Assignment 2 - Group: 43

Alejandro Diaz (adia2600)
SID: 500229318
University of Sydney

Nicholas Stollmann (nsto8788)
SID: 312062796
University of Sydney

Shashwati Dutta (sdut7674)
SID: 500693287
University of Sydney

Tutors: *Yixuam Zhang, Chen Chen, Kunze Wang*

Abstract

We attempt to find the best machine learning model to use for image classification tasks. By creating three different algorithms the goal of this paper is to find the most suited model to use for the ever-growing field of image recognition, with its applications covering many different sectors. We will not only find the best model to fit this dataset but will also attempt to predict which models have the most potential to improve and which models have the flexibility to be used for different datasets.

The results are based on the Chars74k dataset but the models should be transferable to any dataset of black and white letters and numbers. We tested many different types of models but will show the results of only the top three, all of which have accuracies comfortably over 80%.The findings showed that a random forest model and a convolutional neural network are both suited to deal with such problems. The K-nearest neighbour also gave reasonable results, however it did have some limitations and did not prove to be as accurate as the other two. Whilst the results of the top two algorithms suggest they are both equally effective, we believe that the convolution neural network has the most potential for improvement and that it is the most suited to be scaled and adjusted for differing datasets with similar problems.

Keywords: Principal Component Analysis (PCA), Hyper-parameter, K-Nearest Neighbour, Random Forest, Convolutional Neural Network (CNN), Cross-Validation

1. Introduction

The aim of this report is to find the best method to undertake character recognition. The dataset we are analysing is made up of 62992 synthesised characters from computer fonts (1). The data is split into 62 classes of handwritten images made up of all letters in lower and upper case, as well as the ten digits. The importance of accurate character recognition is at an all-time high with all kinds of paperwork being transitioned online along with many other applications (2). Optical character recognition (OCR) is a method of converting handwritten text to computer text (3) and that is exactly the type of problem our algorithms could be used to solve. Using character recognition for certain documents (e.g. reference books, court cases etc.) may have great importance, therefore having high accuracy is crucial, especially when the text may include thousands of words. The importance of this algorithm also increases the need for a large dataset with which to train the model, as to make it more applicable to real world problems. Furthermore, this type of model could be used to transfer old handwritten documents or books to a computer where they can be kept more safely and can be analysed much more easily.

The approach we will take will be to train a Random Forest classifier, a K-Nearest Neighbour classifier and a Convolutional Neural Network. The above were the chosen models, since they are mostly non-parametric and do not make assumptions about the type of data. Convolutional neural networks are also some of the most common models used in all types of image recognition and character recognition. After applying pre-processing and briefly testing several other models, our belief was reinforced that these would be the best models to apply to this problem.

2. Previous Work

Chars74 datasets are very popular datasets when it comes to image character recognition. While these datasets have been widely used for Optical Character Recognition (OCR), ML researchers also provided multiple machine learning algorithms that performed fairly well, for instance (de Campos et al. 2009)(4) have used nearest neighbors and SVM techniques and outperformed commercial OCR system on the dataset with segmented characters from natural scenes. They were able to achieve 0.55 accuracy when trained using 15 samples per class. For character recognition one of the key machine learning technique is Convolutional Neural Network (CNN) and we also see many related work using CNN, (Alexander Viklund et al. 2017)(5) have constructed CNN that is trained and tested on the Chars74K dataset, with 15 images per class for training and 15 images per class for testing. They were also able to reach 0.55 accuracy when classifying the test images. (Vishnu Sundaresan et al. 2015) (6) proposed different classification techniques such as KNN (with accuracy 0.35), Linear Classifier (with accuracy 0.30), LeNet (with accuracy 0.45), AlexNet (with accuracy 0.63) and the victor CNN (with accuracy 0.71). They have also used the dataset with segmented characters from natural scenes.

As most of the previous work focused on the characters from natural scenes, in this project, we propose three classification methods, K- Nearest Neighbor, Random Forest and Convolutional Neural Network on characters from computer fonts with 4 variations (combinations of italic, bold and normal). We aim to perform Principal component Analysis as a pre-processing step and train all the proposed models with 60 percent of the data, tune hyper-parameters using 20 percent validation data and finally test model performance(accuracy) on remaining 20 percent.

3. Methods

In this section, we provide formal formulations for the different implemented classifiers as well as the preprocessing techniques we have used.

3.1 Preprocessing

An essential part of every Machine Learning problem is to understand what kind of data we are dealing with and what preprocessing is required. Hence, in this section we present some preprocessing techniques we have applied to prepare our data for training.

3.1.1 Balanced or Imbalanced Dataset

It is globally known that an imbalanced dataset can negatively affect the performance of a model due to the classes having a different number of observations. In case we are dealing with an imbalanced dataset, we would need to apply some techniques, such as over-sampling or under-sampling to rebalance the data.

As we can observe in the Figure 1, the selected dataset is well balanced as the labels contain the same number of samples.

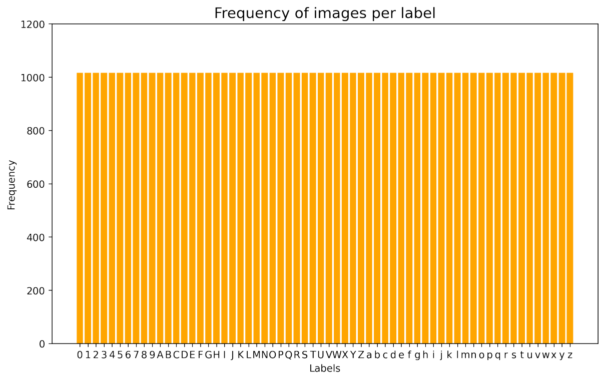

Figure 1: Frequency of images per label in the original data.

3.1.2 Train test split

To avoid data leakage, which refers to the sharing of information between the test and training sets, we should perform the train-test split before standardising our data, in this case, we split the dataset into:

Set	Percentage
Training Set	0.60
Validation Set	0.20
Test Set	0.20

Table 1: Train-test split

3.1.3 Standardise Features

An important step to be able to correctly perform the dimensionality reduction and improve the performance of our model is to standardize our images. However, to correctly standardise our data, we should use the training set to calculate the mean and variance, normalize the training set and then normalize the validation and test set using the same mean and variance from training set.

$$\text{scaled train} = \frac{(\text{train} - \mu_{train})}{\sigma_{train}}$$

$$\text{scaled validation} = \frac{(\text{validation} - \mu_{train})}{\sigma_{train}}$$

$$\text{scaled test} = \frac{(\text{test} - \mu_{train})}{\sigma_{train}}$$

3.1.4 Dimensionality Reduction

Dimensionality Reduction is the transformation of high-dimensional data into another representation of reduced dimensionality. This allows us to simplify the problem and mitigate the curse of dimensionality and other undesired properties of high-dimensional space (7). Additionally, by reducing the data onto a lower dimensional space it facilitates its interpretation, something which is often difficult in case of large datasets.

In this case, we used one of the most common linear dimensionality reduction techniques applied nowadays, the Principal Component Analysis (PCA) (7). Given a data matrix $X \in \mathbb{R}^d$, our goal is to project X onto a k dimensional subspace ($k < d$) such that the variance of the project data is maximized. Through literature findings the two main definitions for PCA are:

- Minimum Error Formulation: Project the data onto a lower-dimensional space such that the variance of the projected data is maximized.
- Maximum Error Formulation: Project the data onto a lower-dimensional space such that the mean squared distance between data points and their projections is minimized.

The idea behind these definitions is simple, reduce the dimensionality of our data while preserving as much the variability as possible (8). Researchers try to avoid selecting the number of principal components (PCs) using subjective criteria. Many methods have been developed to solve this problem as well as to choose the number of PCs using an objective method (9).

In this case and for simplicity, we analyzed the percentage of information retention to select the number of principal components we should take. In the Figure 2 can be observed that the % of information retention depends on the number of principal components we select (10). Most of the information is retained by the first components. Furthermore, the

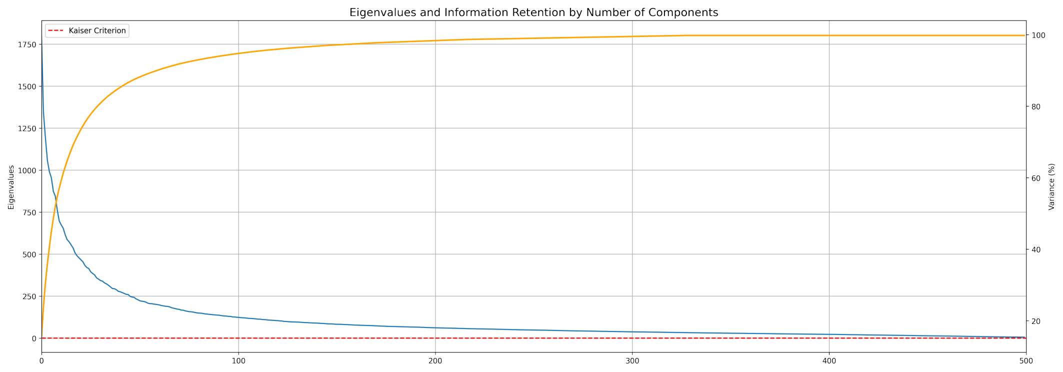

Figure 2: Blue: *Eigenvalues in Descending Order, where the red line indicates the Kaiser Criterion.* Orange: *Percentage of information retention by the number of components.*

figure shows that by keeping around 250 features we retained about 98% of the representation of the image data. Therefore, we projected our data onto a dimensional subspace with $k = 256$ and the projected data XV_{dxk}, where V_{dxk} represents the transformation matrix, was used as input for our classifier models.

We mentioned before that the dimensionality reduction techniques are used to reduce the dimensionality of our data while preserving as much of the variability as possible. Additionally, they provide a method to make large datasets more interpretable. This enables us to use our PCA algorithm to project our data X with dimensions $(62992, 784)$ onto a subspace with $k = 3$, which enables us to visualize in 3D our matrix (Figure 3).

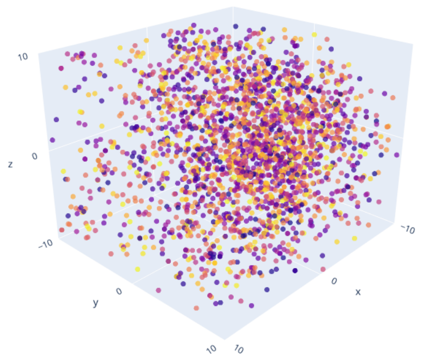

Figure 3: Representation of our data $X \in \mathbb{R}^{(62992x784)}$ projected onto a k = 3 dimensional subspace, where each color represents a class.

On the other hand, the Figure 4 contains the reconstruction of the letter t for several values of principal components (PCs). It seems that the quality of the reconstruction directly depends on the number of components selected. If we increase the number of PCs, the reconstructed image looks more like the original. Additionally, there is practically no difference between the image generated using $k = 256$ and the original image $k = 784$. We can conclude that the value of k we chose is equal to 256 produces a satisfactory result.

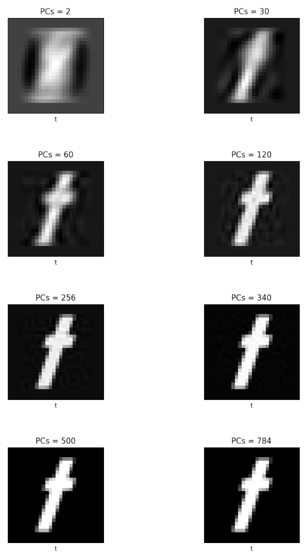

Figure 4: Data reconstruction for different values of principal components.

3.2 Random Forest

A random forest classifier is a non-parametric model that is made up of an ensemble of decision trees. This model makes no assumption about the type of data and having correlated features does not have much of a negative impact. The intuition behind a decision tree is very simple, it splits the data wherever the most information is gained, then it continues to split the data iteratively until it reaches a point where all classes are separated or until it reaches a given depth (11). A decision tree is highly relevant here as a random forest is an ensemble of decision trees; it will take the average of all the decision tree predictions within the forest. However, it is important that the trees making up the random forest are as uncorrelated as possible, so that it is not constantly creating identical trees that only use certain parts of the data or so the results are not dominated by only a small number of the features. The following illustration 5 shows exactly how a random forest classifier works (12).

The algorithm behind a random forest works in a way that will reduce correlation between trees and will maximise the accuracy of each tree. The way to avoid a correlated tree is by using a bootstrapped sample of the data to build each tree, that is, the training data will be resampled with replacement to build each individual tree (13). Whilst this is a good tool to reduce the similarity between trees, there is another trick used. At each split,

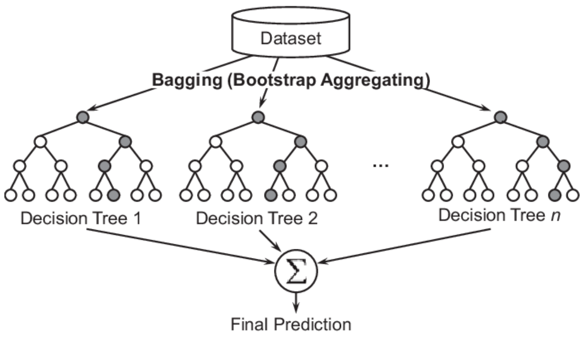

Figure 5: Illustration of a random forest

instead of picking the best feature to split on out of all features, only a subset of features is randomly made available to be split on. This will reduce the similarity between trees, as they will not split on the same feature at every iteration. The Gini criterion is also used when the algorithm is deciding how to split the data of each tree, and by adding all the values of the Gini importance index, we can calculate the importance of each variable (13).

3.3 K-Nearest Neighbours

K-Nearest Neighbour (KNN) is one of widely used supervised machine learning method. The popularity it gained is not only because of its performance but also, it's easy implementation. KNN is a non-parametric model which means it does not assume any underlying data distribution and this makes it advantageous with many real-world datasets as most of these data do not follow any mathematical theoretical assumption. Thus, this classifier is used in many fields. KNN does not build any predictive model, thus there is no learning phase.(14) To make predictions KNN uses the entire dataset which makes it a lazy learner.

KNN algorithm is very easy to implement. The only assumption KNN makes is that 'similar things are closer to each other'. (15) The prediction algorithm starts by choosing a value of 'k' followed by calculating distance of the previously unseen instance from every instance in the dataset. Different distance metrics are used in this step which include Euclidean distance, Hamming, or Manhattan. Euclidean distance is the most popular one. Once all the distances are calculated, the distance array was then was then sorted in ascending order and top k distances are chosen. To decide on the prediction class, a majority vote was then conducted. Finally, the previously unseen instance was assigned to a class that is the most appearing class of the first k observation in the sorted distance array. The figure below (figure 6) is an illustration of the algorithm with 2-dimensional data (16).

As we noticed, KNN algorithm is distance-based algorithm thus highly affected by scale of the variables. It is never advisable for an algorithm be biased towards variable with higher magnitude thus it is essential that the dataset is scaled before running the algorithm. (17) Another key factor that we need to be mindful before running KNN is the dimension (number of features) of the data. KNN does not perform well with high dimensional data. The

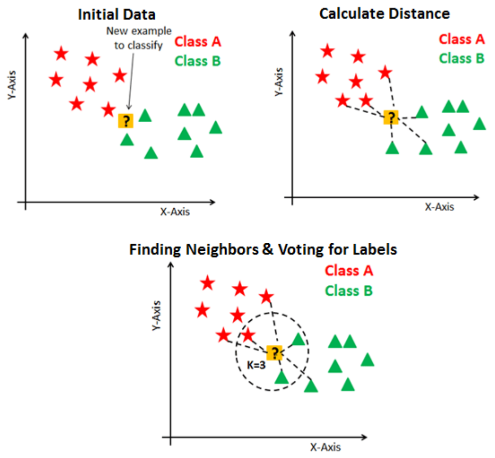

Figure 6: Illustration of KNN Algorithm

data instances need to grow exponentially with increase in dimension for KNN to perform well without over-fitting.

In this study, to deal with the curse of dimensionality stated above, Principal Component Analysis has been conducted before applying KNN, which also takes care of feature scaling.

3.4 Convolutional Neural Network

One of the most popular deep neural networks widely used nowadays for image classification is the Convolutional Neural Network (CNN). It takes its name from the mathematical linear operation between matrices called convolution (18). The Convolutional Neural Networks are composed of multiple layers, some of them are:

- Convolutional Layer: This layer is the core of a convolutional neural network. It is built by a set of kernels and computes the dot product between the entries of the filter and the input of the layer. An image after passing through a convolutional layer is extracted to a feature map.

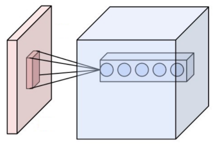

Figure 7: Representation of a convolutional layer.

- Pooling Layer: This layer is responsible to reduce the dimensions of the data. It combines many outputs into a single output as input for the next layer 9. There are different types of pooling layers based on how is the computation performed. It may compute a max or an average. The Max-Pooling is widely used nowadays.

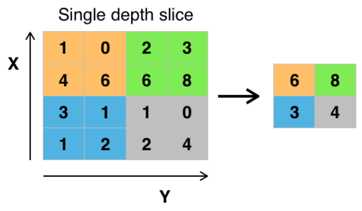

Figure 8: Representation of a pooling layer with a $2x2$ filter.

- Fully-connected Layer: This layer connects every neuron in one layer to every neuron in the next layer. It is based on the principles of the multi-layer perceptron (MLP) (18).

The architecture of our network is summarized in Figure 9. It contains an input layer, three convolutional layers and one fully-connected. The output of the fully-connected layer is fed to a softmax function which produces a distribution over the 62 class labels.

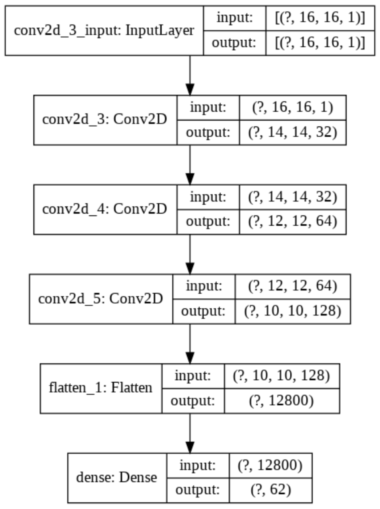
conv2d_3_input: InputLayer | input: [(?, 16, 16, 1)] | output: [(?, 16, 16, 1)]
conv2d_3: Conv2D | input: (?, 16, 16, 1) | output: (?, 14, 14, 32)
conv2d_4: Conv2D | input: (?, 14, 14, 32) | output: (?, 12, 12, 64)
conv2d_5: Conv2D | input: (?, 12, 12, 64) | output: (?, 10, 10, 128)
flatten_1: Flatten | input: (?, 10, 10, 128) | output: (?, 12800)
dense: Dense | input: (?, 12800) | output: (?, 62)

Figure 9: Architecture of the Convolutional Neural Network implemented.

4. Experiments and Results

In this assignment, we implemented different classifiers and performed a series of experiments to analyse the performance and determine which algorithm is the best that achieves the highest accuracy on the test set.

The dataset used in our experiments is the Chars74k dataset (English language) which consists of characters in natural images. In this case, we selected the synthesized set of characters. This dataset contains 62992 images of characters from computer fonts. The

Figure 10 presents some random images extracted from the dataset.

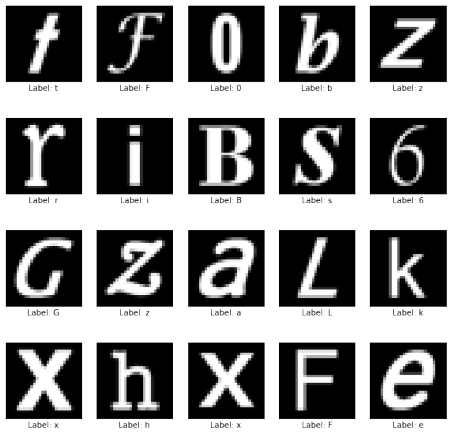

Figure 10: Random examples from the Chars74k dataset.

We will use fine-tuning techniques to test and fine-tune the hyper-parameters of our algorithms. Finally, we will compare the different classifiers using the accuracy metric.

4.1 Metrics

Accuracy: The performance of our classifier will be evaluated in terms of the top-1 accuracy metric:

$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{Total number of test examples used}} * 100\%$$

4.2 Hardware

The code has been developed using Python 3x and Jupyter Notebook and we have taken advantage of the computational capacity provided by Google Colab. However, a computer with the following specifications could easily run these experiments:

Model	MSI GE63VR 7RF
Processor	Intel Core i7 7700HQ 2.80GHz
RAM	16gb
Graphics	GTX 1070 8GB

4.3 Random Forest Results

Whilst the random forest classifier by sklearn has many parameters automised to max out the training accuracy, it is important to tune the parameters manually by using the validation set. After attempting to use grid search validation methods, the model was taking an extremely long time, so it was decided to tune the parameters separately. The number of trees to use in a forest will often improve the accuracy up to a point, after which the improvement rate will plateau, meaning no more trees should be added. Whilst in this model it would not cause overfitting, because of the nature of how a random forest works, having too many trees will greatly slow down the algorithm for little to no improvement in accuracy. The following graph reinforces the plateauing effect after the number of trees is increased.

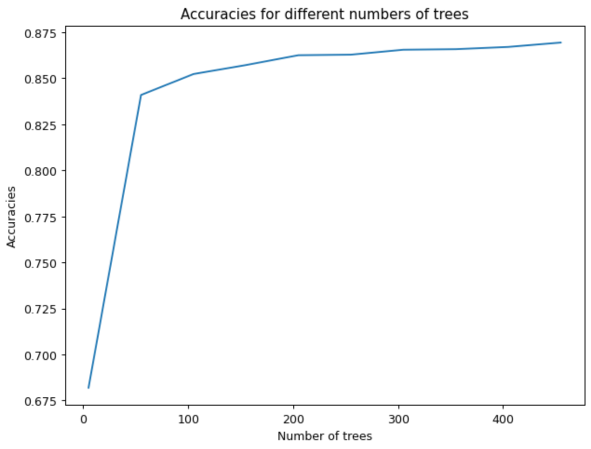

Figure 11: Accuracy as number of trees is changed

The elbow in Figure 11 is quite evident, showing that at around 200 trees the accuracy improvement is minimal. However, in this example as the classifier is not taking too long to run, the maximum from the graph is selected, meaning 455 trees would be used. The next hyperparameter to be tested is the criterion method, the options are Gini or entropy. Comparing the accuracies when using either of those two measures shows very little difference, with the entropy measure coming out slightly on top. The last parameter to be tuned is the maximum number of features to be considered at each split. Whilst there is an automatic function to calculate the maximum number of features, doing it manually produces a different result; it shows that using 15 features gave the best validation accuracy.

The automatic tuning for this variable may improve the training accuracy, however it does not always translate to increase accuracy on testing data, which is what happened in this situation. These results re-iterate the need for a validation set separate to the training set for hyper parameter tuning. The final hyper-parameter left to train is the maximum depth to use for each tree in the forest; this is an important factor as it can reduce overfitting drastically in some situations. The following figure 13 shows the accuracy values for different maximum depth values.

Whilst in a simple decision tree this parameter would show over fitting and a reduction in validation accuracy as the maximum depth increases, this is not as much the case here, since the random forest uses hundreds of decision trees in conjunction. Whilst there is a clear elbow in the figure at a maximum depth of around 40, the actual best value of 180 was used, since this was already a huge reduction compared to the default (no maximum

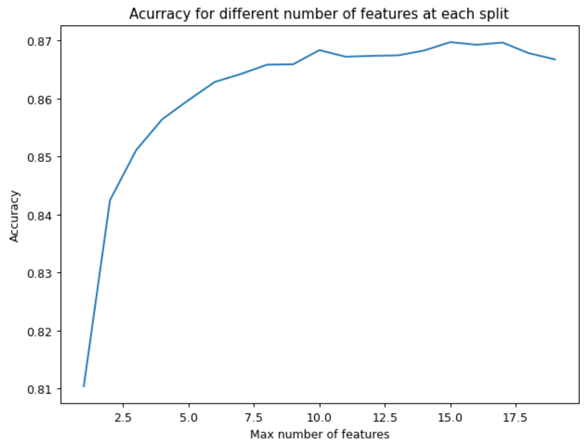

Figure 12: Accuracy as number of maximum features is changed

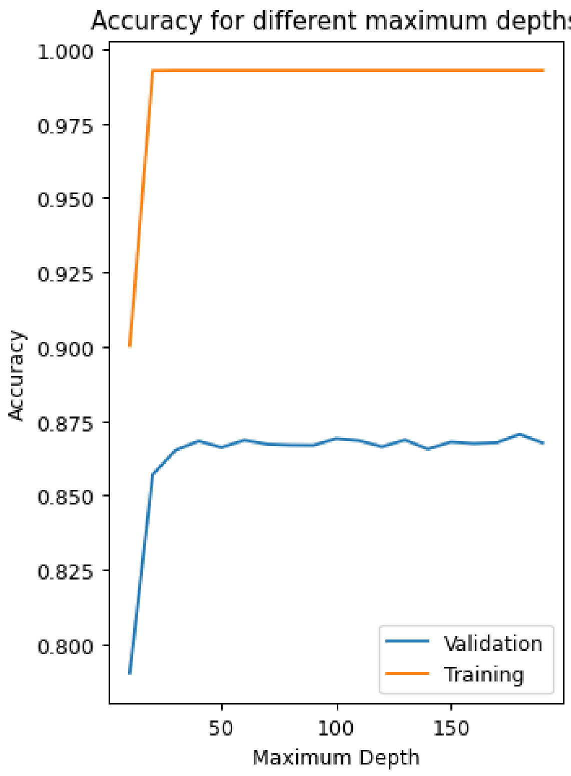

Figure 13: Accuracy as the maximum depth is changed

depth) and it reduced the algorithm run time substantially.

Whilst cross-validation is not always necessary for a random forest, as it has a bootstrapping function within the classifier, a 10-fold cross validation was conducted using the training set and the validation set. Doing a cross-validation of this type gives a more general accuracy result and, in this case, ended up giving an accuracy of 88.55%. The final accuracy for the optimal random forest classifier was then calculated with the completely untouched test and that gave a final value of 87.02%. The below Figure 14 is a plot of the first decision tree of the forest:

Figure 14: Decision tree 1 in the Random forest

4.4 K Nearest Neighbors Results

For K-Nearest Neighbor (KNN), choosing the ideal number of neighbors, that is the value of k is crucial. The smaller value of k might be influenced by noisy points whereas larger value of k might be under the risk that the neighbourhood may include points from other classes, also it can make it computationally very expensive. To obtain ideal k, a series of iterations of the algorithm was made on the validation dataset and accuracy was calculated for each iteration. Figures below show how accuracy changes over different value of k. Initially a wide range of k was tested from which we comprehended that for our data, large value of k will not produce desirable result. Thus, in the second set of iterations a smaller range of k was tested (shown in figure 15). Even though the highest accuracy was achieved at k=1, it is never advisable to choose the value of k so low since it is highly unstable and might lead to over fitting the model. As a trade off in this scenario we choose ideal k=5 where we have a reasonable accuracy level (82.5%) and we do not run the risk of over fitting the model. A lower value of k might be justifiable for our dataset since the labels are balanced and images for each character are quite similar with minimal noise.

Once the value of k is decided, we move to tune the next hyper-parameter. With sklearn there are quite a few parameters that we can tweak, out of which we decided to choose the best weight metric parameter and best power parameter by fine tuning our model. Weight metric defines what weight function to be used in prediction, and both 'uniform' and 'distance' metrics were tested, and it was observed that 'distance' metric performs well with accuracy of 84.64% (shown in figure 16). Multiple power parameters were also tested and it is observed that optimal power parameter p is 2 which is equivalent to using 'Euclidean Distance' (shown in figure 16).

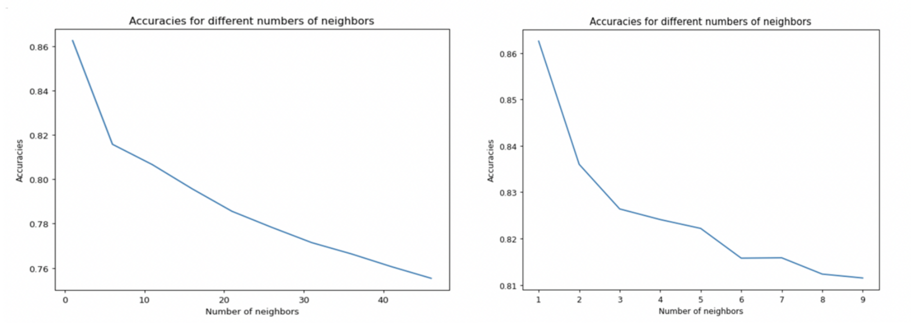

Figure 15: Left:*Accuracy vs a wider range of value of k.* Right:*Accuracy vs a smaller range of value of k.*

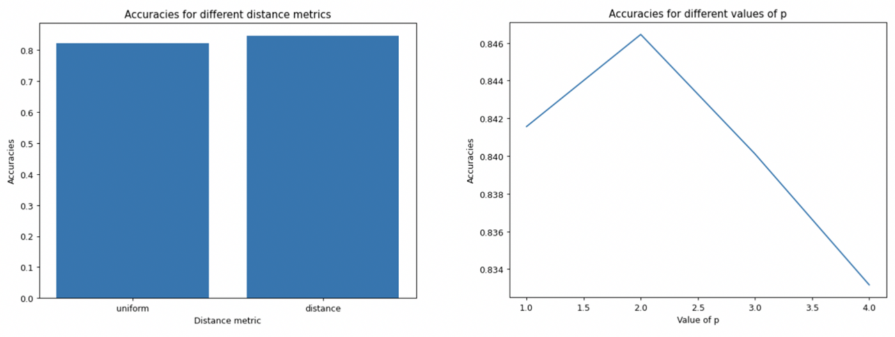

Figure 16: Left:*Accuracy for different weight metrics.* Right:*Accuracy for different power parameter.*

Once all the optimal model parameters were determined, these parameters were then used on the test dataset and we got 83.81% accuracy. To re-assure this result a 10-fold cross validation was conducted using training and validation dataset together, which ultimately produced an accuracy of 85.41%, that proves that our model parameters are reliable, as we observe similar result at different folds. We will further use these cross validation accuracy to compare our models.

4.5 CNN Results

To tune the parameters of the convolutional neural network we used the validation and test set to find the optimal parameters for our CNN. In this case, we performed the fine-tuning to find the optimal values for:

- Number of filters: This is the number of neurons in each convolutional layer of our architecture.
- Number of kernels: This is usually chosen empirically and it depends on the task we want to perform.

The Table 4.5 contains the different values for the number of filters and the number of kernels we have experimented with:

Filters	(32, 64, 128)	(32, 128, 356)
Kerels	(3, 3, 3)	(3, 4, 5)

Table 2: Number of filters and kernels for fine-tuning.

After performing the experiments we found that the highest accuracy in the test set is obtained with:

Filters	(32, 128, 356)
Kerels	(3, 3, 3)

Table 3: Configuration for which we obtained the highest accuracy in the test set.

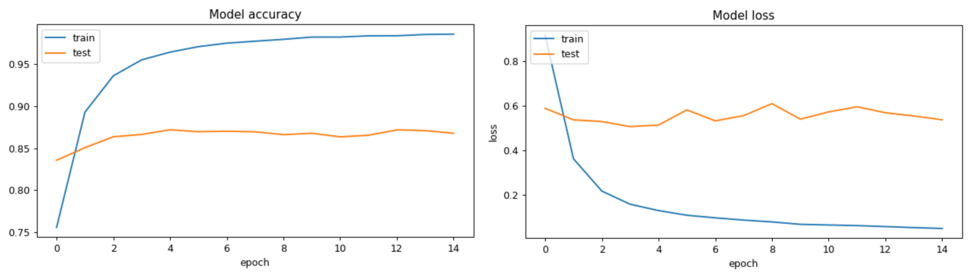

Figure 17: Left:*Train and validation accuracy at each iteration.* Right:*Train and validation loss at each iteration.*

The accuracy obtained using this configuration was of 87% on the test set. Additionally, the left-hand panel of the Figure 17 shows the evolution of the train and validation accuracy at each iteration. We can observe how the CNN achieves high accuracy on the test set from the first iterations, around 86% - 87%, while the accuracy for the training set increases with each iteration. On the other hand, the training loss decreases at each iteration but the test loss remains practically constant over all iterations. This leads us to the conclusion that we should implement some methods to try to reduce this error, for example by adding dropout layers or other methods of regularization.

4.5.1 Model Comparison

In this section we will compare all the classifiers we previously demonstrated. Although we will only focus on the accuracy metric, it is important to mention that the Knn model has the slowest runtime and to solve this, we should implement multiprocessing to train the model using the different CPUs of the machine. On the other hand, the Random Forest model and the Convolutional Neural Network have a similar training time.

The Figure 18 shows the accuracy distribution computed using 10-fold cross-validation for the Random Forest and K-nearest Neighbours classifiers. It can be observed how the accuracy obtained on the test set is higher for the Random Forest than for the K-nearest Neighbours.

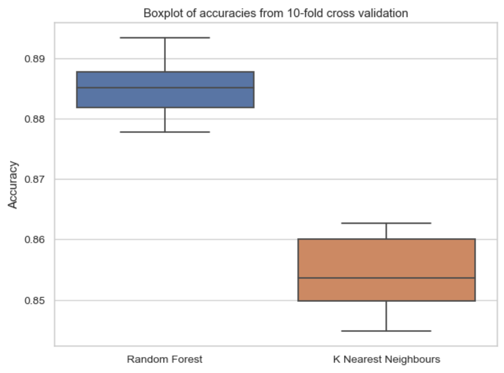

Figure 18: Accuracy comparison of Random Forest and Knn classifiers using 10-fold cross validation

As we previously observed, the optimal value for the number of neighbor k in the K-nn algorithm is 5, which indicates that we are taking into account the k nearest images to make the prediction. This dataset is perfectly balanced and the images do not contain noise which can justify a low value of k. Although it is common knowledge that the K-nearest Neighbours classifier is very flexible since it does not make any assumption on the data, it seems that in this case, the Random Forest algorithm performs better.

Classifier	Acc
Random Forest - $Estimators = 455$ - $Features = 15$ - $Depth = 180$	**0.870**
K-nn - *k=5*	0.838
CNN - $filters = (32, 128, 356)$ - $kernels = (3, 3, 3)$	**0.870**

Table 4: This table presents the results of the classifiers trained on the entire training set and the accuracy achieved on the test set.

The Table 4 shows the accuracy obtained for each model and we can observe that the performance of the Random Forest is pretty similar to the performance achieved using a Convolutional Neural Network. Additionally Figure 19 below shows the confusion matrix for the Random Forest classifier. Whilst all the models have reasonable accuracies we believe that the only model that has room for improvement would be the Convolutional Neural Network.The precision and recall scores for all models were identical to the accuracy so we did not include them in the table. We have tuned most of the parameters in the K-nn model and the Random Forest classifier so we do not believe the accuracy for either model can be increased further. However, the Convolutional Neural Network could be expanded on with additional layers being added, so it could very well be the superior model for this dataset.

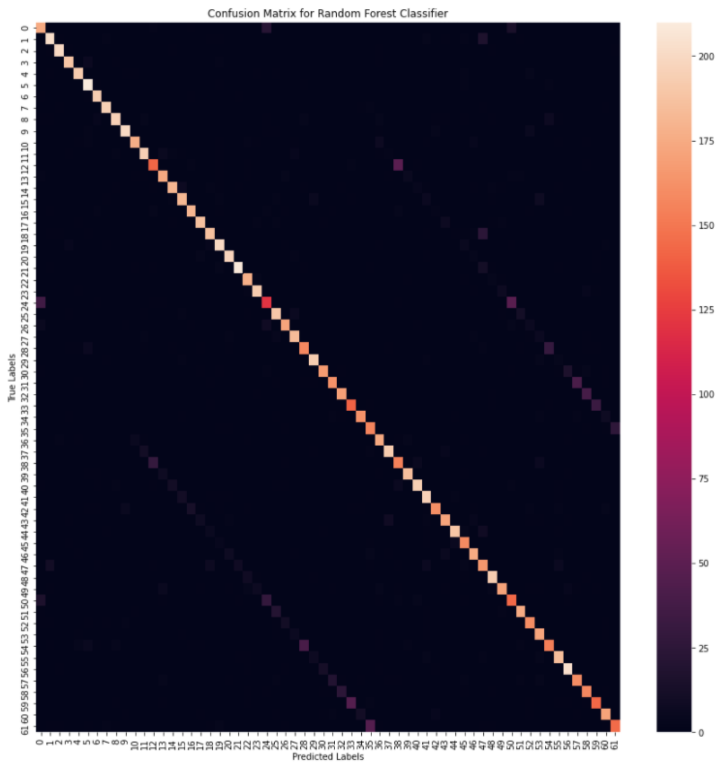

Figure 19: Confusion matrix for Random Forest classifier

Figure 20 below shows 16 examples of labels predicted by the optimised Random Forest model, the 8 on the left were correctly predicted, and the 8 on the right were incorrectly predicted.

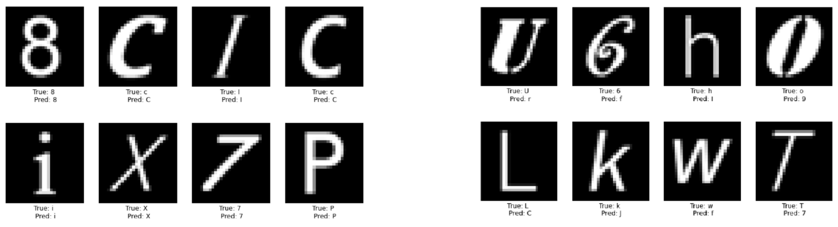

Figure 20: Left:*Correct predictions generated using Random Forest.* Right:*Incorrect predictions generated using Random Forest.*

5. Conclusion

In this assignment, we have implemented three different classification algorithms and we discussed the use of dimensionality reduction to improve the performance of our models. We fine-tuned the hyper-parameters of the classifiers using training (60% of the data) and validation set (20% of the data) and finally applied the optimal parameters on the test dataset (20% of the data). To further support the reliability of our model we performed 10 fold cross-validation. All the classifiers had reasonable performances, but Random Forest and Convolutional neural network has outperformed K-nearest neighbour with an accuracy of 87%. As CNN is one of most promising algorithms in the field of image recognition, we believe adding more layers might improve its performance further. Lastly, we choose Random Forest to be the most suited algorithm for our dataset because it gave the highest accuracy by a small margin.In terms of computational intensity all three algorithms were quite similar. We ensured that python used all processors in the models where available to further reduce run time.

In the future, we can extend this study in the following aspects:

- Compare the performance of these models on the Chars74k dataset with characters obtained from natural images.
- Increase the number of layers in the convolutional neural network.
- Use pretrained models for this task.

References

[1] T. de Campos, “The chars74k dataset,” 2012.

[2] “Optical character recognition,”

[3] “Optical character recognition (ocr) - important feature in tech world,” 2019.

[4] B. R. B. Teofilo E. de Campos and M. Verma, “Character recognition in natural images,” 10 2009.

[5] A. Viklund and E. Nimstad, “Character recognition in natural images utilising tensorflow,”

[6] V. Sundaresan and J. Lin, “Recognizing handwritten digits and characters,”

[7] P. E. v. d. H. J. van der Maaten, L., “Dimensionality Reduction: A Comparative Review,” 2009.

[8] I. T. Jolliffe and J. Cadima, “Principal Component Analysis: a review and recent developments,” *Philos Trans A Math Phys Eng Sci*, 2015.

[9] W. F. Zwick, William R.; Velicer, “Comparison of five rules for determining the number of components to retain,” *Psychological Bulletin*, 1986.

[10] A. D. Santos, “Assignment 1 comp5318,” 10 2020.

[11] T. Yiu, “Understanding random forest,” 6 2019.

[12] M. Y. T.-Y. C. Yu Ting Ho, Chun-Feng Wu and Y.-H. Chang, “Replanting your forest: Nvm-friendly bagging strategy for random forest,” 2019.

[13] L. Breiman and A. Cutler, “Random forests,”

[14] O. M’Haimdat, “Understand the fundamentals of the k-nearest neighbors (knn) algorithm,”

[15] M. Chatterjee, “A quick introduction to knn algorithm,”

[16] A. Navlani, “Knn classification using scikit-learn,”

[17] P. Sharma, “Why is scaling required in knn and k-means,”

[18] S. Albawi, T. Abed Mohammed, and S. ALZAWI, “Understanding of a convolutional neural network,” 08 2017.

Appendix

The experiments of this work were executed using Python 3x in Google Colab. The code is provided in two Jupyter Notebooks called:

- *group43_other_algorithms.ipynb*:
  - This notebook contains implementation of all the classifiers that were experimented in this study including the one with highest accuracy (it was left in that file to show the tuning process).
  - To execute the code, you need to follow below steps:
    1. Data to be used can be found here. Please download the 'EnglishFnt.tgz' file which contains characters from computer fonts with 4 variations (combinations of italic, bold and normal). Unzip the file and save the "English" folder in the same directory as the .ipynb file.
    2. Ensure there is a folder called 'data' in the same directory as the .ipynb file so the .h5 files can be saved.
    3. Run codes under Libraries to load the required libraries
    4. Run the codes under 'Data Loading' section to load the data, if successful the shape of the data should be (62992,784) and for the labels (62992,).
    5. In the same chunk, we have provided codes to convert the data in .h5 file ("Converting to .h5 file") and also read from .h5 file ("Loading data from .h5 file") for future work.
- *group43_best_algorithm1.ipynb*:
  - This notebook contains the implementation of the classifier which achieved the highest accuracy on the test set.
  - To execute the code, please make sure the data is already converted into h5.file(as instructed above), then you just need to go through the notebook and execute all cells.
- *Group contributions*:
  - Everyone contributed evenly, ie (33% each).