

# AMATH 582 Final Project

Haoran Zhao, Hongda Li, Shuangshan Li

March 19, 2021

## Abstract

In this paper, we explore different techniques by analyzing the EMNIST data set. Principal component analysis (PCA) and Linear Discriminant Analysis (LDA) are used to reduce dimensionality of the data set, then three types of classifiers, support vector machine (SVM), Decision Tree Random Forest, and Neural Network are used to classify these data set. The objective is to understand the models better, both in a practical and a theoretical sense.

## 1 Introduction and Overview

The objective of the project is to explore and understand the EMNIST data set using various Machine Learning models and dimensionality reduction techniques. Mathematical understandings were also involved which seek to better understand the behavior of the models, the performance of the model, and derive understanding of the data set from behaviors of the models. LDA and PCA are used as dimensionality reduction techniques, SVM, Random Forest and Neuro-Nets are used as predictive models.

### 1.1 The EMNIST Data set

The EMNIST data set is an extension of the MNIST data set. It consists of labeled handwritten digits[2], lower-cased and upper-cased letters while supporting different types of grouping.

For our purposes, the "by class" grouping has been chosen, where all 62 labels are grouped, put into lexicographical order to match integers labels. The first ten integers are mapped to the 0 to 9 integers, 26 to 52 are mapped to the Capital Letters, and the 52 to 61 integers are mapped to the lower cased letters. The overall ENMIST data set is partitioned into 2 groups, one for training and one for testing. If not specified, results produced are sampled from the test group.

Images for the handwritten letters and digits are normalized and framed under a  $28 \times 28$  pixel image. For each single sample, they are standardized.

### 1.2 Training of Models and Irreducible Errors

Training on multiple Classical Machine Learning models and modern machine learning models on the dataset suggests the existence of irreducible error, which arises in the form of ambiguity between a certain class of digits, lower-cased and upper-cased letters. For example, "2,z,Z", "5sS", "0oO", etc. Each of these symbols belongs to a different label but it's hard to distinguish when written by hands.

These ambiguous symbols not only create difficulties for modern machine learning models such as neural network, but it also creates confusion for classical models such as the SVM and Random Forest. Errors for these classical models suggest potential difficulties for other models such as neural network or other variance of it.

## 2 Theoretical Background

In this section we will go through several of the popular machine learning models, dimensionality reduction techniques and measurements of performance. Major part of the discussion will centered around the

mathematics and idea behind them. The presentation is made with the hope to be informative about the mathematical background of the project, and how the mathematical understandings leads to a better understanding of the structure and topology of the data via training and validating Machine Learning Models. Our hypothesis is that, the performance Neural Net works, or Convolutional Neural Network is at least as good as classical. In addition, we hypothesize the existence of irreducible errors in the EMNIST data set.

## 2.1 Data Standardization

Data standardization is used for the classical training method. This is necessary because models such as the SVM, LDA, PCA, and Decision Tree have biases depending on the samples' size for each of the classes. Hence, all the data is Standardized by flattening the image of the data, and for all features of each sample is set to zero mean and it's normalized such that all features lies in  $[0, 1]$ . In addition, for training SVM and Decision tree, there is the exact same number of samples presented for each of the labels.

## 2.2 Dimensionality Reduction Using Fisher's LDA

Principal Component Analysis and Linear Discriminant Analysis are used together to create low dimensional embeddings for the feature space. The original feature space for each of the image lies in a 784 dimension because each pixel of the image is considered a feature and the image has size  $28 \times 28$ .

LDA is both a classifier and a dimensionality reduction technique, here we introduce Fisher's LDA, the idea is to look for a subspace (Not necessary orthogonal) such that, the embeddings of all the samples in the training set has maximal spread and minimal deviance among all the class label. One a high level, the problem is phrased as [10]:

$$\max_{\|v\|=1} \left\{ \frac{v^T S_{\text{Between}} v}{v^T S_{\text{Within}} v} \right\} \quad (1)$$

2 of the matrices are defined as:

$$S_{\text{between}} = \sum_{i=1}^k ((u_i - \mu)(u_i - \mu)^T) \quad (2)$$

$$S_{\text{within}} = \sum_{i=1}^k \left( \sum_{x_i \in C_i} (x_i - m_i)(x_i - m_i)^T \right)$$

Where quantity  $\mu_i$  denotes the average of the class  $C_i$  projected onto the chosen subspace  $v$ , and the quantity  $m_i$  is the centroid of all the samples for the class  $C_i$ :

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} v^T x_i \quad \mu = \frac{1}{k} \sum_{i=1}^k \mu_i \quad (3)$$

$$m_i = \sum_{x_i \in C_i} x_i \quad (4)$$

Notice that equation (1) is the key here, the solution to the optimization problem is Eigenvector of the maximal Eigenvalue for the matrix  $S_{\text{within}}^\dagger S_{\text{between}}$ , however, the solution is not unique, all Eigenvector can be projected onto, and this is the subspace of embeddings for the LDA.

Notice that taking the inverse of the matrix to be full rank, and LDA can only project the feature space down to  $\min(k, n) - 1$  where  $k$  is the number of classes and  $n$  is the total number of the samples.

However, the drawback for includes the fact that it assumes each class to have Gaussian Distributions of the same Covariance Matrix [14] to get the best separation.

## 2.3 PCA Dimensionality Reduction

PCA: Principal Component Analysis is an unsupervised learning technique and it's usually used as a dimensionality reduction technique. The Singular Value Decomposition exchange any matrix  $A$  into the

product of 3 matrices:  $U\Sigma V^T$ , where  $U, V^T$  are orthogonal matrices and  $\Sigma$  is a diagonal matrix with Singular Values on the diagonal.

In practice, matrix  $A$  is the data matrix and the lower dimensional embedding are the  $\Sigma V^T$  if  $A$  is a column data matrix and  $U\Sigma$  if  $A$  is a row matrix.

PCA is very different for one reason, it supports non-linearity and it's based on the idea of explaining variance in the data set. And nothing can demonstrate this idea better than the Low-rank Approximation theorem[11]:

$$U_k \Sigma_k V_k^T = \underset{\text{rank}(X)=k}{\operatorname{argmin}} \left\{ \|X - A\|_F^2 \right\} \quad (5)$$

The  $k$  truncated singular value decomposition of matrix  $A$  is the solution to the  $k$  rank approximation problem under the Frobenius Norm (The norm measures the variance across differences between these 2 matrices).

This is incredibly useful in a sense that, if the variance of the dataset can be explained by a low rank matrix, then those structure will be revealed via the Principal Components, the  $U$ , or the  $V$  matrix (Depending on whether it's row or column data matrix).

Principal Component is very beneficial for describing the data in an orthogonal subspace that is much lower than the original features space where the data is located int. The idea is so powerful that, Randomized SVD and Sparse SVD algorithm is implemented in the Sklearn Library[7].

However, compare to the Fisher's LDA embeddings, PCA is lossy once the truncated SVD is used. Besides, SVD is sensitive to noise and unitary transformation of the original data[12] (Preservation of Inner products). Rotated samples are very likely to create more singular values. Here we take the conservative approach of choosing 90% or more energy for singular value by default.

## 2.4 LDA + PCA for Dimensionality Reduction and Data Mining

Our idea is combining both PCA and LDA for dimensionality reduction. We use PCA and LDA to first visualize the separations of data using the lower-dimensional embeddings to get some ideas about the dataset. Classical Machine Learning models such as the SVM and LDA have complexity (Training and predicting complexity) that scales with the number of features, samples, and labels involved. Hence creating embeddings in a lower dimension is extremely beneficial to the running time of these classical models' algorithms.

To get the best out of both algorithms, PCA is performed on images partition by labels into equally sized classes, and then LDA is performed on the PCA embeddings of the data using the training set. Implementation is easily carried out using *sklearn*.

When making predictions with a model, the testset is projected onto the PCA components and then transformed into the LDA sub-space before feeding into the prediction model.

## 2.5 SVM

The idea behind SVM: Support Vector Machine is simple and powerful. The problem is formulated as a Quadratic Optimization problem, summarized as the following:

$$\min_{\beta, \beta_0} \left\{ \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^N \xi_i \right\} \text{ s.t: } \begin{cases} y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i & \forall i \\ \xi_i \geq 0, \sum_i (\xi_i) \leq 1 - \xi_i & \forall i \end{cases} \quad (6)$$

The quantity  $\beta, \beta_0$  are parameters that define the separating hyperplane, and the slack variables  $\xi_i$  are used as a penalty for placing the samples on the wrong side of the hyperplane. In practice, the problem is put into a dual primal format, and then optimization on the KTT (Karush-Kuhn-Tucker conditions) is used (e.g: Interior Points Method).

And, buy consider the Fenchel Transform on the primal objective of the function (Which is itself), we obtain the Lagrangian of the system in the following form:

$$\mathcal{L}(\beta, \beta_0, \xi, \alpha) = \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i [y_i(x_i^\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^N \mu_i \xi_i \quad (7)$$

And the constraints of the primal problem is relaxed using the positivity assumption on the variables:  $\alpha_i, \mu_i, \xi_i \geq 0 \forall i$ .

Much more theoretical Perspective of this part is highlighted in the book *Elements of Statistical Learning*[15], where taking the derivative on the Lagrangian wrt the primal variable  $(\beta_i, \xi_i)$  will yield the results that:

$$\begin{aligned}\beta &= \sum_{i=1}^N \alpha_i y_i x_i \\ 0 &= \sum_{i=1}^N \alpha_i y_i \\ \alpha_i &= C - \mu_i \quad \forall i\end{aligned}\tag{8}$$

Often in practice, many noble algorithms are involved in solving the SVM optimal solutions. Most of them are put into the dual form because it has a connection to the Kernel Function.

In practice, SVM is very effective under the task of binary classification for a small sample but much worse for when features space is in a very high dimension and for multi-class classification where many labels are involved. This is the case because of the complexity of the algorithm scales with both dimension of features space, the number of samples and the number of labels. The complexity is given by  $\mathcal{O}(mn^2)$  where  $n$  denotes the number of samples and  $m$  denotes the number of features[8].

Dimensionality technique introduced previously to improve the speed of the SVM. In addition, SVM serves as a great tool for looking into the structure of the data and highlighting potential difficulties for other machine learning models, it serves as a “Lower Bound” for how well other machine learning models can perform. Finally, via experiments, it's a model that hardly over-fit and under-fit is indicative of potential irreducible errors on the data(Assuming that, no abstract features exist for Neural Networks to look for).

## 2.6 Decision Tree and Random Forest

The idea behind the decision tree is yet, another simple and powerful idea. There are 2 types of decision tree, regression tree where it seeks to approximate a function and the classification tree for classifying the samples given labels for training

Under the most basic premise, each node of the tree splits the samples into 2 sets trying to maximize the information gain when splitting which is measured by entropy. The entropy of the tree can be measured by the expression:

$$E(p) = \sum_{i=1}^k -p_i \log(p_i)\tag{9}$$

Where the quantity  $p_i$  denotes the portion of samples with label  $i$  and  $k$  denotes the total number of samples in the current node of the tree. Observe that the function reaches maximum when one of the labels saturated the node of the tree.

To make a branch on the tree, a separation criterion is made on one of the feature, and it seeks to maximizes the information gain, which can be summarized as one expressions[13]:

$$E(\text{Parent}) - \frac{N_{<k}}{N} E(F_1 < k) - \frac{N_{\geq k}}{N} E(F_1 \geq k)\tag{10}$$

Where, the number  $N_{<k}$  is the total number of samples in the left node of the parent node and  $N_{\geq k}$  is the total number of samples in the right node. And the expression  $E(F_1 < k)$  denotes the condition energy of the left node with all the sample where their feature  $F_1$  satisfies the criterion:  $F_1 < k$ , it's the same story on the right side of the node.

The simplicity of the decision tree gives it very large variance on the model, which means that depending on which feature is chosen to split and hyper parameters for the model, many different trees can be configured. In general, the construction of the optimal tree is a NP-Hard problem. Hence usually, heuristics are used for constructing the tree given a training sample[9].

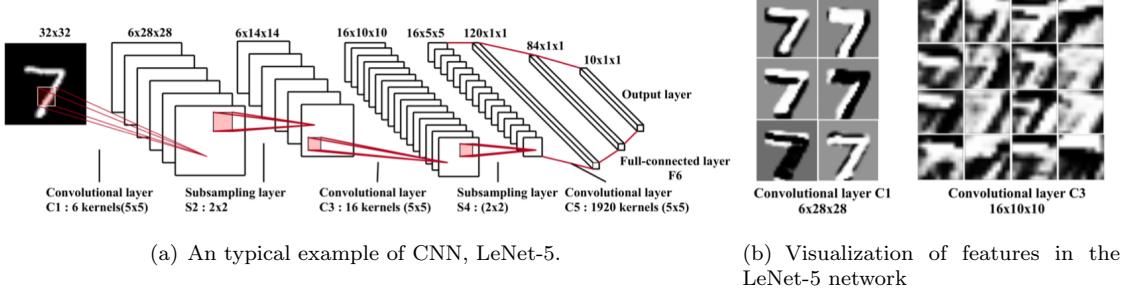


Figure 1: A typical CNN LeNet-5

However, the power of tree is manifested by a forest. The idea is using bootstrap to train multiple tree and use the idea of bagging, where we either takes the average of the prediction from the group of trees trained from the data (Committee Method), or we weight their votes by penalizing trees that are more complex (Stacked Generalization). When the tree is a classifier, we simply take the votes to produce the predicted label[16].

Nonetheless, one of the Random Forest Classifier's major practical advantages over the SVM is the speed of training, which comes from our observation. The algorithm allows the tree to be trained concurrently, reducing the amount of time needed to train the Random Forest.

## 2.7 Convolutional Neural Network

The Convolutional Neural Network is a deep-learning architecture that has been extensively studied in the last few years. The CNN model is designed for multi-class classifications. There are many variants of CNN architectures but their basic components are very similar. One typical example is the famous LeNet-5 shown in Figure 2 which consists of three types of layers, namely convolutional, pooling, and fully connected layers.

### 2.7.1 Convolutional Neural Network Theory

The convolution layer is made of multiple convolution kernels that learn feature representations of the inputs and generate feature maps. A feature map is the result of first convolving the input with a learned kernel, and next, by applying on the convolved results an element-wise non-linear activation function.

In mathematical therm, the feature value,  $z_{i,j,k}^l$ , at a certain location  $(i, j)$  in the  $k^{th}$  feature map of the  $l^{th}$  layer is[3]:

$$z_{i,j,k}^l = \mathbf{w}_k^l \mathbf{x}_{i,j}^l + b_k^l \quad (1)$$

where  $\mathbf{w}_k^l$  and  $b_k^l$  are the weight vector and bias term of the  $k^{th}$  filter of the  $l^{th}$  layer respectively. The activation function introduces non-linearity to CNN, which is important for multi-layer network to detect non-linear features. Let  $a()$  denote the non-linear activation function, it applies to the convolutional feature  $z_{i,j,k}^l$ :

$$a_{i,j,k}^l = a(z_{i,j,k}^l) \quad (2)$$

Typical activation function are sigmoid, tanh and ReLU. In this project we use ReLU which is:

$$\text{ReLU}(X) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} = \max(0, x) \quad (3)$$

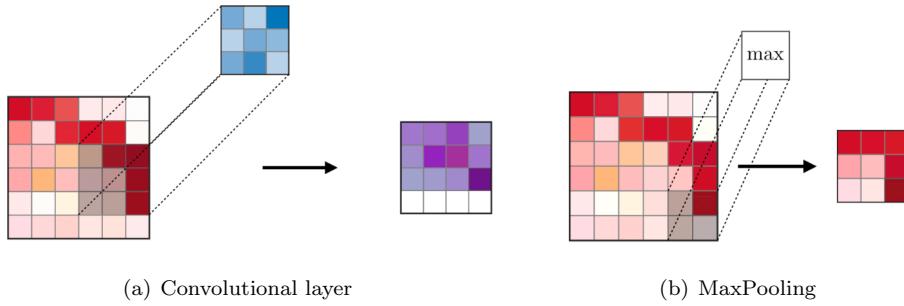


Figure 2: Different layers

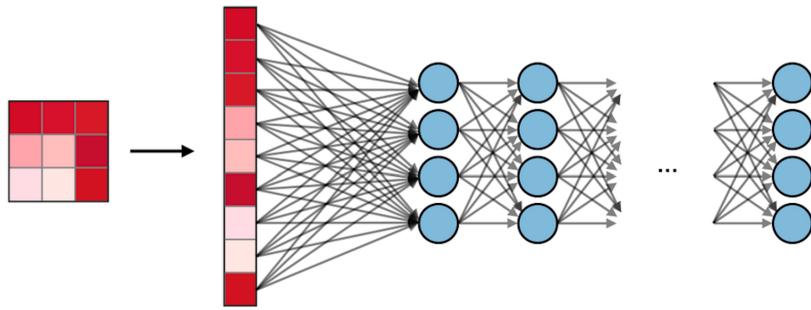


Figure 3: Illustration of fully connected layer

### 2.7.2 Network Architecture

**The convolution layer**(CONV) is the core of any CNN. It uses for extracting information from its input by using several filters that are automatically taught to detect certain features in an image. The user determines the filters' size and their numbers. Each filter will scan through the input from the upper left hand side corner to the bottom right hand side corner, each creating a feature map. The neurons at the output are arranged in a volume with a depth equal to the number of filters, a height equal to  $h_i - h_f + 1$  (where  $h_i$  is the input height and  $h_f$  is the filters height) and a length equal to  $l_i - l_f + 1$  (where  $l_i$  is the input length and  $l_f$  is the filter length).

**The pooling layer** performs the down-sampling in the width and height, reducing the dimensions of its input and, hence, reducing the number of parameters to be computed. This reduces the complexity of the network and the possibility of overfitting.

**The flatten layer** is used to change the shape of the input, making it an array of 1 neuron depth and height, equal in length to the product between the length, depth and height of the input to that layer. This layer is used in every CNN because the output layer must be a one-dimensional vector.

**The dropout layer** is used to reduce overfitting by randomly cutting off a fraction of the nodes in the network. This random dropping of neurons in the network can be used to simulate a great number of different architectures which leads to a better generalization of the CNN.

**The densely connected layer** is a regular fully connected layer. Each of its output neurons is connected to all the neurons from the input. This is usually implemented at the output together with a Softmax function to give the predictions. The nodes at the output of the layer, will, thus, contain the probabilities of the input to the CNN belonging to all classes.

### 2.7.3 Train a Neural Network

In this section some details of training a NN are discussed[1]. Some important definitions used in the context:

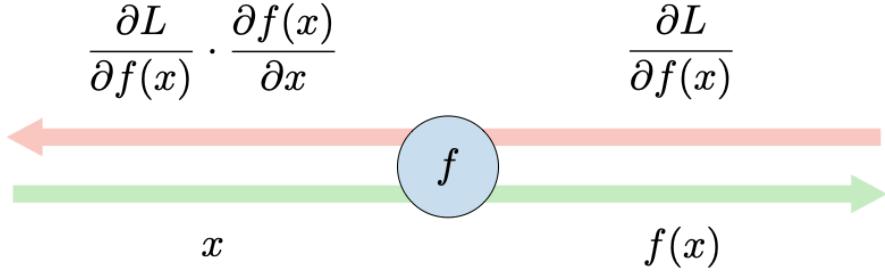


Figure 4: Illustration of Back Propagation

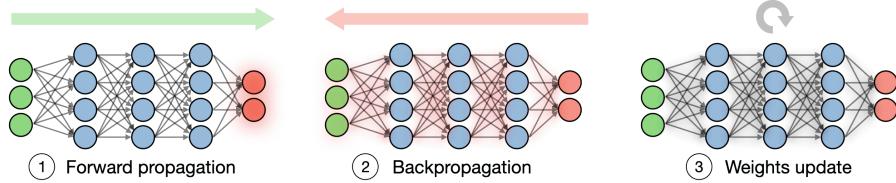


Figure 5: Illustration of weight update

1. **Epoch:** Epoch refers to one iteration where the model sees the whole training set to update its weights.
2. **Mini-batch gradient descent:** During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyper parameter that we can tune.
3. **Loss function:** In order to quantify how a given model performs, the loss function  $L$  is usually used to evaluate to what extent the actual outputs  $y$  are correctly predicted by the model outputs  $z$ .
4. **Cross-entropy loss:** In the context of binary classification in neural networks, the cross-entropy loss  $L(z,y) = -[y \log(z) + (1 - y) \log(1 - z)]$  is commonly used and is defined as follows:

$$L(z,y) = -[y \log(z) + (1 - y) \log(1 - z)] \quad (4)$$

5. **Back Propagation:** Back propagation is a method to update the weights in the neural network by optimizing for minimal errors using information with respect to each weight  $w$  is computed using the chain rule.

$$w \leftarrow w - \alpha \frac{\partial L(z,y)}{\partial w} \quad (5)$$

**Updating Weights:** In a neural network, weights are updated as follows:

1. Step 1: Take a batch of training data and perform forward propagation to compute the loss.
2. Step 2: Back propagate the loss to get the gradient of the loss with respect to each weight.
3. Step 3: Use the gradients to update the weights of the network.

## 2.8 Performance Metric

To measure the performance of the models, confusion matrix is used. The confusion:  $M_{i,j}$  is the number of sample with label  $i$  are being mis-classified as label  $j$ .

The matrix  $M$  exposes all the false positive and false negative value for each of the samples in the test set. The false positive quantity (portion of relevant elements among all elements has been classified into this category) is given by:

$$\text{FalsePositive} = \frac{M - \text{diag}(M)}{M \mathbb{J}} \quad (12)$$

Where, the division is an element-wise operation. For each row of the confusion matrix, we take the sum on the rows and divides the sum of the non-diagonal elements by the sum of that entire row of the matrix. The notation  $\mathbb{J}$  is the a vector consisting of all ones, having the same length as the height of the matrix  $M$ .

Similarly, the False Negative (The portion of elements that has been classified into this category and they are relevant) for each of the label is computed by:

$$\text{FalseNegative} = \frac{M^T - \text{diag}(M^T)}{M^T \mathbb{J}} \quad (13)$$

In addition, the overall accuracy of the model is computed via the sum of the diagonal elements on the confusion matrix over the sum of all the elements in the matrix.

The false-positive and false-negative rates are computed for each of the labels, and it serves as a sign of how much confusion is involved for each of the labels for a given model. However, this is only used for models with a large number of labels, and under the case where there is a limited number of labels, we made a choice to examine the confusion matrix manually.

## 3 Algorithm Implementation and Development

### 3.1 Data Standardization

---

**Algorithm 1:** Algorithm 1: Splitting by Classes and Data Standardization

---

```

1: Input: ClassSize, TestSet, Classes, Shuffle(Optional)
2: Images, Labels = EMNIST Test set if TestSet else EMNIST Training set
3: Initialize: SelectedIndices
4: for L in Classes do
5:   Indices = np.where(Labels==L)
6:   Reshape Indices into a 1-D numpy array
7:   if Shuffle then
8:     Shuffle Indices
9:   end if
10:  Indices = Indices[min(Indices.size, ClassSize)]
11:  SelectedIndices.append(Indices)
12: end for
13: RowDataMatrix := Images[Indices, ::, ::]
14: RowDataMatrix := RowDataMatrix as type np.floats
15: Labels := Labels[Indices]
16: RowDataMatrix := RowDataMatrix/255
17: RowDataMatrix := RowDataMatrix - mean(RowDataMatrix, axis=0)
18: Return: RowDataMatrix, Labels

```

---

The above algorithm [Algorithm 1](#) selects samples evenly distributed from the ENMIST training or test set. The variable “ClassSize” is the minimum amount of samples allowed for each of the given classes stored in the “Classes” variables. A shuffle option is an option that will perform shuffle action on all the indices selected for one of the given class labels.

The algorithm then normalize the data by putting each row of the matrix into zero mean and re-scale the data under the interval  $[0, 1]$ .

---

**Algorithm 2:** Algorithm 2: Subclass Classification

---

```
1: Input: AuxFunc, TestSet, ClassSize
2: Mapping, NewAxisLabels, NewLabels
3: Images, Labels := Images and labels from the EMNIST test set if TestSet else from EMNIST Training
   Set
4: TransformedLabels = np.vectorize(Mapping)(Labels)
5: Initialize: IndicesChosen
6: for II in NewLabels do
7:   Idx = np.where(TransformedLabels == II)
8:   Reshape Idx so it's 1-d Numpy array
9:   Shuffle Idx
10:  Idx := Idx[:min(Idx.size, classsize)]
11:  Merge Idx into: IndicesChosen
12: end for
13: Images := Images[IndicesChosen, ::, ::]
14: LabelsChosen := Labels[IndicesChosen]
15: DataMatrix := Reshape Images and standardize Images
16: Return: DataMatrix, Mapping(Labels), LabelsChosen, NewLabels, NewAxisLabels
```

---

**Algorithm 2** Another alternative algorithm is used to transform the dataset into sub-classes with different groupings. For example, we investigated the classification between some other ways of grouping: by Digits, Upper Cased Letters, and Lower Cased Letters. The algorithm takes in an auxiliary function that returns a function that maps the old labels to new labels (variable: “AuxFun”), a set of new labels (Variable “NewLabels”), along with new names for the labels (Variable “NewAxisLabels”).

The algorithm then groups together these classes with the same labels, shuffles them, and then transforms the old vector labels into the new set of labels. This is used for the training on labels with different type of grouping other than the default way of grouping provided by the EMNIST data set.

### 3.2 Extracting Statistics from the Confusion Matrix

---

**Algorithm 3:** Algorithm 3: Extracting Statistics from the Confusion Matrix

---

```
1: Input: PredictedLabels, ActualLabels
2: M := Get Confusion Matrix using sklearn using PredictedLabels, ActualLabels
3: OverAllAccuracy := sum(diag(M))/sum(M)
4: FalsePositive := sum(M - diag(M), axis=0)/sum(M, axis=0)
5: FalseNegative := sum(M-diag(M), axis=1)/sum(M, axis=1)
6: Sort the FalsePositive and FalseNegative in ascending order, with the labels
7: Plot the Confusion matrix M, and plot FalsePositive and FalseNegative in bar plot.
```

---

x

The algorithm is used to provides visualization for the performance metrics for all models. It's generic and allows for wide use on all predictive models trained in the project.

### 3.3 Training Models

The above algorithm [Algorithm 4](#) describes generic procedures for training a machine learning model using the first 2 algorithms. Dimensionality reduction is packaged into a class using python classes, and it's used before training the model. Embeddings for the samples on a lower dimension are created using the training set. The models are trained using the embeddings, and the test set is embedded into the lower dimension before feeding into the models for predictions.

---

**Algorithm 4:** Algorithm 4: Training Models

---

- 1: TrainX, TrainY = Using [Algorithm 1](#) or [Algorithm 2](#) to get the training, labels set.
  - 2: TestX, TestY = Using [Algorithm 1](#) or [Algorithm 2](#) to get the testing, labels set.
  - 3: DimReduce := Get a model by fitting TrainX, TrainY data using the LDA or PCA in sklearn
  - 4: EmbeddingsTrain := DimReduce.getEmbeddings(TrainX)
  - 5: EmbeddingsTest := DimReduce.getEmbeddings(TestX)
  - 6: Model := Initialize a Model using sklearn
  - 7: Model := Train on EmbeddingsTrain
  - 8: PredictedLabelsTrain = Model.predict(TrainEmbeddings)
  - 9: PredictedLabelsTest = Model.predict(TestEmbeddings)
- 

```

Trial 24 Complete {0h 0m 34s}
val_accuracy: 0.9245991494987488
Best val_accuracy So Far: 0.939423084259032
Total elapsed time: 0h 37m 38s
Search: Running Trial #25
Hyperparameter | Value | Best Value So Far
val_accuracy | 0.92 | 0.92
conv3_depth | 4 | 4
conv4_depth | 16 | 36
pooling | avg | avg
optimizer | sgd | sgd
learning_rate | 0.01 | 0.01
tuner/epochs | 20 | 7
tuner/initial_e... | 7 | 0
tuner/metric | f1 | f1
tuner/round | 1 | 1
tuner/trial_id | 1 | 0
tuner/trial_id | 4554d2dddfde391d... | None

Epoch 8/20
3120/3120 [=====] - ETA: 0.00s - loss: 1.0980 - accuracy: 0.6700 - val_loss: 0.2701 - val_accuracy: 0.9088
Epoch 9/20
3120/3120 [=====] - ETA: 0.00s - loss: 0.2700 - accuracy: 0.9095 - val_loss: 0.2070 - val_accuracy: 0.9281
Epoch 10/20
3120/3120 [=====] - ETA: 0.00s - loss: 0.2001 - accuracy: 0.9298 - val_loss: 0.2064 - val_accuracy: 0.9300
Epoch 11/20
3120/3120 [=====] - ETA: 0.00s - loss: 0.1626 - accuracy: 0.9415 - val_loss: 0.1982 - val_accuracy: 0.9332
Epoch 12/20
3120/3120 [=====] - ETA: 0.00s - loss: 0.1395 - accuracy: 0.9493 - val_loss: 0.1074 - val_accuracy: 0.9361
Epoch 13/20
3120/3120 [=====] - ETA: 0.00s - loss: 0.1186 - accuracy: 0.9552 - val_loss: 0.1054 - val_accuracy: 0.9379
Epoch 14/20
3120/3120 [=====] - ETA: 0.00s - loss: 0.1001 - accuracy: 0.9586 - val_loss: 0.2123 - val_accuracy: 0.9325
Epoch 15/20
3120/3120 [=====] - ETA: 0.00s - loss: 0.1020 - accuracy: 0.9612
741/3120 [=====] - ETA: 2:18 - loss: 0.1020 - accuracy: 0.9612

```

Figure 6: Optimization of Xception

### 3.4 Training CNN Models

In the project we want to classify the grouping of 10 digits and 26 letters. The training processes for both cases are similar. First we read a lot of tutorials and finally we decide to use the **Keras** with backend **Tensorflow**. Next, we determine the architecture of the CNN model. Finally we decide our lay are 2 CONV layers with Max-pooling and Dropout layers inserted. A flattening layer is then used, and after passing the fully connected layers, we get output with a softmax activation function. One thing to mention here is that we use the one-hot code, where one label is represented in vector format. After determining the architecture, the next step is to find the optimal parameters. This can be done with the help of the **Keras Tuner** [5]. We tested different combinations of a number of filers in CONV lay, kernel size, drop out rate and number of nodes in the fully connected layers. Our objective is the highest val\_accuracy. We also tried to train a big model, e.g. ResNet and Xception[4] by using the **Keras Tuner**. Unfortunately, we don't have enough powerful accelerators to boost the training. Alternatively we use Google Colab with GPU provided, however, this process is rather time consuming and Google Colab kill our training process before it finished. The search method is Hyperband[6], which is a novel approach to do hyper parameter optimization. After searching the best parameters in the parameter space, we retrain the model and save the best only with highest val\_accuracy. The validation split is 0.2. In this process, the early stopping callback is utilized to avoid overfitting issues. Finally we use the retrained best model to predict test data set and evaluate the performances of the CNN classifiers. Here I show one plot of hypertuning Xception with keras tuner. The tuning process was long.

## 4 Computational Results

### 4.1 LDA, PCA and Low Dimensional Projection

The projection onto the top 2 LDA sub-space components create the most amount of separations, and projecting the data into the PCA and then transforming it to the LDA still preserves most of the separations between the 2, z and o, 0, as shown in this figure. However, most of the data points with labels 2, z, and 0,

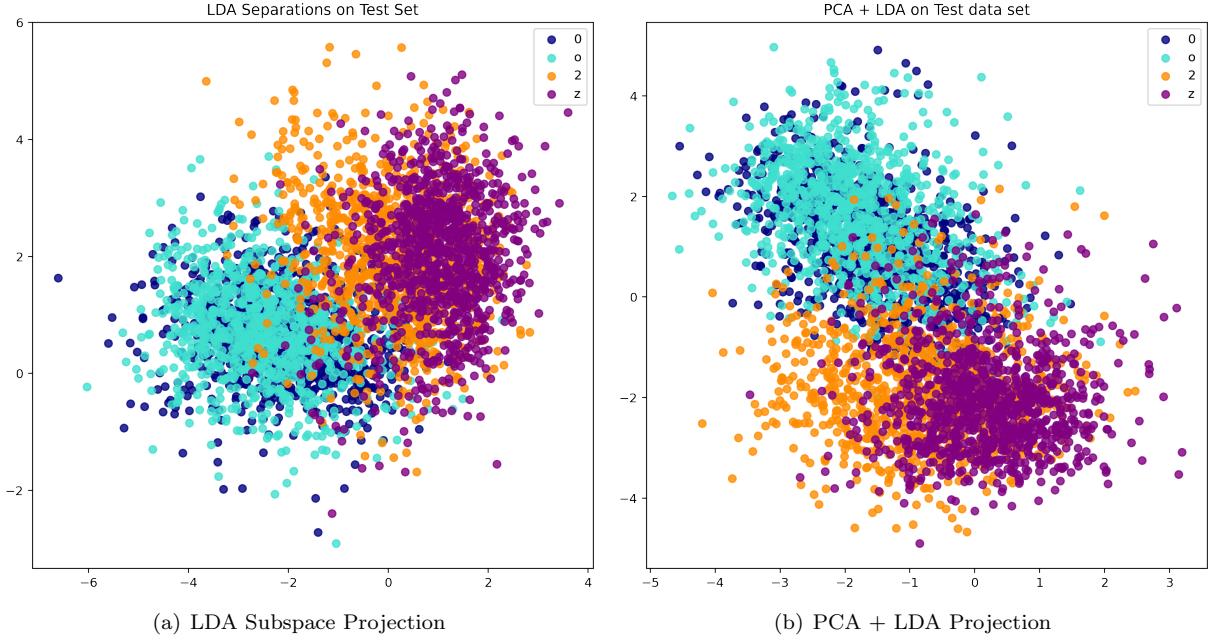


Figure 7: CNN Training curves

O overlaps with each other. Comparatively speaking the projection onto top 2 principal components show much fewer separations between the “Clouds” of data points between the label 2, z and 0,O are completely indistinguishable for each other after projecting on the top 2 principal components. This is shown in [this fig](#).

## 4.2 Hardest to Separate Digits

The confusion matrix for the SVM trained on the PCA + LDA embeddings are shown in [this fig](#). It’s trained on the equal-sized classes and on the labels that are easiest to confuse with and the results are for the test set. Comparatively speaking the random forest models performs with similar accuracy compare to the random forest which is shown in [this figure](#). The overall accuracy for the SVM (kbf kernel) across all labels are 55%. And the training set overall accuracy is 68% indicating an overfit for the SVM model. Random Forest on the other hand, scored a similar results with an overall accuracy of 58% on the test set and 73% on the training set. The minimal size of the leaf node has been tweaked manually to produce better results and prevent overfitting.

Note: Due to the fact that there are less than 1000 samples for the lower case and upper cased letter z, the results are biased and it needs to be re-weighted for to get the real performance for the model.

## 4.3 Super Class Classification

To our interest, we also tested the performance for the random forest on 3 of the super class of the labels (Letters lower and upper cased letters). A Random Forest is trained with embeddings from the PCA and the LDA dimensionality reduction, and the bars and the confusion matrix for random forest is shown in [this fig](#). However, in addition, we adjust the “class\_weight” parameter for the random forest to reduce the amount of false negative for the digit class. And the classes for Digits, lower and upper cased letters are weighted as “3, 1, 1”.

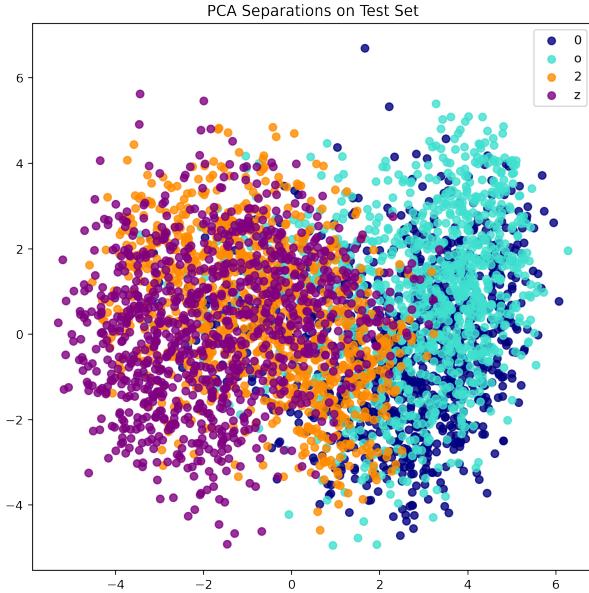


Figure 8: Projection of the PCA Embeddings

#### 4.4 Digits Separation on SVM and Random Forest

The SVM performs slightly worse by scoring a 93% compare to the CNN, which means that the CNN could learn more subtle differences between the digits 8 and 9. In addition, it has the same training errors and test errors for the SVM meaning that it's potentially under-fitting. Results are shown in [here](#). Random forest performs with a very similar score compare to SVM, for simplicity we will abridge the discussion for now.

#### 4.5 Convolutional Neural Network

We trained our own custom CNN mode with 2 CONV layers first, and then followed by a flatten layer and a fully connected layer. The training curves are shown [here](#). Thanks to the early stopping in **Keras**, and by setting patience 5 that before the model overfitting the training process is terminated. The model can be visualized in Figure 13.

However, in the letter classification case, the two curves are separating, which means it has the tendency to be overfitted. The reason can be that the model is too simple for the separate 26 types. We can try more deep and large networks.

With the help of the ModelCheckpoint callback defined by the keras, we save the best models with highest validation accuracy. And finally we get very good results on test dataset 99.63% and 98.88% for digits and letters. These results are impressive.

### 5 Summary and Conclusions

In this project, we explore the usage of SVD and PCA, as well as the CNN on EMNIST dataset. These classifiers perform well. The dimensionality reduction techniques were proven to be effective for speeding up the algorithm by several magnitudes, it also provides more intuitive understanding on the structure of the data set by visualizing the lower dimensional embeddings of the given samples.

Besides, we demonstrated the problem with irreducible errors using the traditional method. However, due to the complexity of training Neuro networks (for example the training of the Residual Neuro-nets took over a day), we were unable to test the hypothesis on all 62 labels, by comparing the convolutional Neuro Nets with classical methods. We also thought of experimenting with classification neuro nets with dimensionality reduction using PCA plus LDA but the results turned out to be unsatisfying.

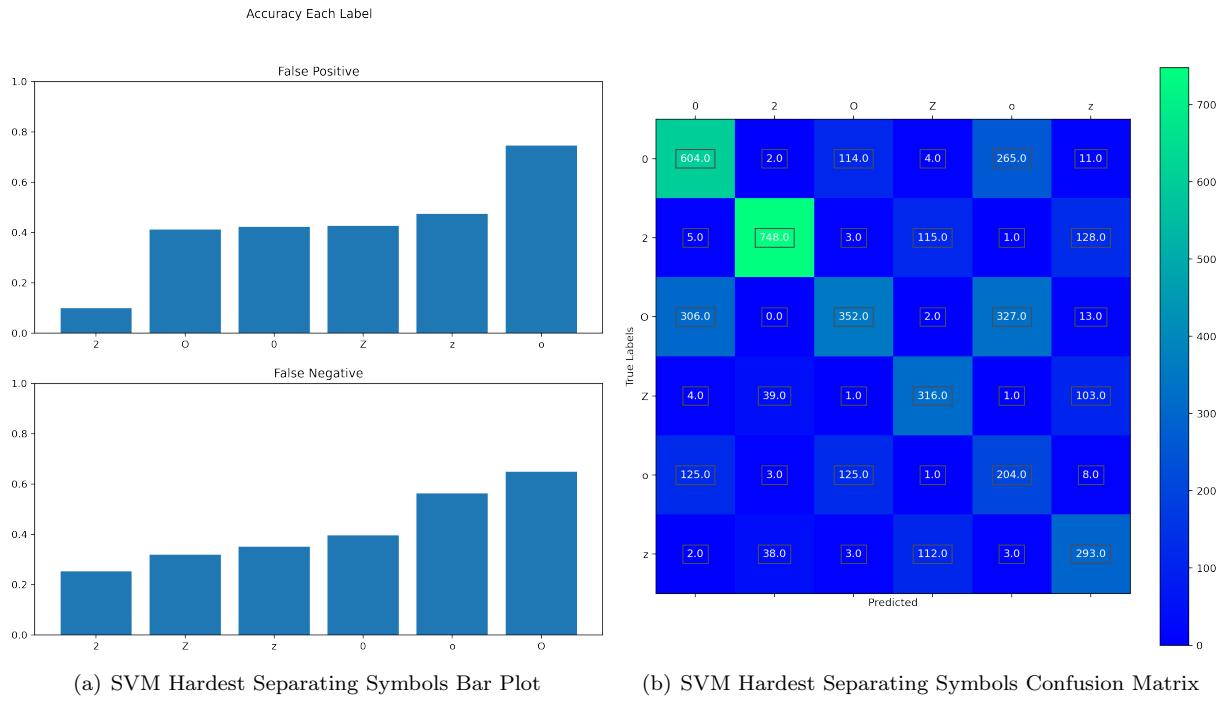


Figure 9: SVM Hardest Separations, Confusion Matrix (Right), FP, FN Bar plot (Left)

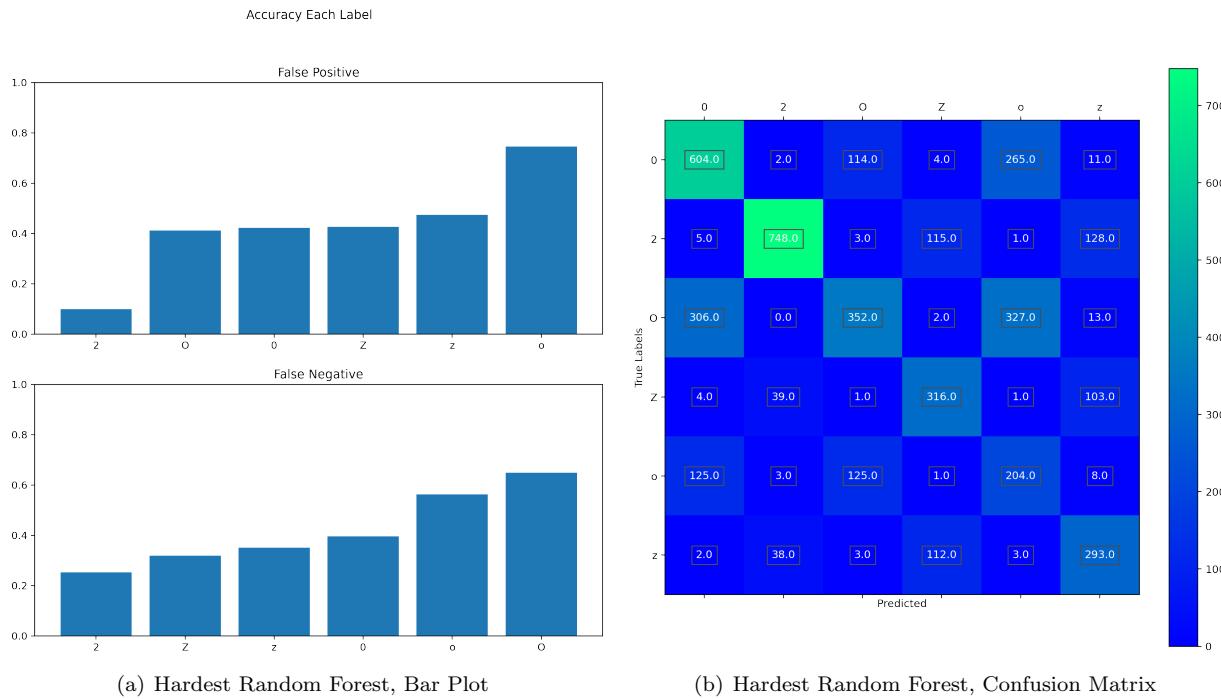


Figure 10: Hardest Separations using Random Forest

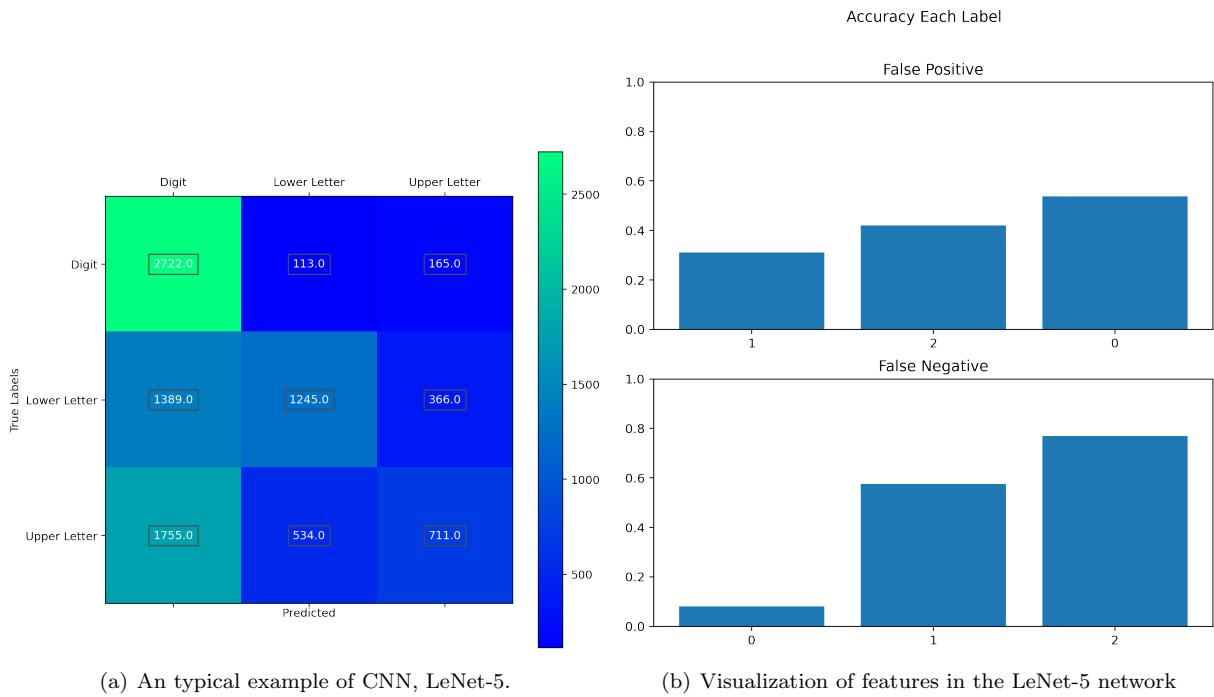


Figure 11: Super class classification: Random Forest with PCA + LDA

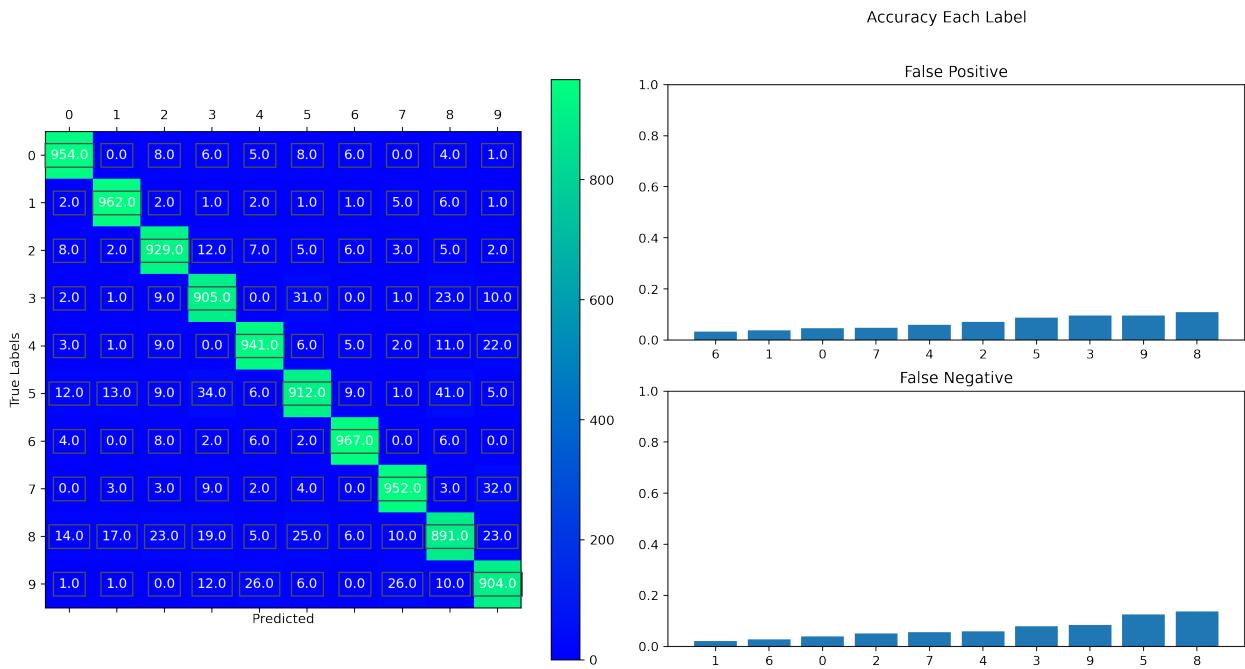


Figure 12: SVM: Separations on Digit

```

Model: "sequential"
-----  

Layer (type)      Output Shape       Param #
conv2d (Conv2D)   (None, 24, 24, 32)    832
-----  

max_pooling2d (MaxPooling2D) (None, 12, 12, 32)    0
dropout (Dropout) (None, 12, 12, 32)    0
conv2d_1 (Conv2D)  (None, 8, 8, 64)     51264
-----  

max_pooling2d_1 (MaxPooling2 (None, 4, 4, 64)    0
dropout_1 (Dropout) (None, 4, 4, 64)    0
flatten (Flatten) (None, 1024)        0
-----  

dense (Dense)     (None, 512)         524800
-----  

dropout_2 (Dropout) (None, 512)        0
-----  

dense_1 (Dense)   (None, 10)          5130
-----  

Total params: 582,026
Trainable params: 582,026
Non-trainable params: 0

```

Figure 13: CNN model

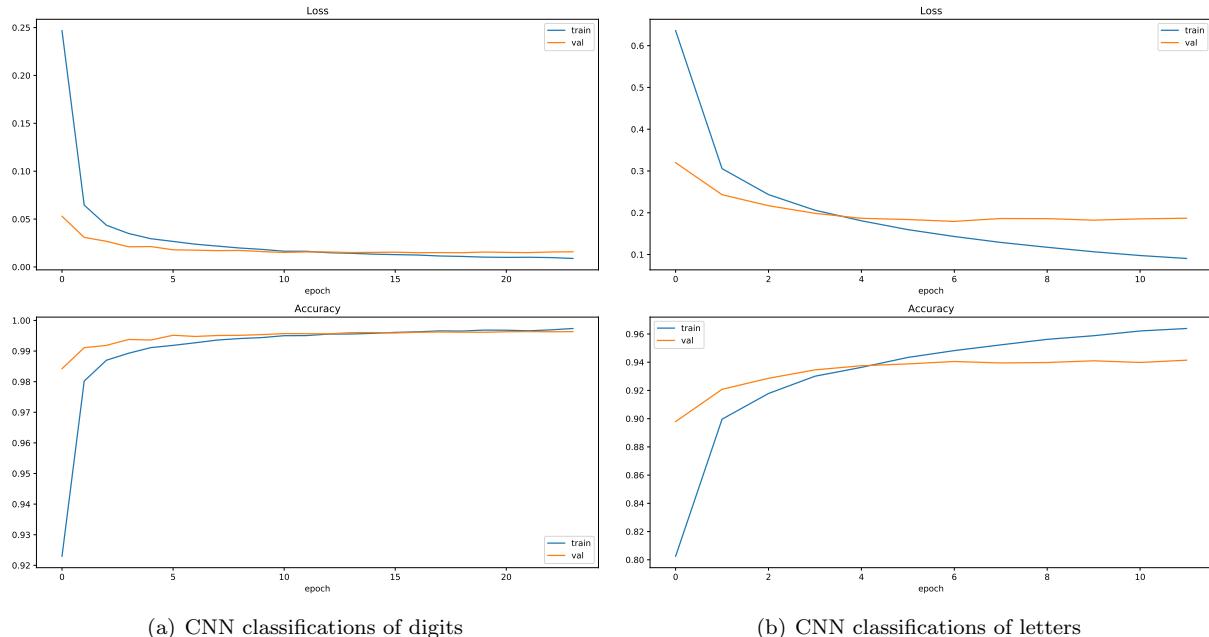
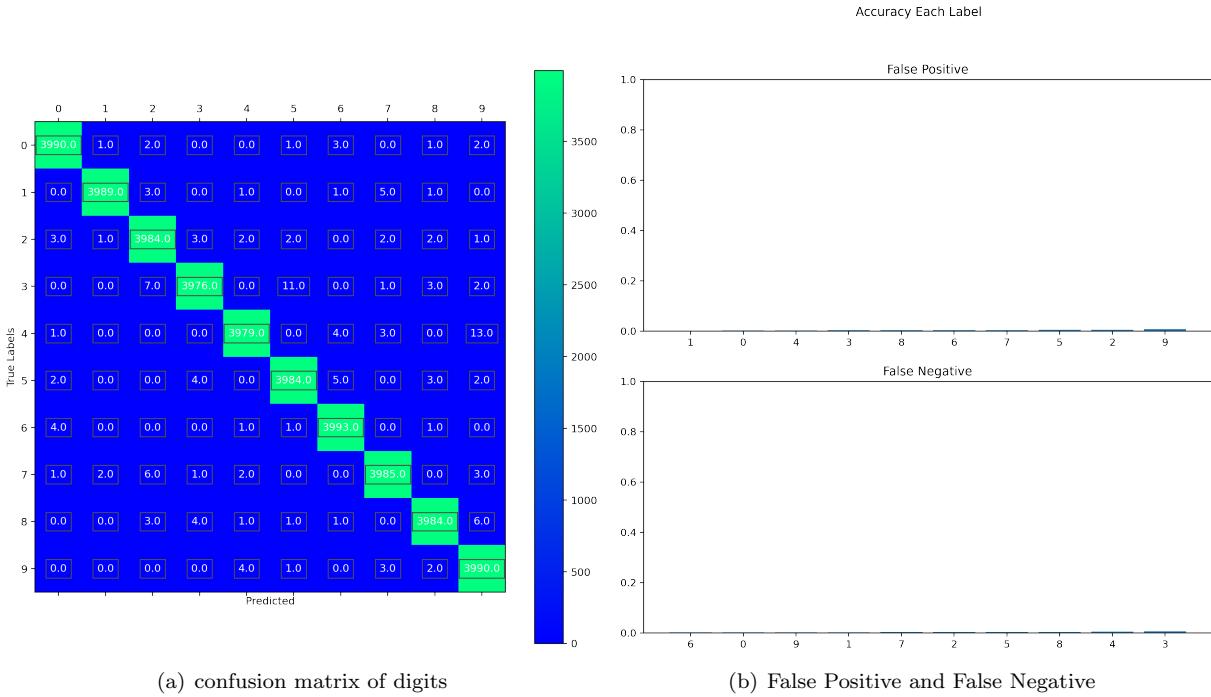


Figure 14: Lower Dimensional Embeddings using PCA, PCA + LDA



(a) confusion matrix of digits

(b) False Positive and False Negative

Figure 15: CNN for digits classification

However, there are still ideas that are left unexplored for this project. Specifically, adjusting the hyper-parameters for the models and using an unsupervised learning method to speed up the classifier. We hypothesized the potential of classifying the data first with unsupervised learning methods such as Gaussian Mixture Models or K-Means to cluster together data points that are close and then assign and train models on the results from the unsupervised classifier. This is done to speed up the training process because the models will be trained on a much smaller sub-set compared to training them directly on all the samples.

As for the CNN, there are still much work to do to improve the accuracy of letter classification. One can try a large and deeper model. Also, some advanced models can be used here, like the popular ResNet and Vgg. However, training these complex models needs lots of resources.

## References

- [1] Afshine Amidi and Shervine Amidi. *CS 230 - Deep Learning Tips and Tricks*. URL: <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks>.
- [2] Gregory Cohen et al. *EMNIST: An extension of MNIST to handwritten letters*. URL: <https://arxiv.org/pdf/1702.05373v1.pdf>.
- [3] Jiuxiang Gu et al. “Recent Advances in Convolutional Neural Networks”. In: *CoRR* abs/1512.07108 (2015). arXiv: [1512.07108](https://arxiv.org/abs/1512.07108). URL: <http://arxiv.org/abs/1512.07108>.
- [4] *HyperModel class*: URL: <https://keras-team.github.io/keras-tuner/documentation/hypermodels/>.
- [5] *Keras Tuner documentation*. URL: <https://keras-team.github.io/keras-tuner/>.
- [6] Lisha Li et al. “Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits”. In: *CoRR* abs/1603.06560 (2016). arXiv: [1603.06560](https://arxiv.org/abs/1603.06560). URL: <http://arxiv.org/abs/1603.06560>.
- [7] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html>.
- [8] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: <https://scikit-learn.org/stable/modules/svm.html#complexity>.
- [9] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: <https://scikit-learn.org/stable/modules/tree.html>.
- [10] Thalles Silva. *An illustrative introduction to Fisher’s Linear Discriminant*. URL: <https://towardsdatascience.com/an-illustrative-introduction-to-fishers-linear-discriminant-9484efee15ac>.
- [11] J. Nationah Kutz Steven L. Brunton. *Data-Driven Science and Engineering: Machine Learning, Dynamical System and Control*. Cambridge University Press, 2017, p. 7.
- [12] J. Nationah Kutz Steven L. Brunton. *Data-Driven Science and Engineering: Machine Learning, Dynamical System and Control*. Cambridge University Press, 2017, p. 35.
- [13] Sam T. *Entropy: How Decision Trees make Decisions*. URL: <https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8>.
- [14] Jerome Friedman Trevor Hastie Robert Tibshirani. *Elements of Statistical Learning, Data mining, Inference and Prediction*. Springer, 2017, p. 108.
- [15] Jerome Friedman Trevor Hastie Robert Tibshirani. *Elements of Statistical Learning, Data mining, Inference and Prediction*. Springer, 2017, p. 408.
- [16] Jerome Friedman Trevor Hastie Robert Tibshirani. *Elements of Statistical Learning, Data mining, Inference and Prediction*. Springer, 2017, p. 288.

## Appendix A Python Code

### A.1 Github url

Github repo: [link](#)

### A.2 main function

```
1 from emnist import extract_training_samples, extract_test_samples
2 import os
3 import numpy as np
4 import scipy.io as sio
5 import matplotlib.pyplot as plt
6 import matplotlib
```

```

7  from collections import Counter
8
9  from sklearn.decomposition import PCA
10 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
11 from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
12 from sklearn.tree import DecisionTreeClassifier
13 from sklearn.svm import SVC, LinearSVC
14 from sklearn.pipeline import make_pipeline
15 from sklearn.preprocessing import StandardScaler
16 from sklearn.metrics import confusion_matrix, plot_confusion_matrix
17
18 ## name space stuff
19 diag = np.diag
20 shuffle = np.random.shuffle
21 reshape = np.reshape
22 zeros = np.zeros
23 scatter = plt.scatter
24 show = plt.show
25 concatenate = np.concatenate
26 mean = np.mean
27 figure = plt.figure
28 title = plt.title
29 legend = plt.legend
30 matshow = plt.matshow
31 array_equal=np.array_equal
32 unique = np.unique
33 ylim = plt.ylim
34
35 ## Meta settings
36 CURRENT_DIRECTORY = None
37 UPPERLETTERS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
38 LOWERLETTERS = "abcdefghijklmnopqrstuvwxyz"
39 DIGITS = "0123456789"
40 matplotlib.rcParams['figure.figsize'] = (10, 10)
41 matplotlib.rcParams["figure.dpi"] = 220
42
43
44 def THREE_CLS_AUX_FUN():
45     """
46         Given this function , it returns a function that does the label mapping
47         and an array for the axis labels. (In the same order as the labels)
48
49         Maps the labels to these classes:
50             1. Is digit
51             2. Is smaller case letter .
52             3. Is cap letter .
53     """
54     def SplitByFunc(label:int):
55         if LabelsToSymbols(label) in DIGITS:
56             return 0
57
58         elif LabelsToSymbols(label) in LOWERLETTERS:
59             return 1
60

```

```

61         elif LabelsToSymbols(label) in UPPERLETTERS:
62             return 2
63         else:
64             raise Exception("Something is wrong this should not happen check
65             code. ")
66     AxisLabels = ["Digit", "Lower Letter", "Upper Letter"]
67     return SplitByFunc, AxisLabels, [0, 1, 2]
68
69 def TWO_CLS_AUXFUN():
70     """
71         Given this function , it returns a function that does the label mapping
72         and an array for the axis labels . (In the same order as the labels)
73
74         Maps the labels to these classes :
75         1. Is digit
76         2. Is not a digit .
77     """
78     def SplitByFunc(label:int):
79         if LabelsToSymbols(label) in DIGITS:
80             return 0
81         else:
82             return 1
83     AxisLabels = ["Digit", "Letters"]
84     return SplitByFunc, AxisLabels, [0, 1]
85
86
87 def SplitbyClasses(classSize=100, classes=None, shuffle_data=False, test_set=False):
88     """
89         Given the labels you want and the maximal size for each of the label ,
90             this function
91             will return the splitted data set for each .
92             Note:
93                 Data is chosen from the training set .
94             :param classes:
95                 An array for the labels you want to filter out from the EMNIST data
96                     set .
97             :param classSize:
98                 For each of the labels , what is the max size we want for each of the
99                     labels .
100
101
102         if test_set:
103             images, labels = extract_test_samples("byclass")
104         else:
105             images, labels = extract_training_samples("byclass")
106         IIdx = []
107         classes = list(range(0, 62)) if classes is None else classes
108         for Label in classes:
109             ClsIdx = np.argwhere(labels == Label)
110             ClsIdx = reshape(ClsIdx, ClsIdx.shape[0])
111             if shuffle_data: shuffle(ClsIdx)
112             ClsIdx = ClsIdx[: min(ClsIdx.shape[0], classSize)]
113             for Index in ClsIdx:

```

```

110     Idx.append(Idx)
111     RowDataMtx = images[Idx, :, :]
112     Labels = labels[Idx]
113     RowDataMtx = reshape(RowDataMtx, (len(Idx), 28*28))
114     RowDataMtx = RowDataMtx.astype(np.float)
115     RowDataMtx /= 255
116     RowDataMtx -= mean(RowDataMtx, axis=1, keepdims=True)
117     return RowDataMtx, Labels
118
119
120 def SplitbyLetterDigits(classSize=100, test_set=False):
121     """
122     Split the training/test data set into 3 big groups:
123     1. The digits.
124     2. The Upper cased letters.
125     3. The lower cased letters.
126
127     Note: This class will always shuffle the data!
128     Note: This class will perform zero mean unit variances STANDARDIZATION.
129
130     :param classSize:
131         The maximal size for each of the classes.
132     :param test_set:
133         Whether to choose the data set from the EMNIST test set, or the data
134         set.
135         The returned labels will still be in the range of [0, 62], but they
136         will be evenly partitioned into the 3 classes above.
137     """
138     if test_set:
139         images, labels = extract_test_samples("byclass")
140     else:
141         images, labels = extract_training_samples("byclass")
142
143     Organizer = LabelsOrganizer()
144     TreeClassesLabels = Organizer.getDataLabels(labels)
145     IndicesChosen = []
146     for II in range(3):
147         Idx = np.where(TreeClassesLabels==II)
148         Idx = Idx[0]
149         shuffle(Idx)
150         Idx = Idx[:min(len(Idx), classSize)]
151         for JJ in Idx:
152             IndicesChosen.append(JJJ)
153
154     RowDataMtx = images[IndicesChosen, ::, ::]
155     Labels = labels[IndicesChosen]
156     RowDataMtx = reshape(RowDataMtx, (len(IndicesChosen), 28 * 28))
157     RowDataMtx = RowDataMtx.astype(np.float)
158     RowDataMtx /= 255
159     RowDataMtx -= mean(RowDataMtx, axis=1, keepdims=True)
160     return RowDataMtx, Labels
161
162

```

```

163 def SplitByFunc(auxFun: callable , test_set=False , classSize:int=100):
164     """
165         Given a class function that transform the labels to the desired labels
166             , this
167             function will split the samples by the function .
168             1. Always shuffle the data .
169             2. The classSize is the maximal size for each class , if given class is
170                 less than it then
171                 all the samples of the class will be given and homogeneity will not
172                 be asserted .
173             3. images and labels will be normalized .
174
175     :param classSize:
176         The maixmal size for each of the classes .
177     :param test_set:
178         Whether to choose the labels and images from the ENMIST test set .
179     :param auxFun:
180         The auxilary function for sub-classes of all 62 labels .
181
182     """
183
184     auxFun , axisLabels , NewLabels = auxFun()
185     if test_set:
186         images , labels = extract_test_samples("byclass")
187     else:
188         images , labels = extract_training_samples("byclass")
189     TransformedLabels = np.vectorize(auxFun)(labels)
190     IndicesChosen = []
191     for II in NewLabels:
192         IIdx = np.where(TransformedLabels==II)
193         IIdx = IIdx[0]
194         shuffle(IIdx)
195         IIdx = IIdx[:min(len(IIdx) , classSize)]
196         for JJ in IIdx:
197             IndicesChosen.append(JJ)
198
199     RowDataMtx = images[IndicesChosen , :: , ::]
200     LabelsChosen = labels[IndicesChosen]
201     RowDataMtx = reshape(RowDataMtx, (len(IndicesChosen) , 28 * 28))
202     RowDataMtx = RowDataMtx.astype(np.float)
203     RowDataMtx /= 255
204     RowDataMtx -= mean(RowDataMtx, axis=1, keepdims=True)
205     return RowDataMtx , \
206             np.vectorize(auxFun)(LabelsChosen) , \
207             LabelsChosen , \
208             NewLabels , \
209             axisLabels
210
211
212 def SymbolsToLabels(symbol:str , SymToLabel=dict()):
213     assert len(symbol) == 1
214     if len(SymToLabel) != 0:
215         return SymToLabel[symbol]
216     Letters = "" .join([chr(97 + II) for II in range(26)])
217     Digits = "" .join(map(str , range(10)))

```

```

214     for II , V in enumerate(Digits + Letters.upper() + Letters):
215         SymToLabel[V] = II
216     return SymToLabel[symbol]
217
218
219 def LabelsToSymbols(label:int , LabelToSymb=dict()):
220     if len(LabelToSymb) != 0:
221         return LabelToSymb[label]
222     Letters = ''.join([chr(97 + II) for II in range(26)])
223     Digits = ''.join(map(str , range(10)))
224     for II , V in enumerate(Digits + Letters.upper() + Letters):
225         LabelToSymb[II] = V
226     return LabelToSymb[label]
227
228
229 class ConfusionMatrix:
230
231     def __init__(this , testLabels , predictedlabels , axisTicks=None):
232         this.TestLabels = testLabels
233         this.PredictedLabels = predictedlabels
234         Conmat = confusion_matrix(testLabels , predictedlabels)
235         this.ConfusionMatrix = Conmat
236         this.TotalAccuracy = \
237             np.sum(diag(Conmat))/np.sum(Conmat)
238         # False positive , none diag row sum, it is but actually it's not.
239         this.FalsePositiveEach = np.sum(Conmat - diag(diag(Conmat)) , axis=0)/(
240             np.sum(Conmat , axis=0))
241         # False negative , none diag column sum, it's not but actually it is.
242         this.FalseNegativeEach = np.sum(Conmat - diag(diag(Conmat)) , axis=1)/(
243             np.sum(Conmat , axis=1))
244         LabelsSorted = unique(this.TestLabels.copy())
245         np.sort(LabelsSorted)
246         if axisTicks is None:
247             Ticks = [LabelsToSymbols(II) for II in LabelsSorted]
248             this.AxisTicks = Ticks
249         else:
250             this.AxisTicks = axisTicks
251
252     def visualize(this , title:str=None):
253         """
254             Visualize the confusion matrix.
255         :return:
256             fig , ax
257         """
258         fig = figure()
259         ax = fig.add_subplot(111)
260         im = ax.matshow(this.ConfusionMatrix , cmap='winter' , interpolation='nearest')
261         fig.colorbar(im)
262
263         for (i , j) , z in np.ndenumerate(this.ConfusionMatrix):
264             ax.text(j , i , '{:0.1f}'.format(z) , ha='center' , va='center' , color =
265                     "white",
266                     bbox=dict(facecolor='none' , edgecolor='0.3'))

```

```

264
265     plt.xlabel("Predicted")
266     plt.ylabel("True Labels")
267     ax.set_xticks(np.arange(len(this.AxisTicks)))
268     ax.set_xticklabels(this.AxisTicks)
269     ax.set_yticks(np.arange(len(this.AxisTicks)))
270     ax.set_yticklabels(this.AxisTicks)
271     if title is not None:
272         fig.suptitle(title)
273     return fig, ax
274
275 def report(this, sort_it=True):
276
277     FPAXTicks = this.AxisTicks
278     FNAXTicks = this.AxisTicks
279     FP = this.FalsePositiveEach
280     FN = this.FalseNegativeEach
281
282     if sort_it:
283         ToSort = list(zip(FP, FPAXTicks))
284         ToSort = sorted(ToSort, key=lambda x: x[0])
285         FP, FPAXTicks = zip(*ToSort)
286
287         ToSort = list(zip(FN, FNAXTicks))
288         ToSort = sorted(ToSort, key=lambda x: x[0])
289         FN, FNAXTicks = zip(*ToSort)
290
291     print(f"Over all accuracy is: {this.TotalAccuracy}")
292     fig, (Top, Bottom) = plt.subplots(2, 1)
293     fig.suptitle("Accuracy Each Label")
294     Top.bar(FPAXTicks, FP)
295     Top.set_title("False Positive")
296     Top.set_ylim((0, 1))
297     Bottom.bar(FNAXTicks, FN)
298     Bottom.set_title("False Negative")
299     Bottom.set_ylim((0, 1))
300     return fig, (Top, Bottom)
301
302
303 class LabelsOrganizer:
304     """
305     Group all the labels by:
306         1. upper cased letters,
307         2. lower cased letters,
308         3. digits.
309     """
310
311     def __init__(this, mapping: callable=None, axisTicks: dict=None):
312         def DefaultMapping(label):
313             if LabelsToSymbols(label) in DIGITS:
314                 return 0
315             elif LabelsToSymbols(label) in LOWERLETTERS:
316                 return 1
317             elif LabelsToSymbols(label) in UPPERLETTERS:

```

```

318         return 2
319     else:
320         raise Exception("Something is wrong this should not happen
321                         check code. ")
322
323     if (mapping is None) != (axisTicks is None):
324         raise Exception("The new map for labels and the label mapping
325                         function has to be given at the same time.")
326
327     if mapping is None:
328         this.Mapping = DefaultMapping
329     else:
330         this.Mapping = mapping
331
332     if axisTicks is None:
333         this.AxisTicks = {0: "Digits", 1: "Lower Cased Letter", 2: "Upper
334                         Letters"}
335     else:
336         this.AxisTicks = axisTicks
337
338     def getDataLabels(this, labels):
339         Labels = np.vectorize(this.Mapping)(labels)
340         return Labels
341
342     def getDataTicks(this):
343         """
344             Returns: the axis labels for the confusion matrix.
345         """
346
347     return list(this.AxisTicks.values())
348
349 class LDADimReduce:
350
351     def __init__(this,
352                  X = None,
353                  y= None,
354                  n_components=None,
355                  classSize=1000,
356                  classes=None,
357                  shuffle=False
358                  ):
359
360         """
361             Creates an instance of the LDA dim_reduce on the EMNIST data set.
362             :param X: (Optional) The data that we want to train the LDA on.
363             :param y: (Optional, require X) The labels of the training data X.
364             :param n_components:
365                 The number of components we want to for the embeddings.
366             :param classSize:
367                 The size of the class to sample. All classes will be sampled by
368                     the same amount.
369             :param classes:
370                 List of labels we want to sample from the EMNIST data set.
371         """

```

```

368     if (X is None) or (y is None):
369         X, y = SplitbyClasses(classSize=classSize, classes=classes,
370                               shuffle_data=shuffle)
371         Template = LDA(n_components=n_components)
372         lda = Template.fit(X, y)
373         this.LdaModel = lda
374         this.Dim = n_components
375         this.ClassSize = classSize
376         this.Data = X
377         this.Labels = y
378
379     def getEmbeddings(this, toTransform=None):
380         """
381             Get the embeddings for the given data
382             :param toTransform: (Optional) if not given, it will return the
383                 embeddings of the data that the
384                 LDA model is trained on.
385             :return:
386                 The embeddings of the data. Each row is a sample.
387             """
388         lda = this.LdaModel
389         if toTransform is not None:
390             Embeddings = lda.transform(toTransform) # Rows are embeddings in
391                                         # all 60 dimensions
392             return Embeddings
393         else:
394             return lda.transform(this.Data)
395
396     class PCADimReduce:
397
398         def __init__(this,
399                     X=None,
400                     y=None,
401                     n_components=0.9,
402                     classSize=1000,
403                     classes=None,
404                     shuffle=False
405                     ):
406             """
407                 Create an instance for the PCA dimn reduce embedding.
408             :param X:
409             :param y:
410             :param n_components:
411             :param classSize:
412             :param classes:
413             """
414             if (X is None) or (y is None):
415                 X, y = SplitbyClasses(classSize=classSize, classes=classes,
416                                       shuffle_data=shuffle)
417             this.PcaModel = PCA(n_components=n_components, svd_solver="full")
418             this.PcaModel.fit(X, y)
419             this.classSize = 1000
420             this.classes = classes

```

```

418     this.Data = X
419     this.Labels = y
420
421     def getEmbeddings(this, X=None):
422         if X is None:
423             return this.PcaModel.transform(this.Data)
424         else:
425             return this.PcaModel.transform(X)
426
427     def getCompEV(this):
428         return this.PcaModel.explained_variance_ratio_
429
430     class DimReduceHybrid:
431         """
432             PCA + LDA embeddings.
433         """
434         def __init__(this, X=None, y=None, classes=None, classSize=1000,
435                     pca_components=0.9, shuffle=False):
436             if (X is None) or (y is None):
437                 X, y = SplitbyClasses(classSize=classSize, classes=classes,
438                                       shuffle_data=False)
439
440             this.PCAModel = PCA(n_components=pca_components)
441             this.PCAModel.fit(X, y)
442             PCAEmbeddings = this.PCAModel.transform(X)
443             this.LDAModel = LDA()
444             this.LDAModel.fit(PCAEmbeddings, y)
445
446             this.classSize = 1000
447             this.pca_components=0.9
448             this.Data = X
449             this.Labels = y
450
451         def getEmbeddings(this, X=None):
452             X = this.Data if X is None else X
453             PCAEmbeddings = this.PCAModel.transform(X)
454             LDAEEmbeddings = this.LDAModel.transform(PCAEmbeddings)
455             return LDAEEmbeddings
456
457     def main():
458         pass
459
460     if __name__ == "__main__":
461         if CURRENT_DIRECTORY is not None:
462             os.chdir(CURRENT_DIRECTORY)
463             print(f"Script CWD: {os.getcwd()}")
464             main()

```

### A.3 Plotting for random forest and SVM

```

1 #!/usr/bin/env python
2 # coding: utf-8

```

```

3
4 # In [1]:
5
6
7 from utils.core import *
8
9
10 # ## Content:
11 # 1. Dimension Reduction
12 # 2. SVM on various sub-groups of labels
13 # 3. Decision Tree on various sub-groups of labels
14
15 # ### Dimensionality Reduction Results:
16 #
17 # ##### Pure LDA separation
18
19 # In [2]:
20
21
22 matplotlib.rcParams['figure.figsize'] = (8, 8)
23 Symbols = "0o2z"
24 classes = [SymbolsToLabels(Sym) for Sym in Symbols]
25 LdaInstance = LDADimReduce()
26 Data, Labels = SplitbyClasses(classSize=1000, classes=classes)
27 # new data that never seemed before.
28 Embeddings = LdaInstance.getEmbeddings(Data);
29 colors = ['navy', 'turquoise', 'darkorange', "purple"]
30 SeparatingModes = [0, 5]
31 for color, II in zip(colors, classes):
32     scatter(
33         Embeddings[Labels == II, SeparatingModes[0]],
34         Embeddings[Labels == II, SeparatingModes[1]],
35         alpha=.8,
36         color=color
37     )
38 legend(list(Symbols))
39 title("LDA Separations on Test Set")
40 show()
41 print("Demonstration ended. ")
42
43
44 # ##### Pure PCA Separation
45 #
46
47 # In [3]:
48
49
50 Symbols = "0o2z"
51 classes = [SymbolsToLabels(Sym) for Sym in Symbols]
52 LdaInstance = PCADimReduce()
53 Data, Labels = SplitbyClasses(classSize=1000, classes=classes)
54 # new data that never seemed before.
55 Embeddings = LdaInstance.getEmbeddings(Data);
56 colors = ['navy', 'turquoise', 'darkorange', "purple"]

```

```

57 SeparatingModes = [0, 5]
58 for color, II in zip(colors, classes):
59     scatter(
60         Embeddings[Labels == II, SeparatingModes[0]],
61         Embeddings[Labels == II, SeparatingModes[1]],
62         alpha=.8,
63         color=color
64     )
65 legend(list(Symbols))
66 title("PCA Separations on Test Set")
67 show()
68 print("Demonstration ended. ")
69
70
71 # ##### PCA + LDA Separations
72
73 # In [4]:
74
75
76 PCAInstance = PCADimReduce(n_components=0.8); print("Getting PCA Model... ")
77 PCAEmbeddings = PCAInstance.getEmbeddings(); print("Getting PCA Embeddings... ")
78
79 TrainLabels = PCAInstance.Labels
80 print("Train LDA on PCA Embeddings... ")
81 LDAMaxComponents = min(61, PCAInstance.PcaModel.n_components - 1)
82 LDAInstance = LDADimReduce(n_components=LDAMaxComponents, # Just to be sure.
83                             X=PCAEmbeddings,
84                             y=TrainLabels) # Use the PCA embeddings to train
85                             LDA
86
87 classes = [SymbolsToLabels(Char) for Char in "0o2z"]
88 TestData, TestLabels = SplitbyClasses(classSize=1000, classes=classes)
89 # Get the LDA embeddings of the PCA embeddings.
90 print("Represent using PCA modes and then on LDA basis... ")
91 Embeddings = LDAInstance.getEmbeddings(PCAInstance.getEmbeddings(TestData))
92 colors = ['navy', 'turquoise', 'darkorange', "purple"]
93
94 ## Plotting the Results
95 SeparatingModes = [0, 1]
96 for color, II in zip(colors, classes):
97     scatter(
98         Embeddings[TestLabels == II, SeparatingModes[0]],
99         Embeddings[TestLabels == II, SeparatingModes[1]],
100        alpha=.8,
101        color=color
102    )
103 legend(list("0o2z"))
104 title("PCA + LDA on Test data set")
105 show()
106
107 # ##### PCA + LDA Embeddings Version 2
108 #

```

```

109
110 # ### SVM
111 # * SVM trained on big subsets.
112 #     * LDA Embeddings for 26 lower cased letters
113 #     * LDA Embeddings for 26 lower cased letters and digits
114 #     * PCA + LDA Embeddings for 26 lower cased letters and digits
115 # * SVM trained on all equally sampled 62 symbols
116 #     * With LDA embeddings
117 # * SVM trained on the hardest to 4 symbols
118
119 # ##### SVM on 26 lower cased Letters , with LDA Embeddings
120 #
121 # Setup the SVM, plitting the training set and the test sets and get their LDA
122 #     embeddings.
123 # In [2]:
124
125
126 DisplayLabels = "abcdefghijklmnopqrstuvwxyz"
127 classes = [SymbolsToLabels(II) for II in DisplayLabels]
128 Model = make_pipeline(StandardScaler(), SVC(gamma="auto")) # Making the SVC
129 #     Model.
130 TrainX, TrainY = SplitbyClasses(classes=classes, classSize=1000)
131 TestX, TestY = SplitbyClasses(classes=classes, classSize=500, shuffle_data=
132     True, test_set=True)
133 DimRe = LDADimReduce(X=TrainX, y=TrainY) # Use Train set to create LDA
134 #     embeddings.
135 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
136 #     LDA
137 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the test
138 #     set.
139
140 # In [3]:
141
142
143 matplotlib.rcParams['figure.figsize'] = (20, 20)
144 PredictedLabels = Model.predict(TestEmbeddings)
145 Conmat = ConfusionMatrix(PredictedLabels, TestY)
146 Conmat.visualize()
147 Conmat.report()
148
149
150 # In [7]:
151
152
153 PredictedLabels = Model.predict(TrainEmbeddings)
154 Conmat = ConfusionMatrix(PredictedLabels, TrainY)
155 Conmat.visualize()

```

```

157 Conmat.report()
158
159
160 # ##### SVM with LDA on [a-z] + [0, 9]
161
162 # In [13]:
163
164
165 DisplayLabels = "" .join(map(str, range(10))) + "abcdefghijklmnopqrstuvwxyz".upper()
166 classes = [SymbolsToLabels(II) for II in DisplayLabels]
167 Model = make_pipeline(StandardScaler(), SVC(gamma="auto")) # Making the SVC
Model.
168
169 TrainX, TrainY = SplitbyClasses(classes=classes, classSize=1000)
170 TestX, TestY = SplitbyClasses(classes=classes, classSize=3000, shuffle_data=True,
test_set=True)
171
172 DimRe = LDADimReduce(X=TrainX, y=TrainY) # Use Train set to create LDA
embeddings.
173 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
LDA
174 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the test
set.
175
176 Model.fit(TrainEmbeddings, TrainY); print(f"Modeling Has been fitted: ")
177 print(Model)
178
179
180 # In [14]:
181
182
183 matplotlib.rcParams['figure.figsize'] = (30, 30)
184 PredictedLabels = Model.predict(TestEmbeddings)
185 Conmat = ConfusionMatrix(PredictedLabels, TestY)
186 Conmat.visualize()
187 Conmat.report()
188
189
190 # In [15]:
191
192
193 matplotlib.rcParams['figure.figsize'] = (30, 30)
194 PredictedLabels = Model.predict(TrainEmbeddings)
195 Conmat = ConfusionMatrix(PredictedLabels, TrainY)
196 Conmat.visualize()
197 Conmat.report()
198
199
200 # ##### SVM with PCA + LDA Embeddings on [a-z] + [0, 9]
201
202 # In [11]:
203
204

```

```

205 DisplayLabels = "" .join(map(str , range(10))) + "abcdefghijklmnopqrstuvwxyz"
206 classes = [SymbolsToLabels(II) for II in DisplayLabels]
207 Model = make_pipeline(StandardScaler() , SVC(gamma="auto")) # Making the SVC
208 Model.
209
210 TrainX , TrainY = SplitbyClasses(classes=classes , classSize=1000)
211 TestX , TestY = SplitbyClasses(classes=classes , classSize=1000, shuffle_data=
212 True , test_set=True)
213 DimRe = DimReduceHybrid(X=TrainX , y=TrainY) # Use Train set to create LDA
214 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
215 LDA
216 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the test
217 set.
218
219
220 # In [12]:
221
222
223 matplotlib.rcParams['figure.figsize'] = (30, 30)
224 PredictedLabels = Model.predict(TestEmbeddings)
225 Conmat = ConfusionMatrix(PredictedLabels , TestY)
226 Conmat.visualize()
227 Conmat.report()
228
229
230 # In [13]:
231
232
233 matplotlib.rcParams['figure.figsize'] = (30, 30)
234 PredictedLabels = Model.predict(TrainEmbeddings)
235 Conmat = ConfusionMatrix(PredictedLabels , TrainY)
236 Conmat.visualize()
237 Conmat.report()
238
239
240 # ##### Linear SVM with LDA for [0-9] + [a-z] + [A-Z]
241 #
242 # Training the Model
243
244 # In [12]:
245
246
247 DisplayLabels = DisplayLabels = "" .join(map(str , range(10))) + "
248 abcdefghijklmnopqrstuvwxyz" + "abcdefghijklmnopqrstuvwxyz".upper()
249 classes = [SymbolsToLabels(II) for II in DisplayLabels]
250 Model = make_pipeline(StandardScaler() , LinearSVC()) # Making the SVC Model.
251 TrainX , TrainY = SplitbyClasses(classes=classes , classSize=2000)
252 TestX , TestY = SplitbyClasses(classes=classes , classSize=1000, shuffle_data=

```

```

    True, test_set=True)

253 DimRe = LDADimReduce(X=TrainX, y=TrainY) # Use Train set to create LDA
254     embeddings.
255 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
256     LDA
257 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the test
258     set.

259 Model.fit(TrainEmbeddings, TrainY); print(f"Modeling Has been fitted: ")
260 print(Model)

261
262 # ##### Confusion Matrix on the Test set
263
264 # In [5]:
265
266
267 # Graphics settings!
268 matplotlib.rcParams['figure.figsize'] = (40, 40)
269 PredictedLabels = Model.predict(TestEmbeddings)
270 Conmat = ConfusionMatrix(PredictedLabels, TestY)
271 Conmat.visualize()
272 Conmat.report()

273
274
275 # In [6]:
276
277
278 PredictedLabels = Model.predict(TrainEmbeddings)
279 Conmat = ConfusionMatrix(PredictedLabels, TrainY)
280 Conmat.visualize()
281 Conmat.report()

282
283
284 # ##### SVM on Hardest Symbols to Separate
285
286 # In [7]:
287
288
289 DisplayLabels = DisplayLabels = "0oOz25s$"
290 classes = [SymbolsToLabels(II) for II in DisplayLabels]
291 Model = make_pipeline(StandardScaler(), SVC(gamma="auto")) # Making the SVC
292     Model.

293 TrainX, TrainY = SplitbyClasses(classes=classes, classSize=1000)
294 TestX, TestY = SplitbyClasses(classes=classes, classSize=3000, shuffle_data=
295     True, test_set=True)

296 DimRe = LDADimReduce(X=TrainX, y=TrainY) # Use Train set to create LDA
297     embeddings.
298 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
299     LDA
300 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the test

```

```

    set.

299 Model.fit(TrainEmbeddings, TrainY); print(f"Modeling Has been fitted: ")
300 print(Model)
301
302
303
304 # In [10]:
305
306
307 matplotlib.rcParams['figure.figsize'] = (8, 8)
308 PredictedLabels = Model.predict(TestEmbeddings)
309 Conmat = ConfusionMatrix(TestY, PredictedLabels)
310 _, ax = Conmat.visualize()
311 ax.set_title("SVM Confusion Test set")
312 _, ax = Conmat.report()
313 Model.score(TestEmbeddings, TestY)
314
315
316 # In [11]:
317
318
319 matplotlib.rcParams['figure.figsize'] = (8, 8)
320 PredictedLabels = Model.predict(TrainEmbeddings)
321 Conmat = ConfusionMatrix(TrainY, PredictedLabels)
322 Conmat.visualize()
323 Conmat.report()
324
325
326 # ### SVM Only on Digits with PCA + LDA
327
328 # In [10]:
329
330
331 DisplayLabels = "0123456789"
332 classes = [SymbolsToLabels(II) for II in DisplayLabels]
333 Model = make_pipeline(StandardScaler(), SVC(gamma="auto", kernel='poly',
       degree=3)) # Making the SVC Model.
334
335 TrainX, TrainY = SplitbyClasses(classes=classes, classSize=10000)
336 TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=
       True, test_set=True)
337
338 DimRe = DimReduceHybrid(X=TrainX, y=TrainY, pca_components=0.7) # Use Train
       set to create LDA embeddings.
339 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
       LDA
340 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the test
       set.
341
342 Model.fit(TrainEmbeddings, TrainY); print(f"Modeling Has been fitted: ")
343 print(Model)
344
345
346 # In [7]:

```

```

347
348
349 # Graphics settings!
350 matplotlib.rcParams['figure.figsize'] = (8, 8)
351 PredictedLabels = Model.predict(TestEmbeddings)
352 Conmat = ConfusionMatrix(PredictedLabels, TestY)
353 Conmat.visualize()
354 Conmat.report()
355
356
357 # In [8]:
358
359
360 matplotlib.rcParams['figure.figsize'] = (8, 8)
361 PredictedLabels = Model.predict(TrainEmbeddings)
362 Conmat = ConfusionMatrix(PredictedLabels, TrainY)
363 Conmat.visualize()
364 Conmat.report()
365
366
367 # ### Decision Tree
368 #
369 # Due to some of the inherent draw backs of a single decision tree, here we
370 # will explore the performance of Ensemble Tree under a optimal dimension
371 # for embeddings.
372 #
373 # #### Hardest Separations on Random Forest:
374 # With PCA + LDA Dimensionality Reduction.
375
376
377 Symbols = "0o2zZO"
378 classes = [SymbolsToLabels(II) for II in Symbols]
379 TrainX, TrainY = SplitbyClasses(classes=classes, classSize=2000)
380 TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=True, test_set=True)
381
382 DimRe = DimReduceHybrid(X=TrainX, y=TrainY, pca_components=0.9) # Use Train
383 # set to create LDA embeddings.
384 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
385 # LDA
386 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the
387 # test set.
388 Model = RandomForestClassifier(n_estimators=200, n_jobs=-1, min_samples_leaf
389 =10)
390 Model.fit(TrainEmbeddings, TrainY)
391
392
393 # In [4]:
394
395
396 PredictedTestLabels = Model.predict(TestEmbeddings)
397 Conmat = ConfusionMatrix(TestY, PredictedTestLabels)

```

```

394 Conmat.visualize()
395 Conmat.report()
396
397
398 # In [5]:
399
400
401 PredictedTestLabels = Model.predict(TrainEmbeddings)
402 Conmat = ConfusionMatrix(TrainY, PredictedTestLabels)
403 Conmat.visualize()
404 Conmat.report()
405
406
407 ##### Random Forest Separating all labels with PCA + LDA
408 # * This is hard.
409
410 # In [3]:
411
412
413 Symbols = "".join(map(str, range(10))) + "abcdefghijklmnopqrstuvwxyz" + "abcdefghijklmnopqrstuvwxyz".upper()
414 classes = [SymbolsToLabels(II) for II in Symbols]
415 TrainX, TrainY = SplitbyClasses(classes=classes, classSize=2000)
416 TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=True, test_set=True)
417
418 DimRe = DimReduceHybrid(X=TrainX, y=TrainY, pca_components=0.9) # Use Train
        set to create LDA embeddings.
419 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
        LDA
420 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the
        test set.
421 Model = RandomForestClassifier(n_estimators=200, n_jobs=-1, min_samples_leaf
        =1/(1000*62))
422 Model.fit(TrainEmbeddings, TrainY)
423
424
425 # In [4]:
426
427
428 matplotlib.rcParams['figure.figsize'] = (60, 60)
429 PredictedTestLabels = Model.predict(TestEmbeddings)
430 Conmat = ConfusionMatrix(TestY, PredictedTestLabels)
431 Conmat.visualize()
432 matplotlib.rcParams['figure.figsize'] = (30, 30)
433 Conmat.report()
434
435
436 # In [5]:
437
438
439 matplotlib.rcParams['figure.figsize'] = (60, 60)
440 PredictedTestLabels = Model.predict(TrainEmbeddings)
441 Conmat = ConfusionMatrix(TrainY, PredictedTestLabels)

```

```

442 Conmat.visualize()
443 Conmat.report()
444
445
446 # ### Random Forest Separating Digits and Lower Cased Letters
447 #
448
449 # In [26]:
450
451
452 Symbols =     "" . join ( map ( str , range ( 10 ))) + "abcdefghijklmnopqrstuvwxyz"
453 classes = [SymbolsToLabels ( II ) for II in Symbols]
454 TrainX , TrainY = SplitbyClasses ( classes=classes , classSize=2000)
455 TestX , TestY = SplitbyClasses ( classes=classes , classSize=1000 , shuffle_data=
    True , test_set=True)
456
457 DimRe = DimReduceHybrid ( X=TrainX , y=TrainY , pca_components=0.9) # Use Train
    set to create LDA embeddings .
458 TrainEmbeddings = DimRe.getEmbeddings () # Get Embeddings from the set trained
    LDA
459 TestEmbeddings = DimRe.getEmbeddings (TestX) # Get the embeddings from the
    test set .
460
461 Model = RandomForestClassifier ( n_estimators=200 , n_jobs=-1 , min_samples_leaf
    =2)
462 Model . fit (TrainEmbeddings , TrainY)
463
464
465 # In [25]:
466
467
468 matplotlib.rcParams [ ' figure . figsize ' ] = (30 , 30)
469 PredictedTestLabels = Model . predict (TestEmbeddings)
470 Conmat = ConfusionMatrix (TestY , PredictedTestLabels)
471 Conmat . visualize ()
472 Conmat . report ()
473
474
475 # ### Separating by Large Sub classes With LDA + PCA using Rand Forest
476 #
477 # Over all separation is the worse among all , WITH WEAKED CLASS WEIGHT, We
    increase the probability of classifying as a digits , and it in deed , is a
    digit .
478
479 # In [4]:
480
481
482 TrainX , TrainY , _ , _ , AxTicks = SplitByFunc (THREE_CLS_AUX_FUN , classSize
    =10000)
483 TestX , TestY , _ , _ , _ = SplitByFunc (THREE_CLS_AUX_FUN , classSize=3000 , test_set
    =True)
484
485 DimRe = DimReduceHybrid ( X=TrainX , y=TrainY) # Use Train set to create LDA
    embeddings .

```

```

486 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
     LDA
487 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the
     test set.
488
489 # Model = RandomForestClassifier(n_estimators=100, n_jobs=-1, max_leaf_nodes=)
490 Model = RandomForestClassifier(n_estimators=200,
491                               n_jobs=-1,
492                               verbose=1,
493                               min_samples_leaf=0.01,
494                               class_weight={0:3, 1: 1, 2: 1})
495 Model.fit(TrainEmbeddings, TrainY)
496 TestLabelsPredicted = Model.predict(TestEmbeddings)
497 Conmat = ConfusionMatrix(TestY, TestLabelsPredicted, axisTicks=AxTicks)
498 matplotlib.rcParams['figure.figsize'] = (8, 8) # Graphics settings
499 Conmat.visualize()
500 show()
501 Conmat.report()
502 show()
503 Model.fit(TestEmbeddings, TestY)
504
505 TrainlabelsPredicted = Model.predict(TrainEmbeddings)
506 Conmat = ConfusionMatrix(TrainY, TrainlabelsPredicted)
507
508 Conmat.visualize()
509 show()
510 Conmat.report()
511 show()
512 Model.fit(TestEmbeddings, TestY)
513
514
515 # ### Binary Classification Between Digits and Data
516 #
517
518 # In [11]:
519
520
521 TrainX, TrainY, _, _, AxTicks = SplitByFunc(TWO_CLS_AUX_FUN, classSize=10000)
522 TestX, TestY, _, _, _ = SplitByFunc(TWO_CLS_AUX_FUN, classSize=3000, test_set=True)
523
524 DimRe = DimReduceHybrid(X=TrainX, y=TrainY) # Use Train set to create LDA
     embeddings.
525 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
     LDA
526 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the
     test set.
527
528 # Model = RandomForestClassifier(n_estimators=100, n_jobs=-1, max_leaf_nodes=)
529 Model = RandomForestClassifier(n_estimators=200,
530                               n_jobs=-1,
531                               verbose=1,
532                               min_samples_leaf=0.001,
533                               class_weight={0: 1, 1: 1})

```

```

534 Model.fit(TrainEmbeddings, TrainY)
535 TestLabelsPredicted = Model.predict(TestEmbeddings)
536 Conmat = ConfusionMatrix(TestY, TestLabelsPredicted, axisTicks=AxTicks)
537 matplotlib.rcParams['figure.figsize'] = (8, 8) # Graphics settings
538 Conmat.visualize()
539 show()
540 Conmat.report()
541 show()

542
543 TrainlabelsPredicted = Model.predict(TrainEmbeddings)
544 Conmat = ConfusionMatrix(TrainY, TrainlabelsPredicted, axisTicks=AxTicks)
545
546 Conmat.visualize()
547 show()
548 Conmat.report()
549 show()

550
551 #
552 # ###
553 # Random Forest on the Digits
554 #
555 #
556
557 # In [15]:
558
559
560 Symbols = ''.join(map(str, range(10)))
561 classes = [SymbolsToLabels(II) for II in Symbols]
562 TrainX, TrainY = SplitbyClasses(classes=classes, classSize=float("inf"))
563 TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=True, test_set=True)
564
565 DimRe = DimReduceHybrid(X=TrainX, y=TrainY, pca_components=0.9) # Use Train
      set to create LDA embeddings.
566 TrainEmbeddings = DimRe.getEmbeddings() # Get Embeddings from the set trained
      LDA
567 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the
      test set.

568
569 Model = RandomForestClassifier(n_estimators=200, n_jobs=-1)
570 Model.fit(TrainEmbeddings, TrainY)

571
572
573 # In [14]:
574
575
576 matplotlib.rcParams['figure.figsize'] = (30, 30)
577 PredictedTestLabels = Model.predict(TestEmbeddings)
578 Conmat = ConfusionMatrix(TestY, PredictedTestLabels)
579 Conmat.visualize()
580 Conmat.report()

581
582
583 # ###
584 # Reference Resources

```

```

584 #
585 # —
586 # ##### Reference Materials Used:
587 #
588 #
589 # SK: plotting the confusion chart
590 # https://scikit-learn.org/stable/auto_examples/model_selection/
      plot_confusion_matrix.html
591 #
592 #
593 # SK: LDA
594 # https://scikit-learn.org/0.16/modules/generated/sklearn.lda.html#sklearn
      .lda.LDA.transform
595 #
596 #
597 # SK: PCA
598 # https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.
      html
599 #
600 #
601 # SK: Understanding the Decision Tree Structure:
602 # https://scikit-learn.org/stable/auto_examples/tree/
      plot_unveil_tree_structure.html#sphx-glr-auto-examples-tree-plot-unveil-
      tree-structure-py
603 #
604 #
605 # SK: Decision Tree
606 # https://scikit-learn.org/stable/modules/tree.html#classification
607 #
608 # Highlighting the pro and cons of using decision tree
609 #
610 #
611 # SK: RandomForestClassifier
612 # https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
      RandomForestClassifier.html
613 #
614 #
615 # A better version of decision tree.
616 #
617 #
618 # Research Paper: Solvers For SVM
619 # https://leon.bottou.org/publications/pdf/lin-2006.pdf
620 #
621 #
622 # Sk: SVM Complexity
623 # https://scikit-learn.org/stable/modules/svm.html#complexity
624 # Hilighting the complexity for using the SVM classifier.
625 #
626 #
627 # Book: Elements of Statistical Learning:
628 # Highlighting the reduction process of the SVM problem into a quadratic
      programming problem, and solver that focuses on KKT conditions and stuff
      like that.
629 #

```

```

630 #
631 # Medium Article:
632 # https://towardsdatascience.com/entropy-how-decision-trees-make-decisions
633 # -2946b9c18c8
634 # An introduction to decision tree , Information gain , entropy etc.
635 #
636 #
637 # Medium Article:
638 # https://towardsdatascience.com/an-illustrative-introduction-to-fishers-
639 # linear-discriminant-9484efee15ac
640 # Putting Fisher's Discriminant Analysis into an optimization problem.
641 #
642 #
643 # ##### Theoretical Part References
644 # 1. Dim Reduce with Fisher's LDA
645 #     * Fisher's Formation of the LDA as an optimization problem.
646 #         * https://towardsdatascience.com/an-illustrative-introduction-to-
647 #             fishers-linear-discriminant-9484efee15ac
648 #     * Elements of Statistical Learning , Page 108
649 # 2. PCA Dim Reduce
650 #     * Eckar Young Theorem:
651 #         * Page 7 of Data science for Engineering.
652 #     * Alignment Problem , and Unitary Transformation:
653 #         * Page 35
654 # 3. PCA + LDA Dim reduce:
655 #     * cite : Transform Function from the sklearn library.
656 # 4. SVM:
657 #     * Elements of statistical learning: page 408
658 #     * SK learn for SVM complexity
659 # 5. Random Forest:
660 #     * Page 288, Elements of Statistical Learning.
661 #     * Weakness of decision tree. https://scikit-learn.org/stable/modules/
662 #         tree.html#tree
663 #     * Information gain and Entropy of decision tree: https://
664 #         towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946
665 #         b9c18c8
666 #

```

#### A.4 Codes for CNN training

```

1 # -*- coding: utf-8 -*-
2 """Custom-Model-digits.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1woOclxd3lykGKKJkfAmMaG95o8-xutUN
8 """
9
10 !git clone https://github.com/sorki/python-mnist
11 !./python-mnist/bin/mnist_get_data.sh
12 !pip3 install emnist keras-tuner
13 from emnist import extract_training_samples

```

```

14
15 print("Imported the EMNIST libraries we need!")
16
17 import os
18 import numpy as np
19 import matplotlib.pyplot as plt
20 import tensorflow as tf
21 from tensorflow import keras
22 from sklearn.model_selection import train_test_split
23 from keras.layers import *
24 from keras.models import *
25 from keras.optimizers import *
26 from keras.utils import to_categorical
27 from emnist import extract_training_samples, extract_test_samples
28
29
30 from sklearn.model_selection import GridSearchCV
31 from keras.wrappers.scikit_learn import KerasClassifier
32
33 import kerastuner
34
35 # Note: If you are using the Google Colab, test the GPU source
36 gpu_info = !nvidia-smi
37 gpu_info = '\n'.join(gpu_info)
38 if gpu_info.find('failed') >= 0:
39     print('Select the Runtime > "Change runtime type" menu to enable a GPU
accelerator, ')
40     print('and then re-execute this cell.')
41 else:
42     print(gpu_info)
43
44 train_X, train_y = extract_training_samples("digits");
45 test_X, test_y = extract_test_samples("digits");
46
47 # Normalise
48 train_X = train_X.astype('float32')
49 train_X /= 255
50 test_X = test_X.astype('float32')
51 test_X /= 255
52
53 #train_X, val_X, train_y, val_y = train_test_split(images, labels, test_size=
0.10, random_state=42)
54
55 # Reshape image for CNN
56 train_X = train_X.reshape(-1, 28, 28, 1)
57 test_X = test_X.reshape(-1, 28, 28, 1)
58
59 num_classes = np.unique(train_y)
60 train_y = to_categorical(train_y)
61 test_y = to_categorical(test_y)
62
63 def model_builder(hp):
64     model = keras.Sequential()
65     hp_filters = hp.Choice('filters', values=[32, 64, 128])

```

```

66     hp_kernel_size = hp.Choice('kernel_size', values=[3, 5])
67     hp_dropout_rate = hp.Float('rate', min_value=0, max_value=0.5, step=0.1)
68     model.add(Conv2D(filters=hp_filters, kernel_size=hp_kernel_size,
69                       activation='relu', kernel_initializer='he_uniform', input_shape=(28,
70                                         28, 1)))
71     model.add(MaxPooling2D((2, 2)))
72     model.add(Dropout(rate=hp_dropout_rate))
73     model.add(Conv2D(filters=hp_filters * 2, kernel_size=hp_kernel_size,
74                       activation='relu', kernel_initializer='he_uniform'))
75     model.add(MaxPooling2D((2, 2)))
76     model.add(Dropout(rate=hp_dropout_rate))
77     model.add(Flatten())
78     hp_units = hp.Choice('units', values=[64, 128, 256, 512])
79     model.add(Dense(units=hp_units, activation='relu', kernel_initializer='
80                   he_uniform'))
81     model.add(Dropout(rate=hp_dropout_rate))
82     model.add(Dense(10, activation='softmax'))

83
84
85 # Tune the learning rate for the optimizer
86 # Choose an optimal value from 0.01, 0.001, or 0.0001
87     hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

88     model.compile(optimizer=keras.optimizers.Adam(learning_rate=
89                     hp_learning_rate),
90                     loss=keras.losses.CategoricalCrossentropy(from_logits=True),
91                     metrics=['accuracy'])

92
93
94 tuner = kerastuner.Hyperband(model_builder,
95                               objective='val_accuracy',
96                               max_epochs=10,
97                               factor=3,
98                               directory='my_dir',
99                               project_name='intro_to_kt')
100 stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
101 tuner.search(train_X, train_y, epochs=50, validation_split=0.2, callbacks=[stop_early])

102
103 # Get the optimal hyperparameters
104 best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]
105
106
107 # Build the model with the optimal hyperparameters and train it on the data
108 # for 50 epochs
109 checkpoint_filepath = './checkpoint'
110 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
111     filepath=checkpoint_filepath,
112     save_weights_only=False,
113     monitor='val_accuracy',
114     mode='max',
115     save_best_only=True)

116
117 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

```

```

113 cb_list = [model_checkpoint_callback, early_stop]
114 model = tuner.hypermodel.build(best_hps)
115 history = model.fit(train_X, train_y, epochs=50, validation_split=0.2,
116 callbacks=cb_list)
117
118 val_acc_per_epoch = history.history['val_accuracy']
119 best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
120 print('Best epoch: %d' % (best_epoch,))
121
122 print(f" filters: {best_hps.get('filters')}\n kernel_size: {best_hps.get('kernel_size')}\n dropout_rate:{best_hps.get('rate')}\n units:{best_hps.get('units')}")
123
124 hypermodel = tuner.hypermodel.build(best_hps)
125
126 # Retrain the model
127 hypermodel.fit(train_X, train_y, epochs=best_epoch, validation_split=0.2)
128
129 eval_result = hypermodel.evaluate(test_X, test_y)
130 print("[test loss, test accuracy]:", eval_result)
131
132 # Retrieve the best model.
133 best_model = tuner.get_best_models(num_models=1)[0]
134
135 # Evaluate the best model.
136 eval_result_best = best_model.evaluate(test_X, test_y)
137 print("[test loss, test accuracy]:", eval_result_best)
138
139 hypermodel.save('custom_model_digits.h5')
140
141 history.history['val_accuracy']
142
143 y_pred = hypermodel.predict(test_X)
144 y_label = (y_pred > 0.5)
145
146 import sklearn.metrics as metrics
147 cm = metrics.confusion_matrix(test_y.argmax(axis=1), y_label.argmax(axis=1))
148
149 cm
150
151 best_model.summary()
152
153 # Build the model with the optimal hyperparameters and train it on the data
154 # for 50 epochs
155 checkpoint_filepath = "custom-model-digits-best.h5"
156 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
157     filepath=checkpoint_filepath,
158     save_weights_only=False,
159     monitor='val_accuracy',
160     mode='max',
161     save_best_only=True)
162 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

```

```

163 cb_list = [model_checkpoint_callback, early_stop]
164 model = tuner.hypermodel.build(best_hps)
165 history = model.fit(train_X, train_y, epochs=50, validation_split=0.2,
166 callbacks=cb_list)
167
168 val_acc_per_epoch = history.history['val_accuracy']
169 best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
170 print('Best epoch: %d' % (best_epoch,))
171
172 # plot loss during training
173 plt.subplot(211)
174 plt.title('Loss')
175 plt.plot(history.history['loss'], label='train')
176 plt.plot(history.history['val_loss'], label='val')
177 plt.legend()
178 # plot accuracy during training
179 plt.subplot(212)
180 plt.title('Accuracy')
181 plt.plot(history.history['accuracy'], label='train')
182 plt.plot(history.history['val_accuracy'], label='val')
183 plt.legend()
184
185 plt.tight_layout()
186 plt.show()
187
188 del model
189
190 from keras.models import load_model
191
192 # returns a compiled model
193 # identical to the previous one
194 model = load_model('custom-model-digits-best.h5')
195
196 type(history)
197
198 import pandas
199
200 pandas.DataFrame(history.history).to_csv("history.csv")
201
202 tmp = model.evaluate(test_X, test_y)
203 print("[test loss, test accuracy]:", tmp)
204
205 # -*- coding: utf-8 -*-
206 """Custom-Model-letters.ipynb
207
208 Automatically generated by Colaboratory.
209
210 Original file is located at
211 https://colab.research.google.com/drive/179ySpL-OSBjpHiLgD1UT3ieTqAmEmx-
212 """
213
214 !git clone https://github.com/sorki/python-mnist
215 !./python-mnist/bin/mnist_get_data.sh

```

```

12 !pip3 install emnist keras-tuner
13 from emnist import extract_training_samples
14
15 print("Imported the EMNIST libraries we need!")
16
17 import os
18 import numpy as np
19 import matplotlib.pyplot as plt
20 import tensorflow as tf
21 from tensorflow import keras
22 from sklearn.model_selection import train_test_split
23 from keras.layers import *
24 from keras.models import *
25 from keras.optimizers import *
26 from keras.utils import to_categorical
27 from emnist import extract_training_samples, extract_test_samples
28
29
30 from sklearn.model_selection import GridSearchCV
31 from keras.wrappers.scikit_learn import KerasClassifier
32
33 import kerastuner
34
35 # Note: If you are using the Google Colab, test the GPU source
36
37 gpu_info = !nvidia-smi
38 gpu_info = '\n'.join(gpu_info)
39 if gpu_info.find('failed') >= 0:
40     print('Select the Runtime > "Change runtime type" menu to enable a GPU
        accelerator, ')
41     print('and then re-execute this cell.')
42 else:
43     print(gpu_info)
44
45 train_X, train_y = extract_training_samples("letters");
46 test_X, test_y = extract_test_samples("letters");
47
48 # Normalise
49 train_X = train_X.astype('float32')
50 train_X /= 255
51 test_X = test_X.astype('float32')
52 test_X /= 255
53
54 #train_X, val_X, train_y, val_y = train_test_split(images, labels, test_size=
      0.10, random_state=42)
55
56 # Reshape image for CNN
57 train_X = train_X.reshape(-1, 28, 28, 1)
58 test_X = test_X.reshape(-1, 28, 28, 1)
59
60 num_classes = np.unique(train_y)
61 train_y = to_categorical(train_y)
62 test_y = to_categorical(test_y)
63
```

```

64 train_y = np.delete(train_y , 0, axis=1)
65 test_y = np.delete(test_y , 0, axis=1)
66
67 def model_builder(hp):
68     model = keras.Sequential()
69     hp_filters = hp.Choice('filters', values=[32, 64, 128])
70     hp_kernel_size = hp.Choice('kernel_size', values=[3, 5])
71     hp_dropout_rate = hp.Float('rate', min_value=0, max_value=0.5, step=0.1)
72     model.add(Conv2D(filters=hp_filters, kernel_size=hp_kernel_size,
73                      activation='relu', kernel_initializer='he_uniform', input_shape=(28,
74                      28, 1)))
75     model.add(MaxPooling2D((2, 2)))
76     model.add(Dropout(rate=hp_dropout_rate))
77     model.add(Conv2D(filters=hp_filters*2, kernel_size=hp_kernel_size,
78                      activation='relu', kernel_initializer='he_uniform'))
79     model.add(MaxPooling2D((2, 2)))
80     model.add(Dropout(rate=hp_dropout_rate))
81     model.add(Flatten())
82     hp_units = hp.Choice('units', values=[64, 128, 256, 512, 1024])
83     model.add(Dense(units=hp_units, activation='relu', kernel_initializer='
84                     he_uniform'))
85     model.add(Dropout(rate=hp_dropout_rate))
86     model.add(Dense(26, activation='softmax'))
87
88
89 # Tune the learning rate for the optimizer
90 # Choose an optimal value from 0.01, 0.001, or 0.0001
91 hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
92
93 model.compile(optimizer=keras.optimizers.Adam(learning_rate=
94                 hp_learning_rate),
95                 loss=keras.losses.CategoricalCrossentropy(from_logits=True),
96                 metrics=['accuracy'])
97
98 return model
99
100 tuner = kerastuner.Hyperband(model_builder,
101                             objective='val_accuracy',
102                             max_epochs=10,
103                             factor=3,
104                             directory='my_dir',
105                             project_name='hp-custom-model-letters')
106 stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
107 tuner.search(train_X, train_y, epochs=50, validation_split=0.2, callbacks=[stop_early])
108
109 # Get the optimal hyperparameters
110 best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]
111
112 # Build the model with the optimal hyperparameters and train it on the data
113 # for 50 epochs
114 checkpoint_filepath = './checkpoint'
115 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
116     filepath=checkpoint_filepath,

```

```

111     save_weights_only=False ,
112     monitor='val_accuracy' ,
113     mode='max' ,
114     save_best_only=True)
115
116 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
117
118 cb_list = [model_checkpoint_callback, early_stop]
119 model = tuner.hypermodel.build(best_hps)
120 history = model.fit(train_X, train_y, epochs=50, validation_split=0.2,
121                       callbacks=cb_list)
122
123 val_acc_per_epoch = history.history['val_accuracy']
124 best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
125 print('Best epoch: %d' % (best_epoch,))
126
127 print(f" filters: {best_hps.get('filters')} \n kernel_size: {best_hps.get('kernel_size')} \n dropout_rate:{best_hps.get('rate')} \n units:{best_hps.get('units')}")
128
129 hypermodel = tuner.hypermodel.build(best_hps)
130
131 # Retrain the model
132 hypermodel.fit(train_X, train_y, epochs=best_epoch, validation_split=0.2)
133
134 eval_result = hypermodel.evaluate(test_X, test_y)
135 print("[test loss, test accuracy]:", eval_result)
136
137 # Retrieve the best model.
138 best_model = tuner.get_best_models(num_models=1)[0]
139
140 # Evaluate the best model.
141 eval_result_best = best_model.evaluate(test_X, test_y)
142 print("[test loss, test accuracy]:", eval_result_best)
143
144 hypermodel.save('custom_model_digits.h5')
145
146 history.history['val_accuracy']
147
148 y_pred = hypermodel.predict(test_X)
149 y_label = (y_pred > 0.5)
150
151 import sklearn.metrics as metrics
152 cm = metrics.confusion_matrix(test_y.argmax(axis=1), y_label.argmax(axis=1))
153
154 cm
155 best_model.summary()
156
157 # Build the model with the optimal hyperparameters and train it on the data
158 # for 50 epochs
159 checkpoint_filepath = "custom-model-letters-best.h5"
160 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
161     filepath=checkpoint_filepath,

```

```

161     save_weights_only=False ,
162     monitor='val_accuracy' ,
163     mode='max' ,
164     save_best_only=True)
165
166 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
167
168 cb_list = [model_checkpoint_callback, early_stop]
169 model = tuner.hypermodel.build(best_hps)
170 history = model.fit(train_X, train_y, epochs=50, validation_split=0.2,
171                       callbacks=cb_list)
172
173 # val_acc_per_epoch = history.history['val_accuracy']
174 # best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
175 # print('Best epoch: %d' % (best_epoch,))
176
177 # plot loss during training
178 plt.subplot(211)
179 plt.title('Loss')
180 plt.plot(history.history['loss'], label='train')
181 plt.plot(history.history['val_loss'], label='val')
182 plt.legend()
183
184 # plot accuracy during training
185 plt.subplot(212)
186 plt.title('Accuracy')
187 plt.plot(history.history['accuracy'], label='train')
188 plt.plot(history.history['val_accuracy'], label='val')
189 plt.legend()
190
191 plt.tight_layout()
192 plt.show()
193
194 del model
195
196 from keras.models import load_model
197
198 # returns a compiled model
199 # identical to the previous one
200 model = load_model('custom-model-letters-best.h5')
201
202 type(history)
203
204 pandas.DataFrame(history.history).to_csv("history-custom-model-letters-best.csv")
205
206 tmp = model.evaluate(test_X, test_y)
207 print("[test loss, test accuracy]:", tmp)
208
209
210 #!/usr/bin/env python
211 # coding: utf-8
212
213
214 # In [1]:

```

```

5
6
7 from emnist import extract_training_samples , extract_test_samples
8 import os
9 import numpy as np
10 import matplotlib.pyplot as plt
11 import tensorflow as tf
12 from tensorflow import keras
13 from sklearn.model_selection import train_test_split
14 from keras.layers import *
15 from keras.models import *
16 from keras.optimizers import *
17 from keras.utils import to_categorical
18 import kerastuner as kt
19 import pandas as pd
20 from keras.models import load_model
21
22 from core import *
23
24 # returns a compiled model
25 # identical to the previous one
26 model = load_model('./CNN_Models/custom-model-digits-best.h5')
27
28
29 # In [2]:
30
31
32 test_X , test_y = extract_test_samples("digits");
33 test_X = test_X.astype('float32')
34 test_X /= 255
35 test_y = to_categorical(test_y)
36 test_X = test_X.reshape(-1, 28, 28, 1)
37
38
39 # In []:
40
41 model.summary()
42
43
44
45 # In [3]:
46
47
48 result = model.evaluate(test_X , test_y)
49 print("[ test loss , test accuracy ]:", result)
50
51
52 # In [4]:
53
54
55 y_pred = model.predict(test_X)
56 y_pred_label = np.argmax(y_pred , axis=1)
57 # y_pred_label = np.where(y_pred_label==1)[1]
58
```

```

59 - , test_y_label = extract_test_samples("digits");
60
61
62 # In [7]:
63
64
65 Conmat = ConfusionMatrix(test_y_label, y_pred_label)
66 Conmat.visualize()
67 Conmat.report()
68
69
70 # In [5]:
71
72
73 df=pd.read_csv('~/CNN-Models/history.csv')
74
75
76 # In [6]:
77
78
79 loss = df['loss'].to_numpy()
80 val_loss = df['val_loss'].to_numpy()
81 accuracy = df['accuracy'].to_numpy()
82 val_accuracy = df['val_accuracy'].to_numpy()
83
84
85 # In [77]:
86
87
88 # plot loss during training
89 plt.subplot(211)
90 plt.title('Loss')
91 plt.plot(loss, label='train')
92 plt.plot(val_loss, label='val')
93 plt.xlabel('epoch')
94 plt.legend()
95 # plot accuracy during training
96 plt.subplot(212)
97 plt.title('Accuracy')
98 plt.plot(accuracy, label='train')
99 plt.plot(val_accuracy, label='val')
100 plt.xlabel('epoch')
101 plt.legend()
102
103 plt.tight_layout()
104 plt.savefig('CNN-digits-best.pdf')
105 plt.show()
106
107
108 # In [ ]:
1
2 #!/usr/bin/env python
3 # coding: utf-8

```

```

4 # In [1]:
5
6
7 from emnist import extract_training_samples, extract_test_samples
8 import os
9 import numpy as np
10 import matplotlib.pyplot as plt
11 import tensorflow as tf
12 from tensorflow import keras
13 from sklearn.model_selection import train_test_split
14 from keras.layers import *
15 from keras.models import *
16 from keras.optimizers import *
17 from keras.utils import to_categorical
18 import kerastuner as kt
19 import pandas as pd
20 from keras.models import load_model
21
22 from core import *
23
24 # returns a compiled model
25 # identical to the previous one
26 model = load_model('./CNN_Models/custom-model-letters-best.h5')
27
28
29 # In [2]:
30
31
32 test_X, test_y = extract_test_samples("letters");
33 test_X = test_X.astype('float32')
34 test_X /= 255
35 test_y = to_categorical(test_y)
36 test_X = test_X.reshape(-1, 28, 28, 1)
37 test_y = np.delete(test_y, 0, axis=1)
38
39
40 # In []:
41
42
43 model.summary()
44
45
46 # In [3]:
47
48
49 result = model.evaluate(test_X, test_y)
50 print("[test loss, test accuracy]:", result)
51
52
53 # In []:
54
55
56 y_pred = model.predict(test_X)
57 y_pred_label = np.argmax(y_pred, axis=1)

```

```

58 # y_pred_label = np.where(y_pred_label==1)[1]
59
60 _, test_y_label = extract_test_samples("letters")
61
62
63 # In [5]:
64
65
66 df=pd.read_csv('./CNN_Models/history-custom-model-letters-best.csv')
67
68
69 # In [6]:
70
71
72 loss = df['loss'].to_numpy()
73 val_loss = df['val_loss'].to_numpy()
74 accuracy = df['accuracy'].to_numpy()
75 val_accuracy = df['val_accuracy'].to_numpy()
76
77
78 # In [7]:
79
80
81 # plot loss during training
82 plt.subplot(211)
83 plt.title('Loss')
84 plt.plot(loss, label='train')
85 plt.plot(val_loss, label='val')
86 plt.xlabel('epoch')
87 plt.legend()
88 # plot accuracy during training
89 plt.subplot(212)
90 plt.title('Accuracy')
91 plt.plot(accuracy, label='train')
92 plt.plot(val_accuracy, label='val')
93 plt.xlabel('epoch')
94 plt.legend()
95
96 plt.tight_layout()
97 plt.savefig('CNN-letters-best.pdf')
98 plt.show()
99
100
101 # In []:

```