Hand-written Alphanumeric Character Recognition

Dylan Coakley

November 27th, 2017

1 Introduction

One of the most fundamental problems in computer vision is that of hand-written character recognition. There has been a lot of research done in the field of character recognition over past decades, and it has seen remarkable success. There is one particular machine learning researcher who has contributed a lot of work, specifically to the character recognition problem itself. In his preliminary papers, such as (4), Yann LeCun compared classic learning techniques to his newly conceived Convolutional Neural Network *LeNet* to show how having an appreciation for spatial relationships in data can increase the accuracy of a machine.

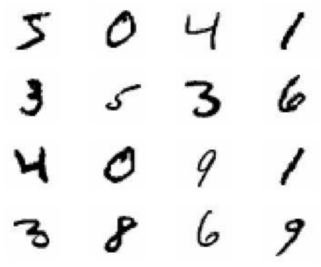
The need for accurate character classifiers has become even more relevant in recent years, with the prevalence of smart-phones, the digitization of files, and the pursuit of automating tasks that were previously done by hand. It is for these reasons and others that I have chosen it for as topic of interest.

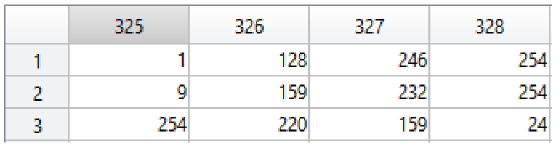
This project concerns the study of creating machines capable of classifying hand-written alphanumeric characters. Multiple supervised learning techniques will be utilized to solve the problem. The learning techniques used were the following: Principal Components Analysis, k-Nearest Neighbours, Support Vector Machine, Discriminant Analysis, Random Forest, and the Convolutional Neural Network. The performance of the models produced by these techniques will be analyzed and compared against each other in order to determine which among them performs objectively better than the others.

The data set that I have found to be the best assembled, and have worked with for the purpose of this project, is called the EMNIST data set. It is an extension of the more well-known NIST data set, originally collected by the National Institute of Standards and Technology in 1995. Subsequently, the samples of individual digits were extracted from the NIST data set and were reformatted to create the Modified NIST (MNIST) data set. In the previous year, the samples of letters were extracted from NIST and were reformatted aswell to create the Extended MNIST (EMNIST) data set. The structure of the EMNIST data set is well-suited for the MATLAB environment.

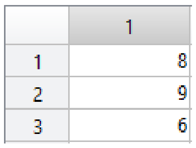
The EMNIST data set can be found at the link in (1). The data set is quite large, as there are 814,255 alphanumeric character samples in total. The authors of the data set also provide various subsets of the total data that can be used to analyze as well, with each of these subsets having proper training and testing sets. For example, EMNIST Digits, EMNIST Letters, EMNIST By Class, and EMNIST By Merge. The authors of the data set applied supervised learning techniques to their own data set, including the subsets, in order to set a benchmark for the accuracies that one may expect to achieve. The documentation given in (2) gives a lot of useful information about the data set.

Each data sample is a 28×28 grayscale image, which is converted into a 1-dimensional array of 784 pixels. Each pixel has a grayscale value in the range of $[0, 255]$. Each data sample also has a corresponding class label. Figure 1 shows the original hand-written form of some digits and example data samples from the set.





	325	326	327	328
1	1	128	246	254
2	9	159	232	254
3	254	220	159	24



	1
1	8
2	9
3	6

Figure 1: Original unmodified hand-written digits, followed by some pixel values and the label of 3 data samples.

For the purpose of this project, the focus was placed on 3 specific subsets of the data set which are described below:

1. EMNIST Digits

   This subset contains the samples of all digits from 0 to 9, and thereby has 10 classes. There are 240,000 samples included in the training set, and 40,000 in the testing set. For the sake of reducing complexity and saving time, I've reduced the sizes of the training and testing sets. Any analysis of EMNIST Digits will use 10,000 samples for the training set, with each class having 1000 samples, and 5000 samples for the testing set, with each class having 500 samples.

2. EMNIST Letters

   This subset contains samples of all of the letters from A to Z. The lowercase and uppercase version of each letter are grouped together in the same label, therefore there are 26 classes. There are 88,800 samples included in the training set, and 14,800 in the testing set. Any analysis of EMNIST Letters will use a reduced 26,000 samples for the training set, with each class having 1000 samples, and 6500 samples for the testing set, with each class having 250 samples.

3. EMNIST Balanced

   This subset contains samples of all digits from 0 to 9 and all letters from A to Z. The lowercase and uppercase version of each letter are grouped together only if one is the scaled version of the other, such as C and c. There are a total of 47 classes, 112,800 samples included in the training set, and 18,800 in the testing set. Any analysis of EMNIST Balanced will use a reduced 47,000 samples for the training set, with each class having 1000 samples, and 9400 samples for the testing set, with each class having 200 samples.

As there are a large number of samples in the training set for each instance there is no worry about a violation of Raudys and Jain's curse of dimensionality, as discussed in (3).

2 Methods

This section introduces the methods applied to the character recognition classification task. The machine learning techniques that were used to solve the character recognition problem were:

1. Principal Components Analysis

   Since the data set is composed of 28×28 images, each sample has 784 individual features. This is a large number of features, so it follows that a decent number of features of each sample contribute little to no information about it. This is especially true as each feature represents a single pixel. The features representing pixels on the outer edges of the image are likely to provide no information at all. By performing a PCA on the data set, an optimal number of principal components can be obtained. These principal components capture the majority of the variance in the data, while reducing the dimensionality of the problem at the same time. The number of principal components found to be optimal will be carried on and used in the majority of the following supervised learning techniques below.

2. k-Nearest Neighbour

   One of the simplest supervised learning techniques. It performs classification by assigning the majority class to a test sample based on its k nearest neighbours. The parameter k, the number of samples to compare the test sample to, is set by the user. The closeness of samples is based on some distance metric such as Euclidean. The k-NN technique will be used to classify the data, while consecutively testing appropriate values for the parameter k.

3. Support Vector Machine

   The goal of the SVM is to construct a decision boundary that maximizes inter-class distance. This decision boundary is defined by a subset of the samples known as support vectors. Similarity measures between samples is found by using a kernel function. SVM models using more complex kernel functions, such as polynomial, or the radial basis function are quite sensitive to changes in their parameters. Some examples are the coefficient γ, the regularization term C, and the degree d. Therefore, these parameters must be analyzed and set to appropriate values in the model.

4. Discriminant Analysis

   Linear and quadratic discriminant analysis are appealing classifiers in a few ways. They operate by assuming that every class follows a Gaussian distribution, which makes their training time short, as they require fewer computations. They're inherently multi-class which is essential for the character recognition problem. They also have no parameters to fine-tune, making them simple to work with.

5. Random Forest

   While a single decision tree usually obtains quite poor performance on its own, a collection of decision trees can actually obtain respectable results. This is the goal of the Random Forest learning technique. It uses a process known as bootstrap aggregation, or bagging as it is most commonly known. The idea behind bagging is to increase the classification accuracy by training multiple instances of classifiers, decision trees in this instance, on different permutations of the features.

6. Covolutional Neural Network

Unlike the previously considered learning techniques, the convolutional neural network is the only one to appreciate the spatial relationships between sample features. This is what gives it an advantage while performing image processing, and makes it especially applicable to the character recognition problem. These machines were developed specifically to perform digit recognition by Yann LeCun in (4).

3 Results

This section presents the results obtained for the various classification tasks based on the EMNIST Digits, Letters, and Balancd datasets. All data manipulation and processing was done in the MATLAB environment.

3.1 Principal Components Analysis

The amount of variance captured by a set of principal components can be computed with the following equation:

$$VarCap(m) = \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{n} \lambda_i},\ where\ m \leq n$$

where λ_i represents the i^{th} eigenvalue, and n is the total number of features in the data set.

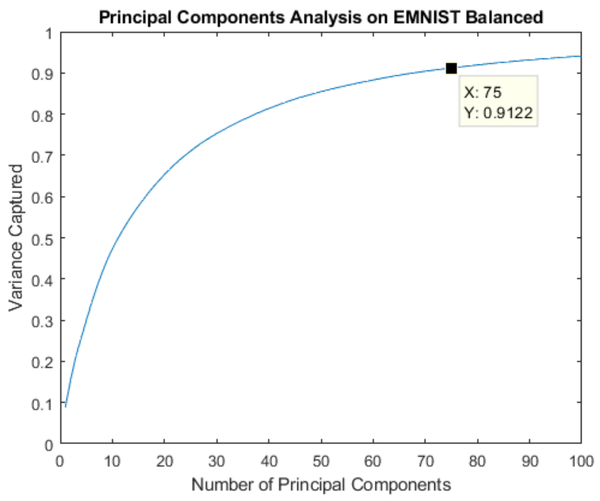


Figure 2: Plot of the amount of variance captured per principal component included.

The plot shown in Figure 2 shows how much variance is captured for different numbers of principal components kept for the EMINST Balanced data set. The majority of the variance, approximately 90%, is captured when keeping around 75 principal components. The plot also starts to plateau at 75, as after this point, there proceeds to be very miniscule increases in the amount of variance captured.

The result of PCA performed on the EMNIST Digits and Letters data sets are very similar to the results for the EMINST Balanced data set. Therefore, in any analysis of the samples using a classifier, save for a convolutional neural network, each sample had its number of features reduced from 784 to the 75 most informative features according to PCA.

3.2 k-Nearest Neighbour

The k-Nearest Neighbour algorithm was put up to the classification task. The distance measure used was standard Euclidean distance. Even values of k had ties broken by assigning the class with closer samples, making them equivalent to the odd value of k one less than them. Therefore, only odd values of k were considered. Figure 3 shows the performance of differing k-NN models on the different data sets.

k-NN Model Accuracy							
EMNIST Dataset	$k = 1$	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 11$	$k = 13$
Digits	95.42%	95.34%	95.32%	95.16%	94.80%	94.48%	94.50%
Letters	82.89%	83.22%	82.78%	82.71%	82.15%	81.92%	81.32%
Balanced	76.04%	76.47%	77.18%	77.01%	76.67%	76.22%	76.18%

Figure 3: Classification accuracies for k-NN models with different values of k

It appears as though lower values of k produce the highest accuracies while classifying alphanumeric characters, as we can see that $k = 1$, $k = 3$, and $k = 5$ minimize the error of classifying EMNIST Digits, Letters, and Balanced, respectively.

3.3 Support Vector Machine

The Support Vector Machine (SVM) was used to solve the recognition problem. The default SVM classifier implemented in the default MATLAB environment can only perform binary classification on a given data set. However, character recognition is a multi-class learning problem by nature. Therefore, MATLAB's `fitcecoc` function was used to resolve this issue.

The `fitcecoc` function takes as a parameter the template of the type of SVM to use. It constructs $\frac{n(n-1)}{2}$ SVM models, where n is the number of unique classes. It proceeds by employing the one-against-one approach, thereby reducing the multi-class problem into a series of binary-class problems from which it can learn. The parameters of SVM, namely γ, were optimized automatically by the `fitecoc` function. Figure 4 shows the results obtained from SVMs using various kernel functions.

SVM Model Accuracy						
EMNIST Dataset	Linear	Poly-2	Poly-3	Poly-4	Poly-5	RBF
Digits	93.28%	95.96%	96.30%	96.32%	96.12%	95.72%
Letters	76.35%	84.69%	85.18%	84.24%	83.97%	83.45%
Balanced	72.72%	79.37%	79.75%	79.32%	79.08%	78.82%

Figure 4: Classification accuracies for SVM models with various kernel functions.

It seems as though a SVM with a polynomial kernel function produces the highest accuracy while classifying alphanumeric characters, as we can see that the kernel Poly-4 minimizes the error of classifying EMNIST Digits, and the kernel Poly-3 minimizes the classification error of EMNIST Letters and Balanced.

3.4 Discriminant Analysis

Discriminant Analysis (DA) was employed to classify the characters. The `DiscrimType` parameter given to MATLAB's `fitcdiscr` function for Linear DA was `'pseudolinear'` and `'pseudoquadratic'` was given to Quadratic DA. All this means is that all classes had the same covariance matrix. MATLAB inverted the covariance matrix using the pseudo-inverse operation. Figure 5 shows the performance of these classifiers on the data sets.

Discriminant Analysis Model Accuracy		
EMNIST Dataset	Linear	Quadratic
Digits	87.88%	95.38%
Letters	60.32%	83.05%
Balanced	57.76%	78.69%

Figure 5: Classification accuracies for discriminant analysis models.

The linear discriminant models performed quite poorly when compared to their quadratic counterparts, which implies that the characters were not easily linearly separable.

3.5 Random Forest

The Random Forest (RF) learning technique was put to use to solve the classification task. After testing various numbers for the amount of decision trees to use, the best results hovered around 100. Therefore, all of the RF models used a bootstrap aggregation of 100 individual decision trees. Figure 6 gives the average classification accuracies of the models.

Random Forest Model Accuracy	
Digits	93.46%
Letters	80.26%
Balanced	75.11%

Figure 6: Average classification accuracies for Random Forest models.

3.6 Convolutional Neural Network

In order to construct a Convolutional Neural Network (CNN) model to classify characters, some data conversion had to be done. A CNN must take, as its input, 2-dimensional samples. However, all samples in the EMNIST data sets are in the form of a 1-dimensional array with 784 features. Therefore, each 1-dimensional sample was reformatted to 2-dimensional 28×28 grayscale image and saved. This was done using MATLAB's `reshape` and `mat2gray` functions.

The structure of the CNN used was not overly complex, it included two convolutional layers, and a single max pooling layer, among others. The exact architecture of the CNN used is shown in Figure 8.

For the training phase of the CNN model, the number of epochs, which refers to the number of times all of the training samples are used once to update the weights, was set to a maximum of 15. This value appeared to be in the appropriate range that was large enough not to underfit and small enough not to overfit the testing set. This is apparent from the plot in Figure 9 as the batch accuracy decreases during every epoch, and there is no straight convergence to 100% accuracy. The learning rate of the model was set to 0.0001 to ensure that it would not obtain unfavourable behaviour, albeit it took longer to train the model because of this choice. Figure 7 shows the accuracy of the CNN models.

CNN Model Accuracy	
Digits	96.68%
Letters	85.92%
Balanced	80.78%

Figure 7: Classification accuracies for Convolutional Neural Network models.

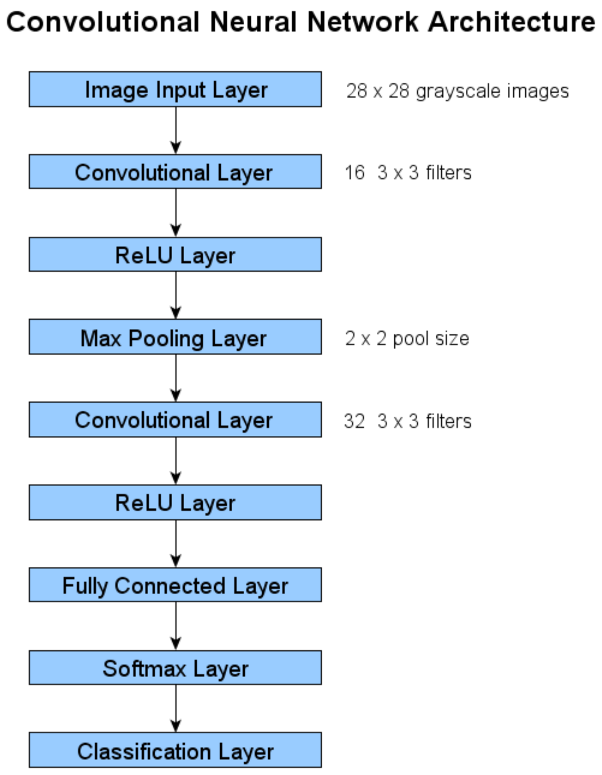


Figure 8: The structure of the layers of the convolutional neural network used for the classification task.

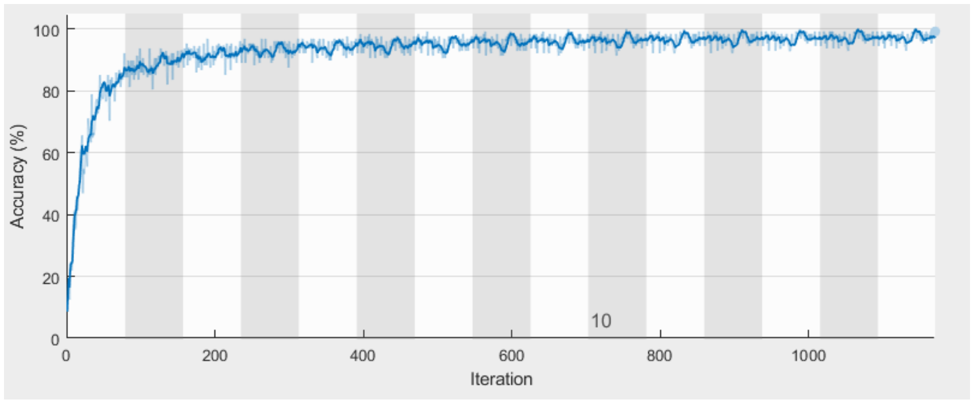


Figure 9: Plot of the accuracy of the CNN model during the training phase.

4 Discussion

This section discusses the significance of the results obtained, and look at some comparisons between techniques. The table in Figure 10 displays an overview of the performances of the supervised learning techniques used to classify the hand-written characters.

Supervised Learning Technique Accuracy					
EMNIST Dataset	*k*-NN	SVM	Discrim. Analysis	Random Forest	Conv. NN
Digits	95.42%	96.32%	95.38%	93.46%	96.68%
Letters	83.22%	85.18%	83.05%	80.26%	85.92%
Balanced	77.18%	79.75%	78.69%	75.11%	80.78%

Figure 10: Overview of the performance of the classifiers.

We can see that the convolutional neural (CNN) network models performed objectively better than the other supervised learning methods assessed in terms of classification accuracy. This kind of result was not totally surprising. The CNN models acknowledged the spatial relationships between pixels in each character sample. This is unique among all of the learning techniques that were examined. Since the data samples are representations of 2-dimensional images, this acknowledgement allowed it to see a better overlying view of the structure of the data, which gave it a certain advantage and a boost in its performance.

The Support Vector Machine (SVM) models also had a great performance, as they only trailed behind the CNN models by approximately 0.5% for each data set. This is even more impressive when you take into consideration that the SVM models based their learning on samples that only used 75 features, compared to the entire 784 features used by the CNN models.

The *k*-Nearest Neighbour and Discriminant Analysis models obtained very similar results, both of which were respectable, considering the simplistic nature of the two techniques. The Random Forest models performed the worse out of all techniques. This may be an implication that ensembles of decision trees are not the most appropriate classifiers for image data, since they split on single pixels. Characters can easily be translated or scaled, making it hard to determine what grayscale value should be the threshold for each split.

Now reflecting on the methods used, the performance of the supervised learning techniques, except CNNs, may have been improved if a different number of principal components were used. Possibly even finding the best number of principal components to use for each individual classifier would have been a better way of analyzing the techniques.

The authors of the EMNIST dataset used an OPIUM-based neural network to classify the samples of their dataset in order to set a benchmark for others. Figure 11 shows the performance of their classifier.

EMNIST Author's Technique Accuracy (2)	
EMNIST Dataset	OPIUM-based NN
Digits	97.22%
Letters	85.15%
Balanced	78.02%

Figure 11: EMNIST Author's Classification Accuracy

I acheived better classification accuracy than the authors on the EMNIST Letters and Balanced data sets. However, while they trained on the entirety of EMNIST Digits, Letters, and Balanced, I only trained on a subset of each data set. This may imply that my models may have been slightly overfitted to the training set.

References

[1] EMNIST Data Set:
`https://nist.gov/itl/iad/image-group/emnist-dataset`

[2] Cohen, G., Afshar, S., Tapson, J., van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from
`http://arxiv.org/abs/1702.05373`

[3] Raudys, Sarunas J., and Anil K. Jain. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. IEEE Transactions on pattern analysis and machine intelligence 13.3 (1991): 252-264.

[4] LeCun, Yann, et al. (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. Retrieved from
`http://yann.lecun.com/exdb/publis/pdf/lecun-95a.pdf`

[5] LeCun, Yann, et al. (1990). Handwritten character recognition using neural network architectures. Retrieved from
`http://yann.lecun.com/exdb/publis/pdf/matan-90.pdf`

[6] Norhidayu Abdul Hamid and Nilam Nur Amir Sjarif (2017). Handwritten Recognition Using SVM, KNN and Neural Network. Retrieved from
`http://arxiv.org/abs/1702.00723`