AMATH 582 Final Project

Haoran Zhao, Hongda Li, Shuangshan Li

March 19, 2021

Abstract

In this paper, we will explore different techniques by analyzing the EMNISt digit images. Principal component analysis (PCA) and Linear Discriminant Analysis (LDA) are used to reduce dimensionally. Then three classifiers, support vector machine (SVM), Decision Tree Random Forest, and Neural Network are implementing to classify these data set.

1 Introduction and Overview

Our objective of the project is to explore and understand the EMNIST data set using various Machine Learning models and dimensinality 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], lowercased and uppercased letters. It supports 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 zero 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.

Images for the handwritten letters and digits are normalized and framed under a 28×28 pixel image.

1.2 Training of Models and Irreducible Errors

Training on multiple Classical Machine Learning models and modern machine learning models on the data set suggested the existence of Irreducible error, which arises in the form of ambiguity between a certain class of digits, lowercased and uppercased letters. For example, "2,z,Z", "5sS", "0oO", etc, where each of the symbols belongs to different labels but hard to distinguish when written by hands.

These ambiguous symbols not only create difficulties for modern machine learning models such as Neuro-nets, but it also creates confusion for classical models such as the SVM and Random Forest. Errors for these models suggest potential difficulties for other models such as neuro net or its variance, which is expensive to train.

2 Theoretical Background

In this section we will go through several of the popular machine learning models, dimensionality reduction techniques and measurements of performance metric. Major part of the discussion will centered around the mathematics and idea behind these subjects. The presentation is made with the hope to be informative about the mathematical background of the project, and how the mathematical understandings of lead to a better understanding of the structure and topology of the data via training on validating Machine Learning

Models. Our hypothesis is that, the performance Neuro Net works, or Convolutional Neuro Net work is as good as classical models if not better. In addition, we hypothesize the existence of Irreducible errors among 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 the feature space's low dimensional embeddings. The original feature space for each of the images lies in a 784 dimension because each pixel of the image is considered a feature and the image has size 28×28.

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 classes labels. One a high level, the problem is phrased as[10]:

$$\max_{\|x\|=1} \left\{ \frac{v^T S_{\text{Between}} v}{v^T S_{\text{Within}} v} \right\} \tag{1}$$

Where 2 of the matrices are defined as:

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

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

Where quantity μ_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:

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

$$m_i = \sum_{x_i \in C_i} x_i \tag{4}$$

Notice that equation Expression 1 is the key here, the solution to the optimization problem is Eigenvector of the maximal Eigenvalue for the matrix $S^{\dagger}_{\text{within}} S_{\text{between}}$, 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 requires the classes of the labels to have Gaussian Distributions of the same covariance matrix for each of the class[14], in addition, it cannot separate non-linear distributions.

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 decompose any matrix A into the

product of 3 matrices: $U\Sigma V^T$, where U, V^T are orthogonal matrices and Σ 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 ΣV^T is the A matrix is the data matrix and $U\Sigma$ if the A matrix 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]:

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

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

This is incredibly useful in a sense that, if the variance of the data set can be explained by some kind of low rank structure, then those structure will be reviewed in the Principal Components, the U, or the V matrix.

Principal Components 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 combineing 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 data structure. Classical Machine Learning models such as the SVM and LDA have complexity that scales with the number of features, samples, and labels involved. Hence creating an 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, this is done using the training set. This can be easily carried out using *sklearn*

When making a prediction with a model, the test set 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:

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

The quantity β, β_0 are parameters that define the separating hyperplane, and the slack variables ξ_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 with the KTT (Karush-Kuhn-Tucker conditions) is used.

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:

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

And the constraints of the primal problem is relaxed using positivity assumptions 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 (β_i, ξ_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 and inner products o all the features, and it gives rise to the use of the Kernel function.

In practice, SVM is very effective under the task of binary classification under 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 both dimension and the number of samples used for training are very large, and the complexity is given by $\mathcal{O}(mn^2)$ where n denotes the number of samples and m denotes the number of features[8].

To overcome this, we use the 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 classification 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.

2.6 Decision Tree and Random Forest

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

Under the most basic premise, each node of the of the tree, it split the samples into 2 sets trying to maximize the information gain when splitting. 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.

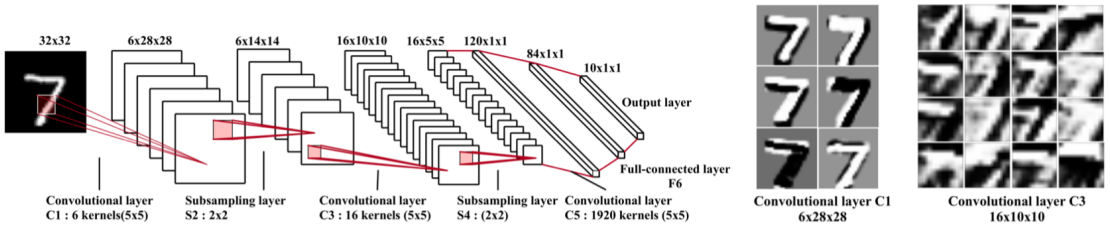
To make a branch on the tree, a separation criterion is made on one of the feature, and it seeks to maximizes the information gain, which can be summarized as one expressions[13]:

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

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 features are chosen to split and the parameters, many different types of tree can be configured. In general, the construction of the optimal tree is an NP-Hard problem. Hence usually, heuristics are used for constructing the tree given a training sample[9].

However, the power of tree is manifested by a forest. The idea is to use 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



(a) An typical example of CNN, LeNet-5.

(b) Visualization of features in the LeNet-5 network

Figure 1: A typical CNN LeNet-5

trained from the data (Committee Method), or we weight their votes using their 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 a multi-class classification. 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 a 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 therms, the feature value, $z^l_{i,j,k}$, at a certain location (i, j) in the k^{th} feature map of the l^{th} layer is[3]:

$$z^l_{i,j,k} = \mathbf{w_k^{l^T}} \mathbf{x^l_{i,j}} + b^l_k \tag{1}$$

where $\mathbf{w^l_k}$ and b^l_k are the weight vector and bias term of the k^{th} filter of the l^{th} layer respectively. The activition function introduces nonlinerities to CNN, which are importrant for multi-layer networsk to detect nonliner features. Let $a()$ denote the nonliner activation function, it applies to the convolutional feature $z^l_{i,j,k}$:

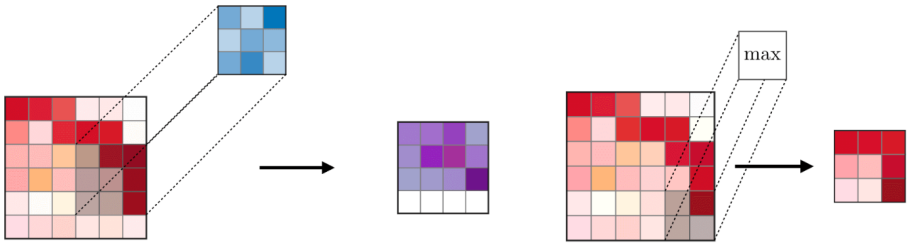
$$a^l_{i,j,k} = a(z^l_{i,j,k}) \tag{2}$$

Typical activation function are sigmodid, tanh and ReLU. In this project we use ReLU with the mathematical form:

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

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



(a) Convolutional layer (b) MaxPooling

Figure 2: Different layers

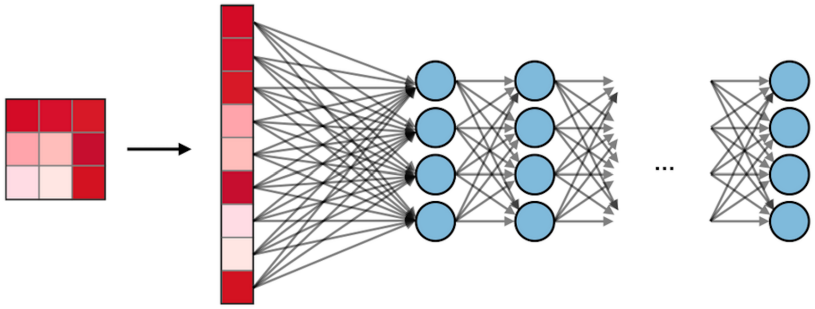


Figure 3: Illustration of fully connected layer

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. Thies 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:

1. **Epoch**: Epoch is a term used to refer 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.

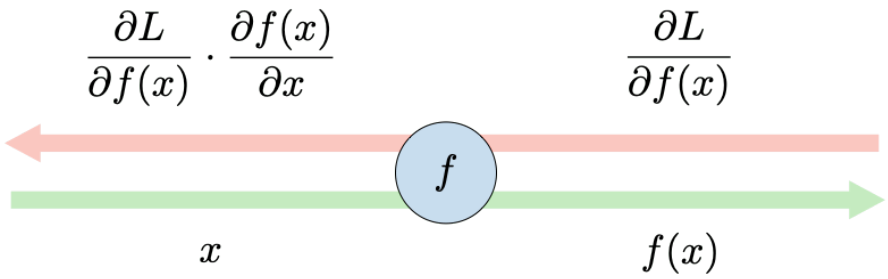


Figure 4: Illustration of backpropagation

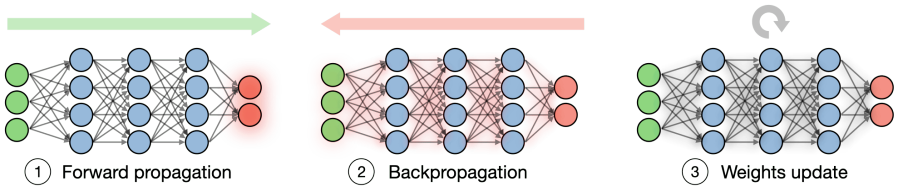


Figure 5: Illustration of weight update

Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter 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)L(z,y)L(z,y) is commonly used and is defined as follows:

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

Backpropagation: Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight w is computed using the chain rule.

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

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: Backpropagate 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: $M_{i,j}$ the number of sample with label i are being misclassified as label j.

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

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

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:

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

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

Input: ClassSize, TestSet, Classes, Shuffle(Optional)
Images, Labels = EMNIST Test set if TestSet else EMNIST Training set
Initialize: SelectedIndices
for L in Classes **do**
    Indices = np.where(Labels==L)
    Reshape Indices into a 1-D numpy array
    if Shuffle **then**
        Shuffle Indices
    end if
    Indices = Indices[min(Indices.size, ClassSize)]
    SelectedIndices.append(Indices)
end for
RowDataMatrix := Images[Indices, ::, ::]
RowDataMatrix := RowDataMatrix as type np.floats
Labels := Labels[Indices]
RowDataMatrix := RowDataMatrix/255
RowDataMatrix := RowDataMatrix - mean(RowDataMatrix, axis=0)
Return: RowDataMatrix, Labels

The above algorithm Algorithm 1 selects evenly distributed samples from the ENMIST training or test set. The variable "ClassSize" indices the minimal 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 rescale the data under the interval $[0, 1]$.

Algorithm 2 Another alternative algorithm is used to transform the data set into subclasses with different groupings. For example, we investigated the classification between the larger set of classes: 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").

Algorithm 2: Algorithm 2: Subclass Classification

Input: AuxFunc, TestSet, ClassSize
Mapping, NewAxisLabels, NewLabels
Images, Labels := Images and labels from the EMNIST test set if TestSet else from EMNIST Training Set
TransFormedLabels = np.vectorize(Mapping)(Labels)
Initialize: IndicesChosen
for II in NewLabels **do**
    Idx = np.where(TransformedLabels = II)
    Reshaoe Idx so it's 1-d Numpy array
    Shuffle Idx
    Idx := Idx[:min(idx.size, classsize)]
    Merge Idx into: IndicesChosen
end for
Images := Images[IndicesChosen, ::, ::]
LabelsChosen := Labels[IndicesChosen]
DataMatrix := Reshape Images and standardize Images
Return: DataMatrix, Mapping(Labels), LabelsChosen, NewLabels, 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

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

x

This 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.

3.4 Training CNN Models

In these project we want to classify 2 cases, 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**.

Algorithm 4: Algorithm 4: Training Models

TrainX, TrainY = Using Algorithm 1 orAlgorithm 2 to get the training, labels set.
TestX, TestY = Using Algorithm 1 orAlgorithm 2 to get the testing, labels set.
DimReduce := Get a model by fitting TrainX, TrainY data using the LDA or PCA in sklearn
EmbeddingsTrain := DimReduce.getEmbeddings(TrainX)
EmbeddingsTest := DimReduce.getEmbeddings(TestX)
Model := Initialize a Model using sklearn
Model := Train on EmbeddingsTrain
PredictedLabelsTrain = Model.predict(TrainEmbeddings)
PredictedLabelsTest = Model.predict(TestEmbeddings)

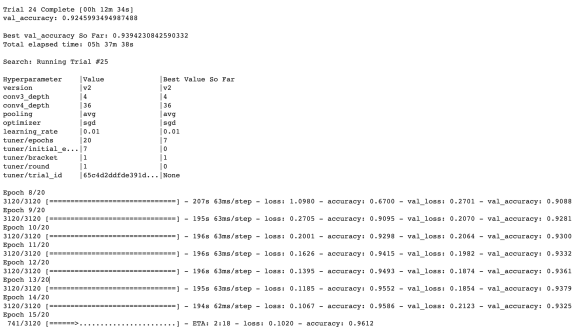


Figure 6: Optimization of Xception

Then our next is to determine the architecture of the CNN model. Finally, we decide our lay are 2 CONV layers with Maxpooling 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 test 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 have no powerful accelerator 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 finishes.

The search method is Hyperband[6], which is a novel approach to do 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.

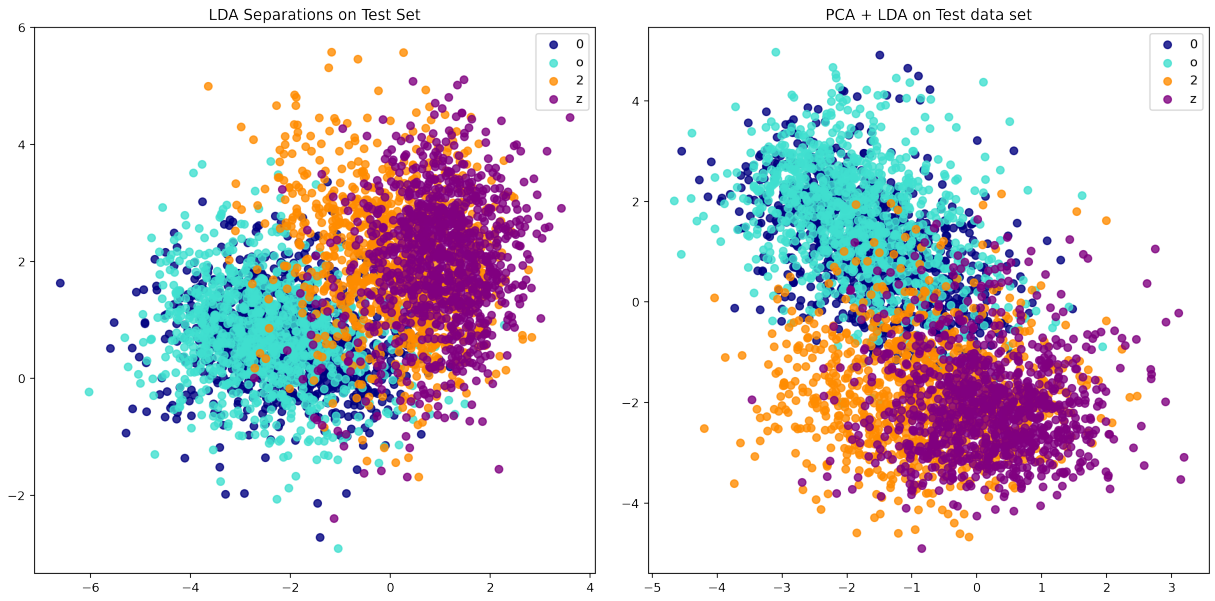
Finally we use the retrained best model to predict test data set and evaluate the performances of the CNN classifirers.

Here I show one plot of hypertuning Xception with keras tuner. The tuning process is pretty 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, O overlaps with each other. Comparatively speaking the projection onto top 2 principal components show



(a) LDA Subspace Projection

(b) PCA + LDA Projection

Figure 7: CNN Training curves

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".

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

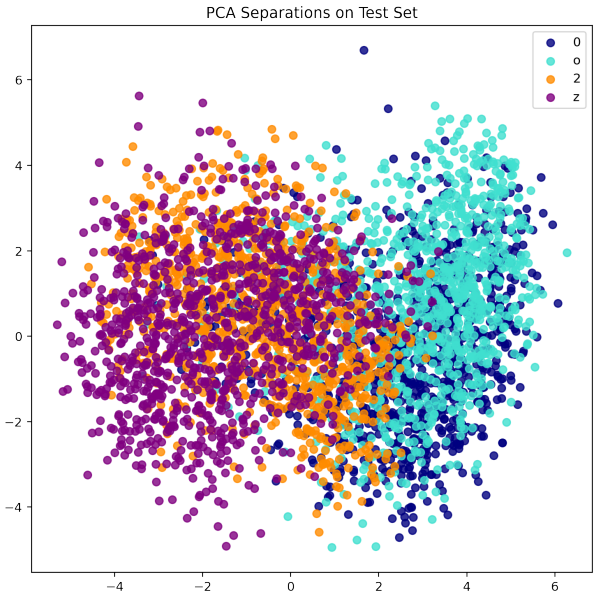


Figure 8: Projection of the PCA Embeddings

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 train 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 earlystopping 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.

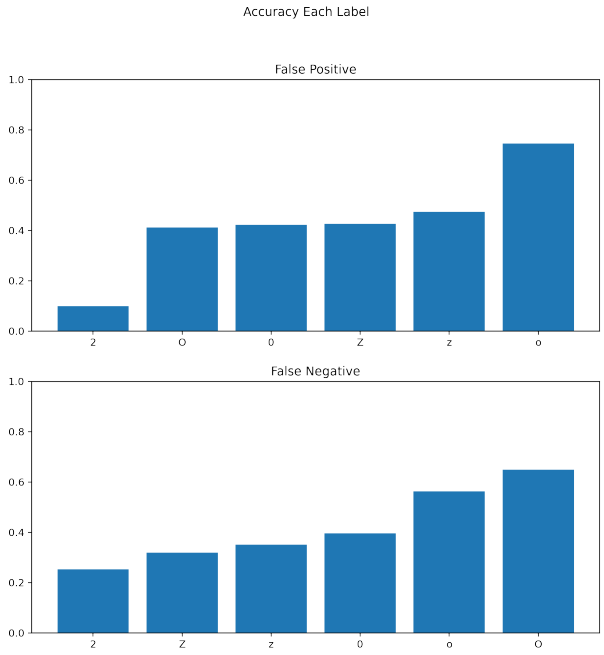
With the help of the ModelCheckpoint callback defined by the keras, we save the best models with highest validation accuracy. And finally we get very good results on test dataset 99.63% and 98.88% for digits and letters. These results are impressive.

5 Summary and Conclusions

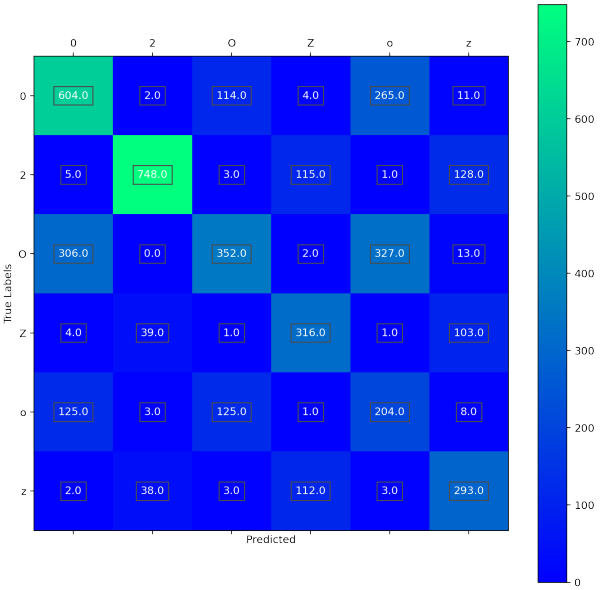
In this project, we explore the usage of SVD and PCA, as well as the CNN on EMNIST dataset. These classifiers perform well. The dimensionality reduction techniques were proven to be effective for speeding up the algorithm by several magnitudes, it also provides more intuitive understanding on the structure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

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

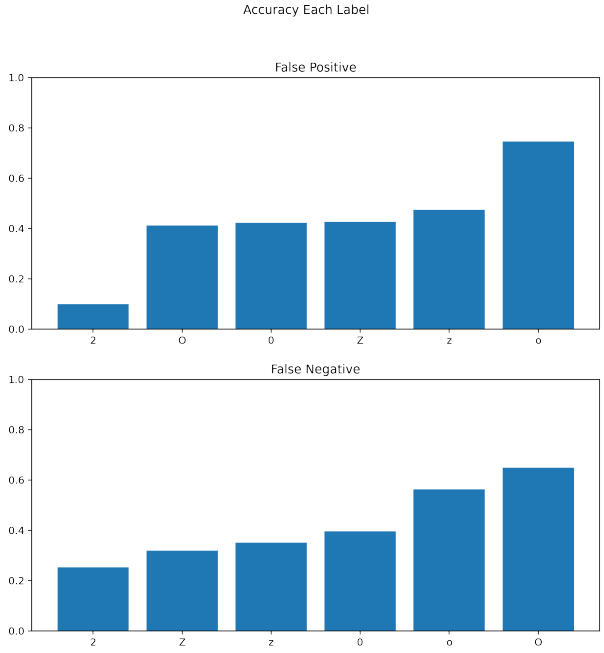


(a) SVM Hardest Separating Symbols Bar Plot

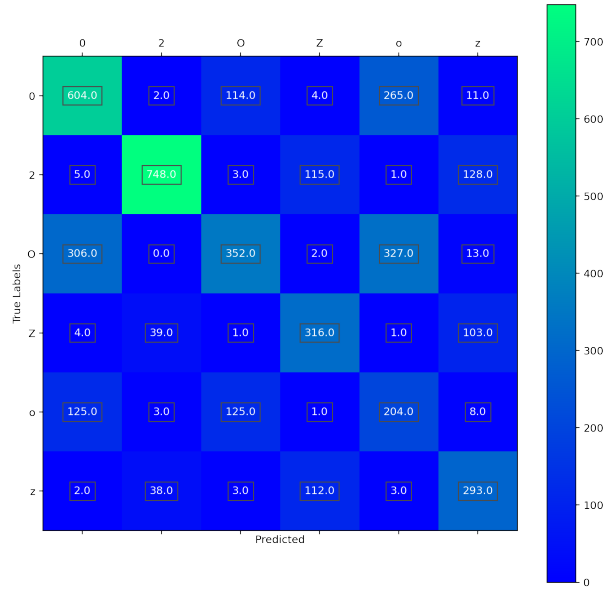


(b) SVM Hardest Separating Symbols Confusion Matrix

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

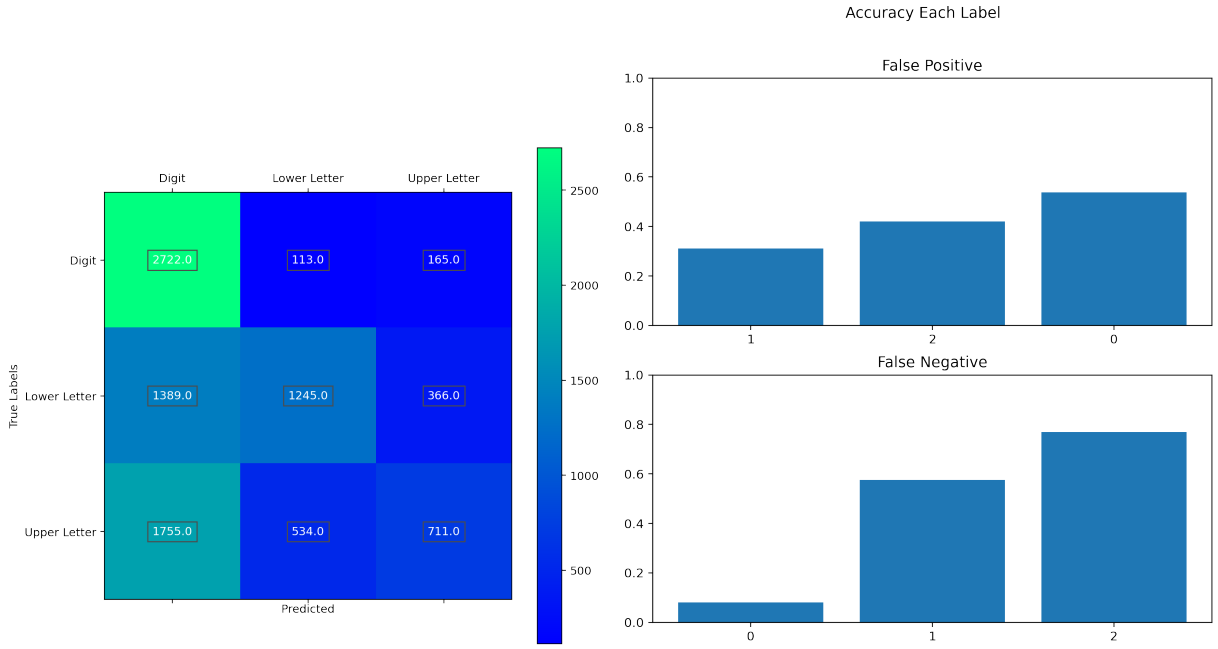


(a) Hardest Random Forest, Bar Plot



(b) Hardest Random Forest, Confusion Matrix

Figure 10: Hardest Separations using Random Forest



(a) An typical example of CNN, LeNet-5.

(b) Visualization of features in the LeNet-5 network

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

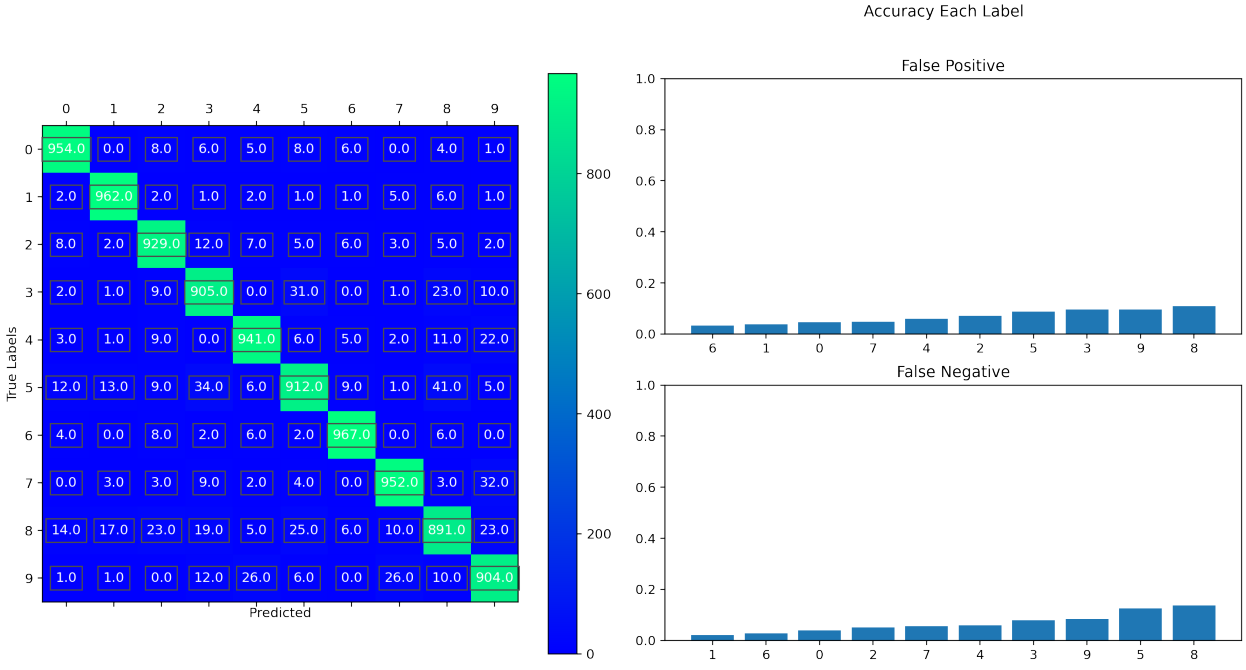
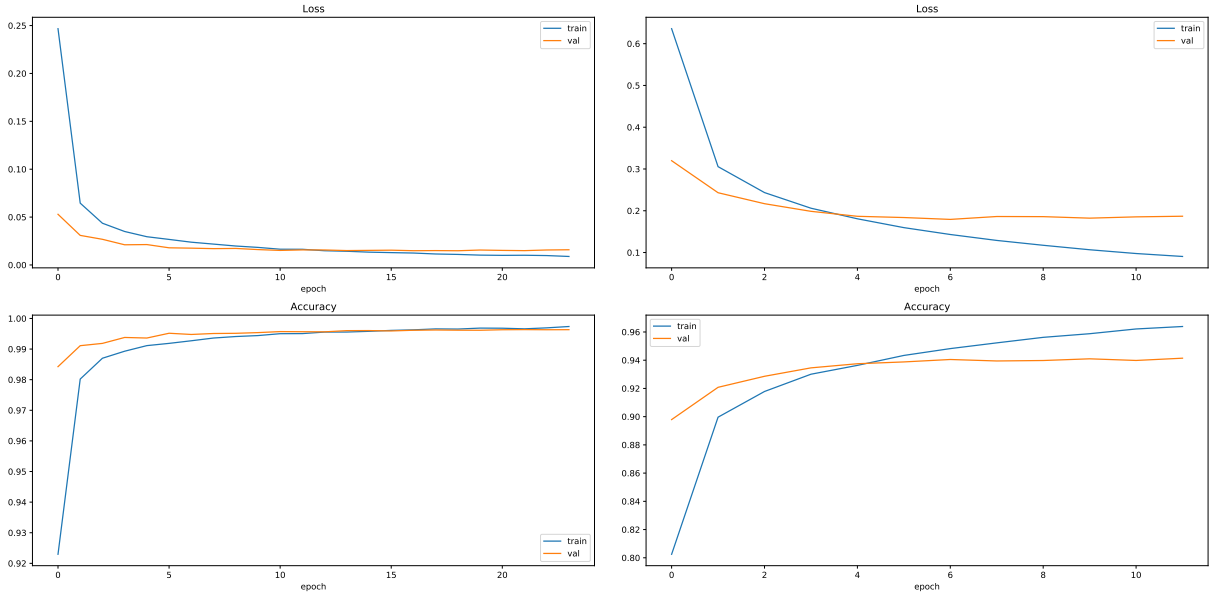


Figure 12: SVM: Separations on Digit

```
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
_________________________________________________________________
```

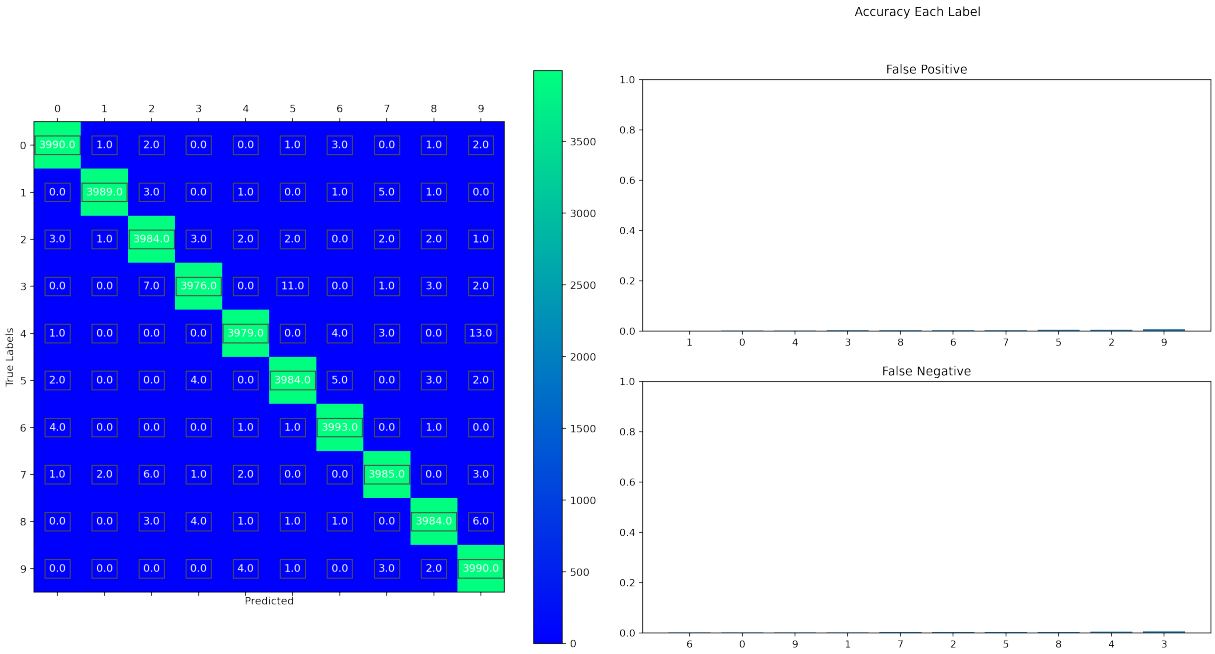
Figure 13: CNN model



(a) CNN classifications of digits

(b) CNN classifications of letters

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

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. 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. 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 Deisions*. 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

```
from emnist import extract_training_samples, extract_test_samples
import os
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import matplotlib
```

```
from collections import Counter

from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, plot_confusion_matrix

## name space stuff
diag = np.diag
shuffle = np.random.shuffle
reshape = np.reshape
zeros = np.zeros
scatter = plt.scatter
show = plt.show
concatenate = np.concatenate
mean = np.mean
figure = plt.figure
title = plt.title
legend = plt.legend
matshow = plt.matshow
array_equal=np.array_equal
unique = np.unique
ylim = plt.ylim

## Meta settings
CURRENT_DIRECTORY = None
UPPER_LETTERS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
LOWER_LETTERS = "abcdefghijklmnopqrstuvwxyz"
DIGITS = "0123456789"
matplotlib.rcParams['figure.figsize'] = (10, 10)
matplotlib.rcParams["figure.dpi"] = 220


def THREE_CLS_AUX_FUN():
    """
        Given this function, it returns a function that does the label mapping
        and an array for the axis labels. (In the same order as the labels)

        Maps the labels to these classes:
        1. Is digit
        2. Is smaller case letter.
        3. Is cap letter.
    """
    def SplitByFunc(label:int):
        if LabelsToSymbols(label) in DIGITS:
            return 0

        elif LabelsToSymbols(label) in LOWER_LETTERS:
            return 1

```

```
61          elif LabelsToSymbols(label) in UPPER_LETTERS:
62              return 2
63          else:
64              raise Exception("Something is wrong this should not happen check
                    code.")
65      AxisLabels = ["Digit", "Lower Letter", "Upper Letter"]
66      return SplitByFunc, AxisLabels, [0, 1, 2]
67
68
69  def TWO_CLS_AUX_FUN():
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,
                this function
90          will return the splited data set for each.
91          Note:
92              Data is chosen from the training set.
93      :param classes:
94          An array for the labels you want to filter out from the EMNISt data
                set.
95      :param classSize:
96          For each of the labels, what is the max size we want for each of the
                labels.
97      """
98      if test_set:
99          images, labels = extract_test_samples("byclass")
100     else:
101         images, labels = extract_training_samples("byclass")
102     Idx = []
103     classes = list(range(0, 62)) if classes is  None else classes
104     for Label in classes:
105         ClsIdx = np.argwhere(labels == Label)
106         ClsIdx = reshape(ClsIdx, ClsIdx.shape[0])
107         if shuffle_data: shuffle(ClsIdx)
108         ClsIdx = ClsIdx[: min(ClsIdx.shape[0], classSize)]
109         for Idex in ClsIdx:
```

```
            Idx.append(Idex)
    RowDataMtx = images[Idx, :, :]
    Labels = labels[Idx]
    RowDataMtx = reshape(RowDataMtx, (len(Idx), 28*28))
    RowDataMtx = RowDataMtx.astype(np.float)
    RowDataMtx /= 255
    RowDataMtx -= mean(RowDataMtx, axis=1, keepdims=True)
    return RowDataMtx, Labels


def SplitbyLetterDigits(classSize=100, test_set=False):
    """
    Split the training/test data set into 3 big groups:
        1. The digits.
        2. The Upper cased letters.
        3. The lower cased letters.

    Note: This class will always shuffle the data!
    Note: This class will perform zero mean unit variances STANDARDIZATION.

    :param classSize:
        The maximal size for each of the classes.
    :param test_set:
        Whether to choose the data set from the EMNISt test set, or the data
            set.
    :Return :
        The returned labels will still be in the range of [0, 62], but they
        will be evenly partitioned into the 3 classes above.
    """
    if test_set:
        images, labels = extract_test_samples("byclass")
    else:
        images, labels = extract_training_samples("byclass")

    Organizer = LabelsOrganizer()
    TreeClassesLabels = Organizer.getDataLabels(labels)
    IndicesChosen = []
    for II in range(3):
        Idx = np.where(TreeClassesLabels==II)
        Idx = Idx[0]
        shuffle(Idx)
        Idx = Idx[:min(len(Idx), classSize)]
        for JJ in Idx:
            IndicesChosen.append(JJ)

    RowDataMtx = images[IndicesChosen, ::, ::]
    Labels = labels[IndicesChosen]
    RowDataMtx = reshape(RowDataMtx, (len(IndicesChosen), 28 * 28))
    RowDataMtx = RowDataMtx.astype(np.float)
    RowDataMtx /= 255
    RowDataMtx -= mean(RowDataMtx, axis=1, keepdims=True)
    return RowDataMtx, Labels


```

```
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, this
166         function will split the samples by the function.
167         1. Always shuffle the data.
168         2. The classSize is the maximal size for each class, if given class is less than it then
169         all the samples of the class will be given and homegnenuity will not be asserted.
170         3. images and labels will be normalized.
171 
172         :param classSize:
173             The maixmal size for each of the classes.
174         :param test_set:
175             Whether to choose the labels and images from the ENMIST test set.
176         :param auxFun:
177             The auxilary function for sub-classes of all 62 labels.
178 
179     """
180     auxFun, axisLabels, NewLabels = auxFun()
181     if test_set:
182         images, labels = extract_test_samples("byclass")
183     else:
184         images, labels = extract_training_samples("byclass")
185     TransformedLabels = np.vectorize(auxFun)(labels)
186     IndicesChosen = []
187     for II in NewLabels:
188         Idx = np.where(TransformedLabels==II)
189         Idx = Idx[0]
190         shuffle(Idx)
191         Idx = Idx[:min(len(Idx), classSize)]
192         for JJ in Idx:
193             IndicesChosen.append(JJ)
194 
195     RowDataMtx = images[IndicesChosen, ::, ::]
196     LabelsChosen = labels[IndicesChosen]
197     RowDataMtx = reshape(RowDataMtx, (len(IndicesChosen), 28 * 28))
198     RowDataMtx = RowDataMtx.astype(np.float)
199     RowDataMtx /= 255
200     RowDataMtx -= mean(RowDataMtx, axis=1, keepdims=True)
201     return RowDataMtx, \
202             np.vectorize(auxFun)(LabelsChosen), \
203             LabelsChosen, \
204             NewLabels, \
205             axisLabels
206 
207 
208 def SymbolsToLabels(symbol:str, SymToLabel=dict()):
209     assert len(symbol) == 1
210     if len(SymToLabel) != 0:
211         return SymToLabel[symbol]
212     Letters = "".join([chr(97 + II) for II in range(26)])
213     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)/(
                np.sum(Conmat, axis=0))
240         # False negative, none diag column sum, it's not but actually it is.
241         this.FalseNegativeEach = np.sum(Conmat - diag(diag(Conmat)), axis=1)/(
                np.sum(Conmat, axis=1))
242         LabelsSorted = unique(this.TestLabels.copy())
243         np.sort(LabelsSorted)
244         if axisTicks is None:
245             Ticks = [LabelsToSymbols(II) for II in LabelsSorted]
246             this.AxisTicks = Ticks
247         else:
248             this.AxisTicks = axisTicks
249 
250     def visualize(this, title:str=None):
251         """
252             Visualize the confusion matrix.
253         :return:
254             fig, ax
255         """
256         fig = figure()
257         ax = fig.add_subplot(111)
258         im = ax.matshow(this.ConfusionMatrix, cmap='winter', interpolation='
                nearest')
259         fig.colorbar(im)
260 
261         for (i, j), z in np.ndenumerate(this.ConfusionMatrix):
262             ax.text(j, i, '{:0.1f}'.format(z), ha='center', va='center', color
                    ="white",
263                     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         1. 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 LOWER_LETTERS:
316                 return 1
317             elif LabelsToSymbols(label) in UPPER_LETTERS:
```

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

```
368         if (X is None) or (y is None):
369             X, y = SplitbyClasses(classSize=classSize, classes=classes,
                    shuffle_data=shuffle)
370         Template = LDA(n_components=n_components)
371         lda = Template.fit(X, y)
372         this.LdaModel = lda
373         this.Dim = n_components
374         this.ClassSize = classSize
375         this.Data = X
376         this.Labels = y
377 
378     def getEmbeddings(this, toTransform=None):
379         """
380         Get the embeddings for the given data
381         :param toTransform: (Optional) if not given, it will return the
                embeddings of the data that the
382         LDA model is trained on.
383         :return:
384             The embeddings of the data. Each row is a sample.
385         """
386         lda = this.LdaModel
387         if toTransform is not None:
388             Embeddings = lda.transform(toTransform)  # Rows are embeddings in
                    all 60 dimensions
389             return Embeddings
390         else:
391             return lda.transform(this.Data)
392 
393 
394 class PCADimReduce:
395 
396     def __init__(this,
397                  X=None,
398                  y=None,
399                  n_components=0.9,
400                  classSize=1000,
401                  classes=None,
402                  shuffle=False
403                  ):
404         """
405             Create an instance for the PCA dimn reduce embedding.
406         :param X:
407         :param y:
408         :param n_components:
409         :param classSize:
410         :param classes:
411         """
412         if (X is None) or (y is None):
413             X, y= SplitbyClasses(classSize=classSize, classes=classes,
                    shuffle_data=shuffle)
414         this.PcaModel = PCA(n_components=n_components, svd_solver="full")
415         this.PcaModel.fit(X, y)
416         this.classSize = 1000
417         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,
                pca_components=0.9, shuffle=False):
435             if (X is None) or (y is None):
436                 X, y = SplitbyClasses(classSize=classSize, classes=classes,
                        shuffle_data=False)
437 
438             this.PCAModel = PCA(n_components=pca_components)
439             this.PCAModel.fit(X, y)
440             PCAEmbeddings = this.PCAModel.transform(X)
441             this.LDAModel = LDA()
442             this.LDAModel.fit(PCAEmbeddings, y)
443 
444             this.classSize = 1000
445             this.pca_components=0.9
446             this.Data = X
447             this.Labels = y
448 
449         def getEmbeddings(this, X=None):
450             X = this.Data if X is None else X
451             PCAEmbeddings = this.PCAModel.transform(X)
452             LDAEmbeddings = this.LDAModel.transform(PCAEmbeddings)
453             return LDAEmbeddings
454 
455 
456     def main():
457         pass
458 
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 sepration
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"]
```

```
SeparatingModes = [0, 5]
for color, II in zip(colors, classes):
    scatter(
        Embeddings[Labels == II, SeparatingModes[0]],
        Embeddings[Labels == II, SeparatingModes[1]],
        alpha=.8,
        color=color
    )
legend(list(Symbols))
title("PCA Separations on Test Set")
show()
print("Demonstration ended. ")


#### PCA + LDA Separations

# In[4]:


PCAInstance = PCADimReduce(n_components=0.8); print("Getting PCA Model...")
PCAEmbeddings = PCAInstance.getEmbeddings(); print("Getting PCA Embeddings..."
    )

TrainLabels = PCAInstance.Labels
print("Train LDA on PCA Embeddings... ")
LDAMaxComponents = min(61, PCAInstance.PcaModel.n_components_ - 1)
LDAInstance = LDADimReduce(n_components=LDAMaxComponents,  # Just to be sure.
                           X=PCAEmbeddings,
                           y=TrainLabels)  # Use the PCA embeddings to train
                               LDA

classes = [SymbolsToLabels(Char) for Char in "0o2z"]
TestData, TestLabels = SplitbyClasses(classSize=1000, classes=classes)
# Ge the LDA embeddings of the PCA embeddings.
print("Represent using PCA modes and then on LDA basis... ")
Embeddings = LDAInstance.getEmbeddings(PCAInstance.getEmbeddings(TestData))
colors = ['navy', 'turquoise', 'darkorange', "purple"]

## Plotting the Results
SeparatingModes = [0, 1]
for color, II in zip(colors, classes):
    scatter(
        Embeddings[TestLabels == II, SeparatingModes[0]],
        Embeddings[TestLabels == II, SeparatingModes[1]],
        alpha=.8,
        color=color
    )
legend(list("0o2z"))
title("PCA + LDA on Test data set")
show()


#### PCA + LDA Embeddings Version 2
#
```

```
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
        embeddings.
122
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
        Model.
129
130 TrainX, TrainY = SplitbyClasses(classes=classes, classSize=1000)
131 TestX, TestY = SplitbyClasses(classes=classes, classSize=500, shuffle_data=
        True, test_set=True)
132
133 DimRe = LDADimReduce(X=TrainX, y=TrainY)  # Use Train set to create LDA
        embeddings.
134 TrainEmbeddings = DimRe.getEmbeddings()  # Get Embeddings from the set trained
        LDA
135 TestEmbeddings = DimRe.getEmbeddings(TestX) # Get the embeddings from the test
        set.
136
137 Model.fit(TrainEmbeddings, TrainY); print(f"Modeling Has been fitted: ")
138 print(Model)
139
140
141 # In[3]:
142
143
144 matplotlib.rcParams['figure.figsize'] = (20, 20)
145 PredictedLabels = Model.predict(TestEmbeddings)
146 Conmat = ConfusionMatrix(PredictedLabels, TestY)
147 Conmat.visualize()
148 Conmat.report()
149
150
151 # In[7]:
152
153
154 PredictedLabels = Model.predict(TrainEmbeddings)
155 Conmat = ConfusionMatrix(PredictedLabels, TrainY)
156 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
```

```
DisplayLabels = "".join(map(str, range(10))) + "abcdefghijklmnopqrstuvwxyz"
classes = [SymbolsToLabels(II) for II in DisplayLabels]
Model = make_pipeline(StandardScaler(), SVC(gamma="auto"))  # Making the SVC Model.

TrainX, TrainY = SplitbyClasses(classes=classes, classSize=1000)
TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=True, test_set=True)

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

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


# In[12]:


matplotlib.rcParams['figure.figsize'] = (30, 30)
PredictedLabels = Model.predict(TestEmbeddings)
Conmat = ConfusionMatrix(PredictedLabels, TestY)
Conmat.visualize()
Conmat.report()


# In[13]:


matplotlib.rcParams['figure.figsize'] = (30, 30)
PredictedLabels = Model.predict(TrainEmbeddings)
Conmat = ConfusionMatrix(PredictedLabels, TrainY)
Conmat.visualize()
Conmat.report()


#### Linear SVM with LDA for [0-9] + [a-z] + [A-Z]
#
# Training the Model

# In[12]:


DisplayLabels = DisplayLabels =    "".join(map(str, range(10))) + "abcdefghijklmnopqrstuvwxyz" + "abcdefghijklmnopqrstuvwxyz".upper()
classes = [SymbolsToLabels(II) for II in DisplayLabels]
Model = make_pipeline(StandardScaler(), LinearSVC())  # Making the SVC Model.

TrainX, TrainY = SplitbyClasses(classes=classes, classSize=2000)
TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=
```

```
    True,  test_set=True)

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

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


#### Confusion Matrix on the Test set

# In[5]:


# Graphics settings!
matplotlib.rcParams['figure.figsize'] = (40, 40)
PredictedLabels = Model.predict(TestEmbeddings)
Conmat = ConfusionMatrix(PredictedLabels, TestY)
Conmat.visualize()
Conmat.report()


# In[6]:


PredictedLabels = Model.predict(TrainEmbeddings)
Conmat = ConfusionMatrix(PredictedLabels, TrainY)
Conmat.visualize()
Conmat.report()


#### SVM on Hardest Symbols to Separate

# In[7]:


DisplayLabels = DisplayLabels = "0oOz25sS"
classes = [SymbolsToLabels(II) for II in DisplayLabels]
Model = make_pipeline(StandardScaler(), SVC(gamma="auto"))  # Making the SVC Model.

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

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

```
    set.
299
300 Model.fit(TrainEmbeddings, TrainY); print(f"Modeling Has been fitted: ")
301 print(Model)
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 inhrant draw backs of a single decision tree, here we
      will explore the performance of Ensemble Tree under a optimal dimeision
      for embeddings.
370 #
371 # #### Hardest Separations on Random Forest:
372 # With PCA + LDA Dimensoinality Reduction.
373
374 # In[3]:
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
      set to create LDA embeddings.
383 TrainEmbeddings = DimRe.getEmbeddings()  # Get Embeddings from the set trained
       LDA
384 TestEmbeddings = DimRe.getEmbeddings(TestX)  # Get the embeddings from the
      test set.
385 Model = RandomForestClassifier(n_estimators=200, n_jobs=-1,  min_samples_leaf
      =10)
386 Model.fit(TrainEmbeddings, TrainY)
387
388
389 # In[4]:
390
391
392 PredictedTestLabels = Model.predict(TestEmbeddings)
393 Conmat = ConfusionMatrix(TestY, PredictedTestLabels)
```

```
Conmat.visualize()
Conmat.report()


# In[5]:


PredictedTestLabels = Model.predict(TrainEmbeddings)
Conmat = ConfusionMatrix(TrainY, PredictedTestLabels)
Conmat.visualize()
Conmat.report()


# #### Random Forest Separating all labels with PCA + LDA
# * This is hard.

# In[3]:


Symbols = "".join(map(str, range(10))) + "abcdefghijklmnopqrstuvwxyz" + "abcdefghijklmnopqrstuvwxyz".upper()
classes = [SymbolsToLabels(II) for II in Symbols]
TrainX, TrainY = SplitbyClasses(classes=classes, classSize=2000)
TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=True, test_set=True)

DimRe = DimReduceHybrid(X=TrainX, y=TrainY, pca_components=0.9)  # Use Train set to create LDA embeddings.
TrainEmbeddings = DimRe.getEmbeddings()  # Get Embeddings from the set trained LDA
TestEmbeddings = DimRe.getEmbeddings(TestX)  # Get the embeddings from the test set.
Model = RandomForestClassifier(n_estimators=200, n_jobs=-1, min_samples_leaf=1/(1000*62))
Model.fit(TrainEmbeddings, TrainY)


# In[4]:


matplotlib.rcParams['figure.figsize'] = (60, 60)
PredictedTestLabels = Model.predict(TestEmbeddings)
Conmat = ConfusionMatrix(TestY, PredictedTestLabels)
Conmat.visualize()
matplotlib.rcParams['figure.figsize'] = (30, 30)
Conmat.report()


# In[5]:


matplotlib.rcParams['figure.figsize'] = (60, 60)
PredictedTestLabels = Model.predict(TrainEmbeddings)
Conmat = ConfusionMatrix(TrainY, PredictedTestLabels)
```

```
Conmat.visualize()
Conmat.report()


### Random Forest Separating Digits and Lower Cased Letters
#

# In[26]:


Symbols =     "".join(map(str, range(10))) + "abcdefghijklmnopqrstuvwxyz"
classes = [SymbolsToLabels(II) for II in Symbols]
TrainX, TrainY = SplitbyClasses(classes=classes, classSize=2000)
TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=
    True, test_set=True)

DimRe = DimReduceHybrid(X=TrainX, y=TrainY, pca_components=0.9)  # Use Train
    set to create LDA embeddings.
TrainEmbeddings = DimRe.getEmbeddings()  # Get Embeddings from the set trained
     LDA
TestEmbeddings = DimRe.getEmbeddings(TestX)  # Get the embeddings from the
    test set.

Model = RandomForestClassifier(n_estimators=200, n_jobs=-1, min_samples_leaf
    =2)
Model.fit(TrainEmbeddings, TrainY)


# In[25]:


matplotlib.rcParams['figure.figsize'] = (30, 30)
PredictedTestLabels = Model.predict(TestEmbeddings)
Conmat = ConfusionMatrix(TestY, PredictedTestLabels)
Conmat.visualize()
Conmat.report()


### Separating by Large Sub classes With LDA + PCA using Rand Forest
#
# 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.

# In[4]:


TrainX, TrainY, _, _, AxTicks = SplitByFunc(THREE_CLS_AUX_FUN, classSize
    =10000)
TestX, TestY, _, _, _ = SplitByFunc(THREE_CLS_AUX_FUN, classSize=3000,test_set
    =True)

DimRe = DimReduceHybrid(X=TrainX, y=TrainY)  # Use Train set to create LDA
    embeddings.
```

```
TrainEmbeddings = DimRe.getEmbeddings()  # Get Embeddings from the set trained LDA
TestEmbeddings = DimRe.getEmbeddings(TestX)  # Get the embeddings from the test set.

# Model = RandomForestClassifier(n_estimators=100, n_jobs=-1, max_leaf_nodes=)
Model = RandomForestClassifier(n_estimators=200,
                               n_jobs=-1,
                               verbose=1,
                               min_samples_leaf=0.01,
                               class_weight={0:3, 1: 1, 2: 1})
Model.fit(TrainEmbeddings, TrainY)
TestLabelsPredicted = Model.predict(TestEmbeddings)
Conmat = ConfusionMatrix(TestY, TestLabelsPredicted, axisTicks=AxTicks)
matplotlib.rcParams['figure.figsize'] = (8, 8)  # Graphics settings
Conmat.visualize()
show()
Conmat.report()
show()
Model.fit(TestEmbeddings, TestY)

TrainlabelsPredicted = Model.predict(TrainEmbeddings)
Conmat = ConfusionMatrix(TrainY, TrainlabelsPredicted)

Conmat.visualize()
show()
Conmat.report()
show()
Model.fit(TestEmbeddings, TestY)


### Binary Classification Between Digits and Data
#

# In[11]:


TrainX, TrainY, _, _, AxTicks = SplitByFunc(TWO_CLS_AUX_FUN, classSize=10000)
TestX, TestY, _, _, _ = SplitByFunc(TWO_CLS_AUX_FUN, classSize=3000,test_set=True)

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

# Model = RandomForestClassifier(n_estimators=100, n_jobs=-1, max_leaf_nodes=)
Model = RandomForestClassifier(n_estimators=200,
                               n_jobs=-1,
                               verbose=1,
                               min_samples_leaf=0.001,
                               class_weight={0: 1, 1: 1})
```

```
Model.fit(TrainEmbeddings, TrainY)
TestLabelsPredicted = Model.predict(TestEmbeddings)
Conmat = ConfusionMatrix(TestY, TestLabelsPredicted, axisTicks=AxTicks)
matplotlib.rcParams['figure.figsize'] = (8, 8)  # Graphics settings
Conmat.visualize()
show()
Conmat.report()
show()

TrainlabelsPredicted = Model.predict(TrainEmbeddings)
Conmat = ConfusionMatrix(TrainY, TrainlabelsPredicted, axisTicks=AxTicks)

Conmat.visualize()
show()
Conmat.report()
show()


#

### Random Forest on the Digits
#

# In[15]:


Symbols = "".join(map(str, range(10)))
classes = [SymbolsToLabels(II) for II in Symbols]
TrainX, TrainY = SplitbyClasses(classes=classes, classSize=float("inf"))
TestX, TestY = SplitbyClasses(classes=classes, classSize=1000, shuffle_data=
    True, test_set=True)

DimRe = DimReduceHybrid(X=TrainX, y=TrainY, pca_components=0.9)  # Use Train
    set to create LDA embeddings.
TrainEmbeddings = DimRe.getEmbeddings()  # Get Embeddings from the set trained
     LDA
TestEmbeddings = DimRe.getEmbeddings(TestX)  # Get the embeddings from the
    test set.

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


# In[14]:


matplotlib.rcParams['figure.figsize'] = (30, 30)
PredictedTestLabels = Model.predict(TestEmbeddings)
Conmat = ConfusionMatrix(TestY, PredictedTestLabels)
Conmat.visualize()
Conmat.report()


### 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.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 #
```

```
#
# Medium Article:
# https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8
# An introduction to decision tree, Information gain, entropy etc.
#
#
# Medium Article:
# https://towardsdatascience.com/an-illustrative-introduction-to-fishers-linear-discriminant-9484efee15ac
# Putting Fisher's Discriminant Analysis into an optimization problem.
#
#

# #### Theoretical Part References
# 1. Dim Reduce with Fisher's LDA
#     * Fisher's Formation of the LDA as an optimization problem.
#         * https://towardsdatascience.com/an-illustrative-introduction-to-fishers-linear-discriminant-9484efee15ac
#     * Elements of Statistical Learning, Page 108
# 2. PCA Dim Reduce
#     * Eckar Young Theorem:
#         * Page 7 of Data science for Engineering.
#     * Alignment Problem, and Unitary Transformation:
#         * Page 35
# 3. PCA + LDA Dim reduce:
#     * cite : Transform Function from the sklearn library.
# 4. SVM:
#     * Elements of statistical learning: page 408
#     * SK learn for SVM complexity
# 5. Random Forest:
#     * Page 288, Elements of Statistical Learning.
#     * Weakness of decision tree. https://scikit-learn.org/stable/modules/tree.html#tree
#     * Information gain and Entropy of decision tree: https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8
#
```

A.4 Codes for CNN training

```
# -*- coding: utf-8 -*-
"""Custom-Model-digits.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1woOc1xd3lykGKKJkfAmMaG95o8-xutUN
"""

!git clone https://github.com/sorki/python-mnist
!./python-mnist/bin/mnist_get_data.sh
!pip3 install emnist keras-tuner
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,
            activation='relu', kernel_initializer='he_uniform', input_shape=(28,
            28, 1)))
69      model.add(MaxPooling2D((2, 2)))
70      model.add(Dropout(rate=hp_dropout_rate))
71      model.add(Conv2D(filters=hp_filters*2, kernel_size=hp_kernel_size,
            activation='relu', kernel_initializer='he_uniform'))
72      model.add(MaxPooling2D((2, 2)))
73      model.add(Dropout(rate=hp_dropout_rate))
74      model.add(Flatten())
75      hp_units = hp.Choice('units', values=[64, 128, 256, 512])
76      model.add(Dense(units=hp_units, activation='relu', kernel_initializer='
            he_uniform'))
77      model.add(Dropout(rate=hp_dropout_rate))
78      model.add(Dense(10, activation='softmax'))
79
80
81      # Tune the learning rate for the optimizer
82      # Choose an optimal value from 0.01, 0.001, or 0.0001
83      hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
84
85      model.compile(optimizer=keras.optimizers.Adam(learning_rate=
            hp_learning_rate),
86                    loss=keras.losses.CategoricalCrossentropy(from_logits=True),
87                    metrics=['accuracy'])
88
89      return model
90
91  tuner = kerastuner.Hyperband(model_builder,
92                          objective='val_accuracy',
93                          max_epochs=10,
94                          factor=3,
95                          directory='my_dir',
96                          project_name='intro_to_kt')
97  stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
98  tuner.search(train_X, train_y, epochs=50, validation_split=0.2, callbacks=[
        stop_early])
99
100 # Get the optimal hyperparameters
101 best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]
102
103 # Build the model with the optimal hyperparameters and train it on the data
        for 50 epochs
104 checkpoint_filepath = './checkpoint'
105 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
106     filepath=checkpoint_filepath,
107     save_weights_only=False,
108     monitor='val_accuracy',
109     mode='max',
110     save_best_only=True)
111
112 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

```
113
114 cb_list = [model_checkpoint_callback, early_stop]
115 model = tuner.hypermodel.build(best_hps)
116 history = model.fit(train_X, train_y, epochs=50, validation_split=0.2,
        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
        for 50 epochs
154 checkpoint_filepath = "custom-model-digits-best.h5"
155 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
156     filepath=checkpoint_filepath,
157     save_weights_only=False,
158     monitor='val_accuracy',
159     mode='max',
160     save_best_only=True)
161
162 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

```
163
164 cb_list = [model_checkpoint_callback, early_stop]
165 model = tuner.hypermodel.build(best_hps)
166 history = model.fit(train_X, train_y, epochs=50, validation_split=0.2,
        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)
```

```
1 # -*- coding: utf-8 -*-
2 """Custom_Model_letters.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/179ySpL_OSBjpHiLgD1UT3ieTqAmEmx_
8 """
9
10 !git clone https://github.com/sorki/python-mnist
11 !./python-mnist/bin/mnist_get_data.sh
```

```
!pip3 install emnist keras-tuner
from emnist import extract_training_samples

print("Imported the EMNIST libraries we need!")

import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.layers import *
from keras.models import *
from keras.optimizers import *
from keras.utils import to_categorical
from emnist import extract_training_samples, extract_test_samples


from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

import kerastuner

# Note: If you are using the Google Colab, test the GPU source

gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Select the Runtime > "Change runtime type" menu to enable a GPU
      accelerator, ')
  print('and then re-execute this cell.')
else:
  print(gpu_info)

train_X, train_y = extract_training_samples("letters");
test_X, test_y = extract_test_samples("letters");

# Normalise
train_X = train_X.astype('float32')
train_X /= 255
test_X = test_X.astype('float32')
test_X /= 255

#train_X, val_X, train_y, val_y = train_test_split(images, labels, test_size=
    0.10, random_state=42)

# Reshape image for CNN
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

num_classes = np.unique(train_y)
train_y = to_categorical(train_y)
test_y = to_categorical(test_y)

```

```
train_y = np.delete(train_y, 0, axis=1)
test_y = np.delete(test_y, 0, axis=1)

def model_builder(hp):
    model = keras.Sequential()
    hp_filters = hp.Choice('filters', values=[32, 64, 128])
    hp_kernel_size = hp.Choice('kernel_size', values=[3, 5])
    hp_dropout_rate = hp.Float('rate', min_value=0, max_value=0.5, step=0.1)
    model.add(Conv2D(filters=hp_filters, kernel_size=hp_kernel_size,
        activation='relu', kernel_initializer='he_uniform', input_shape=(28,
        28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(rate=hp_dropout_rate))
    model.add(Conv2D(filters=hp_filters*2, kernel_size=hp_kernel_size,
        activation='relu', kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(rate=hp_dropout_rate))
    model.add(Flatten())
    hp_units = hp.Choice('units', values=[64, 128, 256, 512, 1024])
    model.add(Dense(units=hp_units, activation='relu', kernel_initializer='
        he_uniform'))
    model.add(Dropout(rate=hp_dropout_rate))
    model.add(Dense(26, activation='softmax'))


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

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

    return model

tuner = kerastuner.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='my_dir',
                     project_name='hp-custom-model-letters')
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
tuner.search(train_X, train_y, epochs=50, validation_split=0.2, callbacks=[
    stop_early])

# Get the optimal hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

# Build the model with the optimal hyperparameters and train it on the data
    for 50 epochs
checkpoint_filepath = './checkpoint'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    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,
        callbacks=cb_list)
121 
122 val_acc_per_epoch = history.history['val_accuracy']
123 best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
124 print('Best epoch: %d' % (best_epoch,))
125 
126 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')}")
127 
128 hypermodel = tuner.hypermodel.build(best_hps)
129 
130 # Retrain the model
131 hypermodel.fit(train_X, train_y, epochs=best_epoch, validation_split=0.2)
132 
133 eval_result = hypermodel.evaluate(test_X, test_y)
134 print("[test loss, test accuracy]:", eval_result)
135 
136 # Retrieve the best model.
137 best_model = tuner.get_best_models(num_models=1)[0]
138 
139 # Evaluate the best model.
140 eval_result_best = best_model.evaluate(test_X, test_y)
141 print("[test loss, test accuracy]:", eval_result_best)
142 
143 hypermodel.save('custom_model_digits.h5')
144 
145 history.history['val_accuracy']
146 
147 y_pred = hypermodel.predict(test_X)
148 y_label = (y_pred > 0.5)
149 
150 import sklearn.metrics as metrics
151 cm = metrics.confusion_matrix(test_y.argmax(axis=1), y_label.argmax(axis=1))
152 
153 cm
154 
155 best_model.summary()
156 
157 # Build the model with the optimal hyperparameters and train it on the data
        for 50 epochs
158 checkpoint_filepath = "custom-model-letters-best.h5"
159 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
160     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,
        callbacks=cb_list)
171 
172 # val_acc_per_epoch = history.history['val_accuracy']
173 # best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
174 # print('Best epoch: %d' % (best_epoch,))
175 
176 # plot loss during training
177 plt.subplot(211)
178 plt.title('Loss')
179 plt.plot(history.history['loss'], label='train')
180 plt.plot(history.history['val_loss'], label='val')
181 plt.legend()
182 # plot accuracy during training
183 plt.subplot(212)
184 plt.title('Accuracy')
185 plt.plot(history.history['accuracy'], label='train')
186 plt.plot(history.history['val_accuracy'], label='val')
187 plt.legend()
188 
189 plt.tight_layout()
190 plt.show()
191 
192 del model
193 
194 from keras.models import load_model
195 
196 # returns a compiled model
197 # identical to the previous one
198 model = load_model('custom-model-letters-best.h5')
199 
200 type(history)
201 
202 import pandas
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)
```

```
1 #!/usr/bin/env python
2 # coding: utf-8
3 
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-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
42 model.summary()
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
```

```
_, test_y_label = extract_test_samples("digits");


# In[7]:


Conmat = ConfusionMatrix(test_y_label, y_pred_label)
Conmat.visualize()
Conmat.report()


# In[5]:


df=pd.read_csv('./CNN_Models/history.csv')


# In[6]:


loss = df['loss'].to_numpy()
val_loss = df['val_loss'].to_numpy()
accuracy = df['accuracy'].to_numpy()
val_accuracy = df['val_accuracy'].to_numpy()


# In[77]:


# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(loss, label='train')
plt.plot(val_loss, label='val')
plt.xlabel('epoch')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(accuracy, label='train')
plt.plot(val_accuracy, label='val')
plt.xlabel('epoch')
plt.legend()

plt.tight_layout()
plt.savefig('CNN-digits-best.pdf')
plt.show()


# In[ ]:
```

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

```

```
# In[1]:


from emnist import extract_training_samples, extract_test_samples
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.layers import *
from keras.models import *
from keras.optimizers import *
from keras.utils import to_categorical
import kerastuner as kt
import pandas as pd
from keras.models import load_model

from core import *

# returns a compiled model
# identical to the previous one
model = load_model('./CNN_Models/custom-model-letters-best.h5')


# In[2]:


test_X, test_y = extract_test_samples("letters");
test_X = test_X.astype('float32')
test_X /= 255
test_y = to_categorical(test_y)
test_X = test_X.reshape(-1, 28, 28, 1)
test_y = np.delete(test_y, 0, axis=1)


# In[ ]:


model.summary()


# In[3]:


result = model.evaluate(test_X, test_y)
print("[test loss, test accuracy]:", result)


# In[ ]:


y_pred = model.predict(test_X)
y_pred_label = np.argmax(y_pred,axis=1)
```

```
# y_pred_label = np.where(y_pred_label==1)[1]

_, test_y_label = extract_test_samples("letters")


# In[5]:


df=pd.read_csv('./CNN_Models/history-custom-model-letters-best.csv')


# In[6]:


loss = df['loss'].to_numpy()
val_loss = df['val_loss'].to_numpy()
accuracy = df['accuracy'].to_numpy()
val_accuracy = df['val_accuracy'].to_numpy()


# In[7]:


# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(loss, label='train')
plt.plot(val_loss, label='val')
plt.xlabel('epoch')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(accuracy, label='train')
plt.plot(val_accuracy, label='val')
plt.xlabel('epoch')
plt.legend()

plt.tight_layout()
plt.savefig('CNN-letters-best.pdf')
plt.show()


# In[ ]:
```