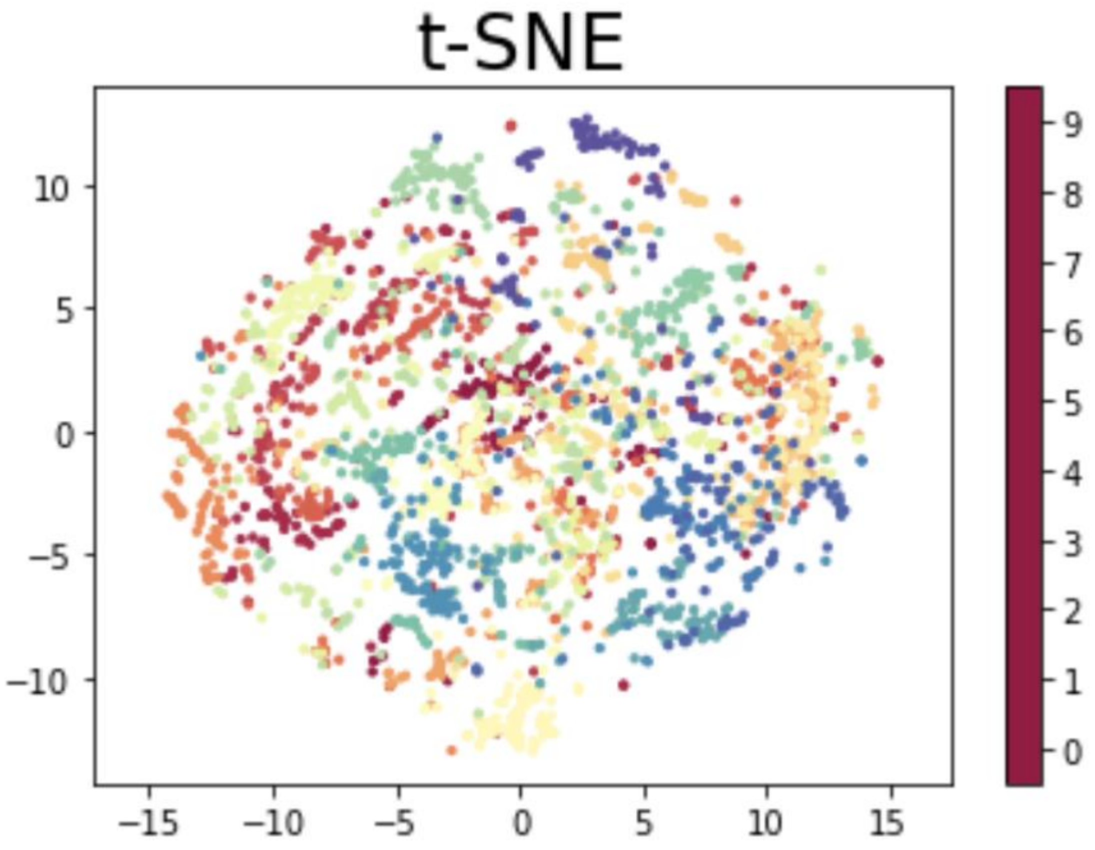


The plot above shows the result of LLE applied to the MNIST dataset. This is comparatively better then the above techniques but still has not clearly clustered the data.

T-distributed Stochastic Neighbor Embedding (t-SNE):

T-Distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction algorithm used for exploring high-dimensional data. T-SNE maps multi-dimensional data to 2 or more dimensions that are suitable for human observation. t-SNE is different from PCA since it only preserves small pairwise distances or local similarities whereas PCA is concerned with preserving large pairwise distances to maximize the variance.

In other words, things that are different end up far apart. This can lead to poor visualization especially when dealing with non-linear manifold structures such as cylinder, ball, curve as we saw in the results of PCA.

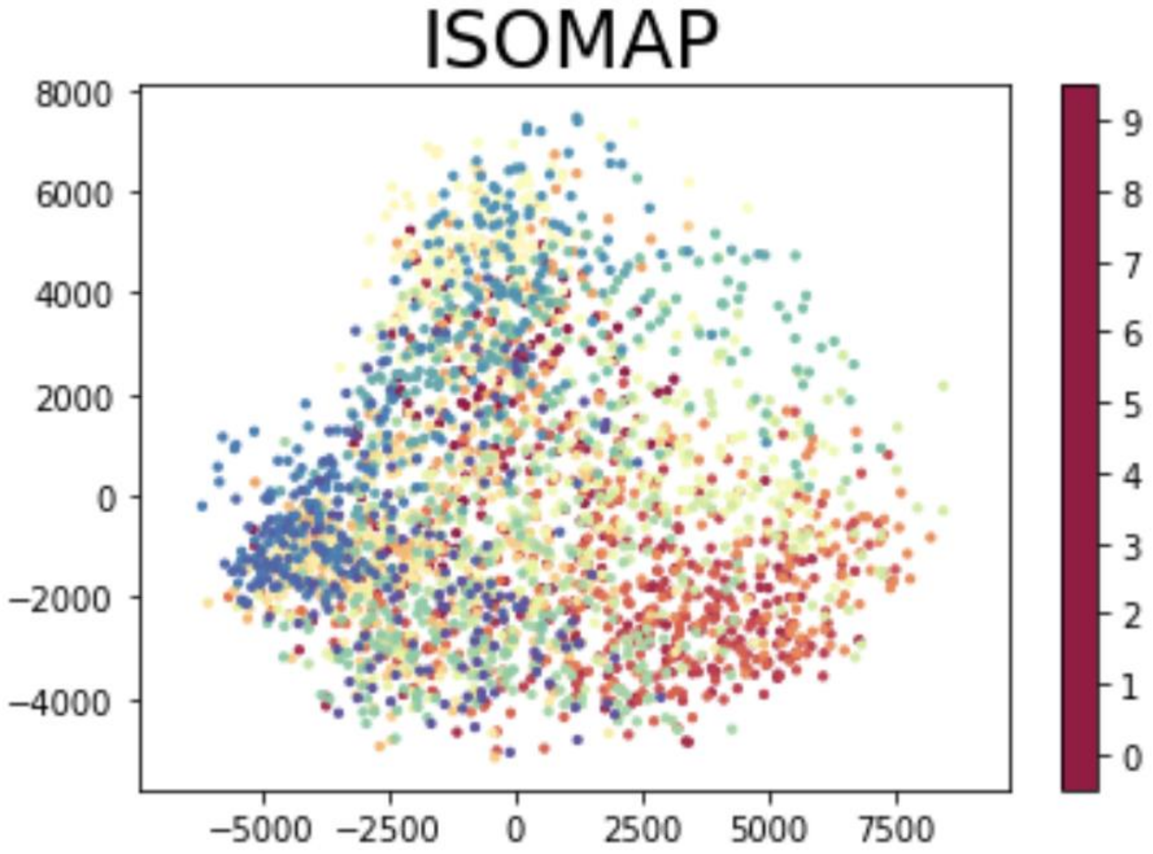


It can be concluded from the above result of T-SNE on MNIST dataset that it has very nicely Clustered the data as compared to the other techniques used before. It performs well for non-linear data. The only problem with T-SNE is that it takes more time as compared to the above techniques. So one better way of doing this could be performing PCA before T-SNE.

Isometric Mapping (ISOMAP):

It is a non linear dimensionality reduction technique. The steps involved in it are as follow

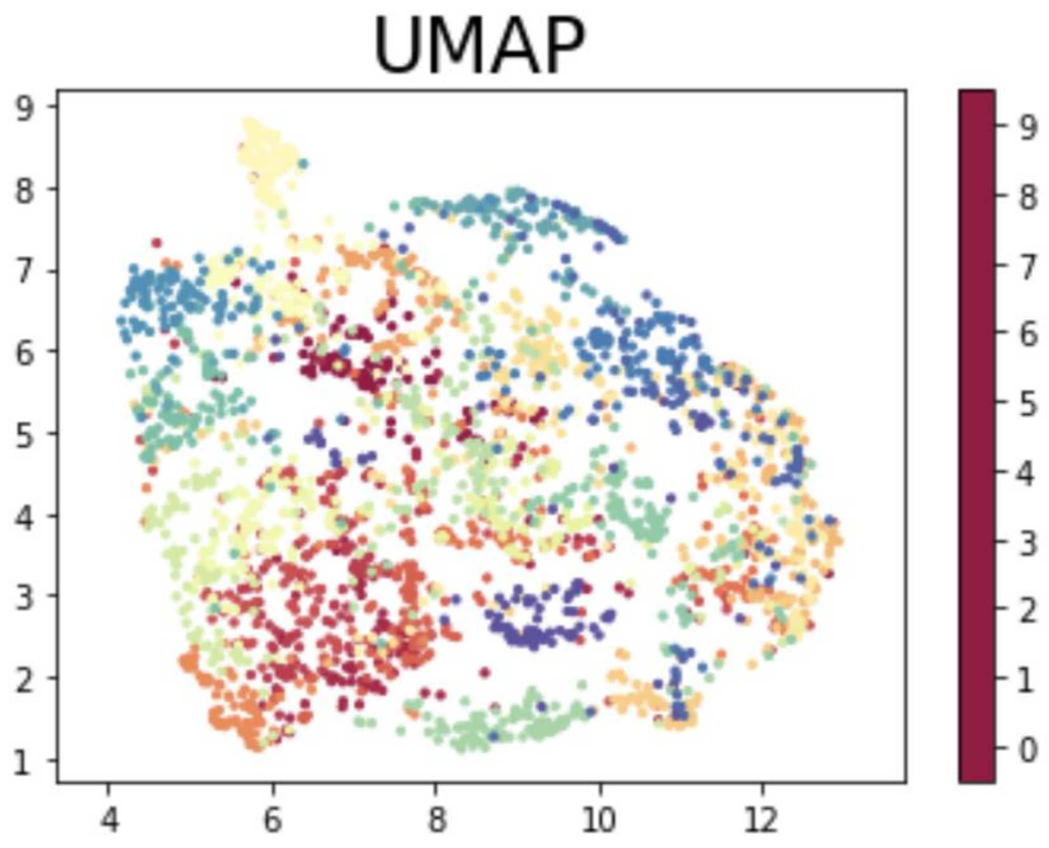
1. First we find the nearest neighbours of each point and then create a weighted graph by connecting each point to its nearest neighbour. The nodes in the graph represent the data points and the weights are the distance between them.
2. Now redefine the distance between the points to be the length of the shortest path between 2 points in the neighbourhood graph.
3. Lastly Apply MDS to the newly created distance matrix.



The above plot shows the result of ISOMAP on our dataset. It shows the digits are better separated as compared to the linear techniques like PCA. Although not fully separated and clear.

Uniform Manifold Approximation and Projection (UMAP):

Uniform Manifold Approximation and Projection (UMAP) is a dimensionality reduction technique that can preserve as much of the local, and more of the global data structure as compared to t-SNE, with a shorter runtime.

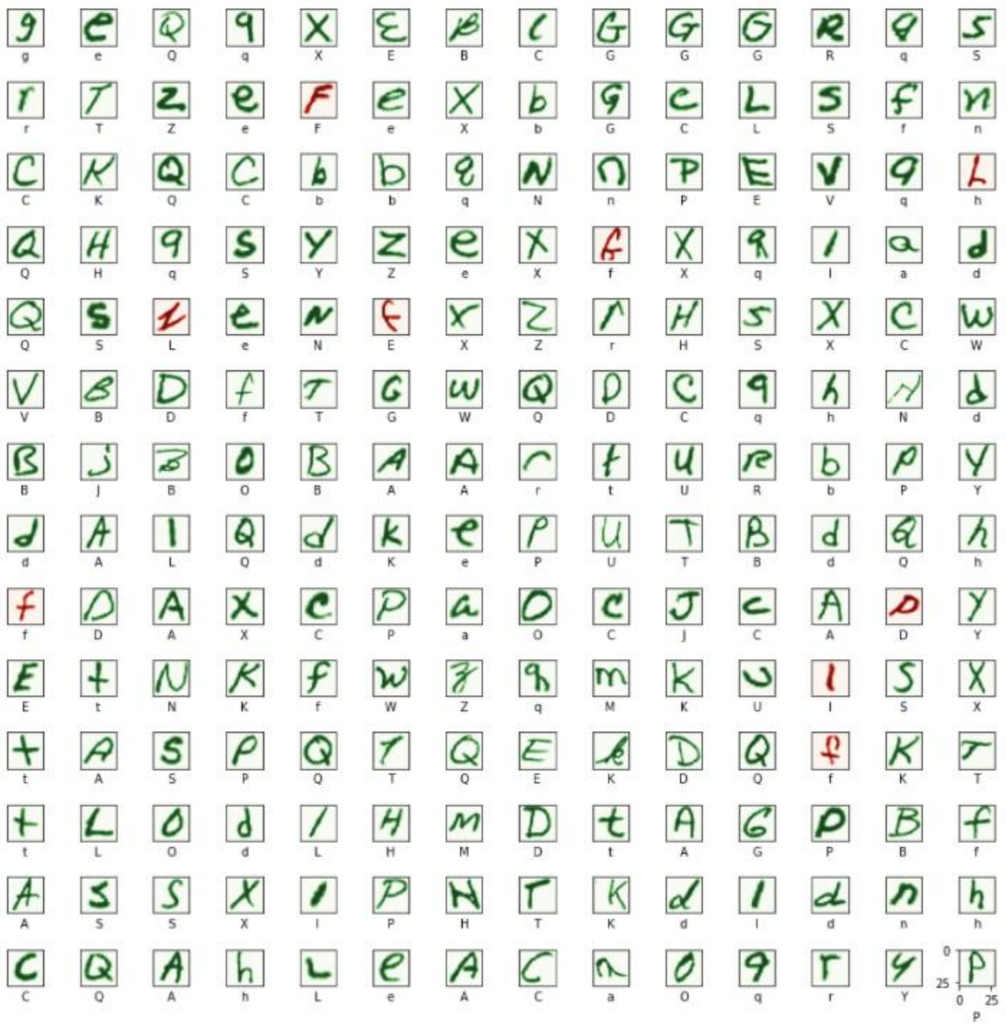


From the above result plot it is clear that UMAP has outperformed all the techniques used in the dataset before. It has done a pretty good job in compressing the digits while retaining the useful information in data and the clusters look clear.

6 Experiments and Results:

We have an accuracy of 92.95% and our model can predict almost all of the uppercase and lowercase letters, except some letters that look similar in both upper and lower case, like ‘C’ and ‘c’ or ’X’ and ‘x’.

The below graph is a layout of examples of prediction we tested out on the model. Green being correctly predicted and red benign falsely predicted.



7 Conclusion:

By gaining a deep understanding of the data through our data understanding and preparation phases, we were able to find better resources for building out the Machine Learning Model that would help provide accurate predictions for hand draw digits. The end result was that we have a model that performs well on real work handwritten letters and and has a 93% accuracy on the EMNIST test data set and this largely because of the confusion between “F” and “f” and “I” and “l”, otherwise the rest of the letters have ~98% accuracy.

During the model design and application development we learned how important it was to find similar examples and use those insights as stepping stones towards our final project. For example the reshaping and transformations of the data we did in the discovery phase allowed us to incorporate user drawings in the application.

One of the potential applications of our project can be to use mobile device cameras to transfer hand-written documents to digital documents. There are still many people preferred writing down their ideas instead of typing it out, We can let the user take a picture of his or her hand-written notes and the system will process their image taken through our model, and output a digital-typed document back for the user.

8 Supplementary Material:

Data Set References

EMNIST dataset: https://www.nist.gov/itl/iad/image-group/emnist-dataset

Direct download: http://www.itl.nist.gov/iaui/vip/cs_links/EMNIST/gzip.zip
Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373
Importing and formatting Image data inspired by AranguranAndres/EMNSIT-Image-classification
Mapping and original file reference Website

Data Reduction References
TensorFlow.js Crash Course - Machine Learning For The Web - Handwriting Recognition Video
Parts of the visualation nad PCA were inspired by Rahul228646's Kaggle notebook
Example Data Visualizations for Images from Kaggle Notebook

Classification Reference
Image Classification in 10 Minutes with MNIST Dataset Article
How to Develop a CNN for MNIST Article

Deployment Reference
Building a VAE Playground with Streamlit Article
Heroku deployment without the app being at the repo root (in a subfolder) Article
timanovsky's subdir-heroku-buildpack github Github