Chars74K Image Classification

Group 83

Jesse Narvasa(500525438)

Mimansa Rana(490459654)

Yuxi Shen(490548486)

Tutors:

Yixuam Zhang

Kunze Wang

Abstract

This paper outlines methods to create a classification model to predict the label of an image within the Chars74k dataset using a combination of techniques. Both Neural Network as well as traditional machine learning algorithms like SVM, K Nearest Neighbor, tree ensemble are used to build models. The best performing model uses a Convolutional Neural Network providing 82.62% accuracy.

Introduction

This study will focus on building a model capable of predicting the correct alphanumeric character within the English alphabet (Latin script) or the Arabic numerical system (0-9), based on the extracted character from an input image taken from natural scenes. Due to the omnipresence of smartphone cameras, video cameras etc. in everyday life, there are many applications of this type of image to text recognition. Applications of this technology are vast and varied, from the ability to digitise printed documents, language translation on images, all the way to street sign recognition for autonomous driving, thus placing a great importance on further research within this area and exploration of possible further enhancements which this study aims to accomplish.

Optical character recognition is one of the growing fields of research within computer vision due to the multitude of applications outlined above, and the selection of the Chars74k dataset provides us with the foundation to build and prototype our model on. The dataset contains a selection of lowercase and uppercase segmented characters from everyday scenes, with a range of font size, color, styling as well as image quality providing the diversity required of a good dataset for training and testing. Specifically, we will be limiting the scope of this study to uppercase and lowercase English alphabet as well as the numerical system as provided within 'EnglishImg.tgz' by the University of Surrey (University of Surrey #).

The study will be focusing on the following five algorithms: SVM, k-Nearest Neighbour, Random Forest, XGBoost, and CNN to determine the best performing classifier for this problem, as measured by top-1 accuracy scores. Image pre-processing and feature extraction techniques are also considered such as data standardisation, Principal Component Analysis, and HOG; which will be measured against the base performance of the algorithms to determine its usefulness and viability in adding to the classification model.

Previous Work

The Chars74k dataset was introduced in 2009 by T. Campos, B. Babu and M. Verma and consists of images of street scenes taken in Bangalore, India using a standard camera. The paper addresses the classification problem using models such as KNN, SVM and MKL and various feature extraction techniques to improve accuracy (T. E. de Campos, 2009). The Histograms of Oriented Gradients technique was introduced by Dalal and Triggs in 2005. They used this technique with linear SVM to show a high accuracy on databases of images, and they study the influence of each stage of computation (Triggs, 2005).

Convolutional Neural Networks (CNN) are the most popular algorithm for image recognition tasks and usually yield state-of-art performance. In this report, only shallow CNNs are demonstrated because of the limitation of CPU used. Modern CNNs are deep with many hidden layers. For example, the famous ResNet, winner of 2015 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), has 152 hidden layers achieving an error rate of 5.37%. The biggest problem of such deep networks is the problem of vanishing gradients. It occurs when the update factors of weights are tiny and the model stops learning after some epochs. It is a great waste of computation power and will not finish the task with good performance. ResNet proposes to solve this by skipping connections, i.e skip a few layers when vanishing gradients are seen.

Aside from deep neural networks, Generative adversarial network (GAN) is proposed as a way of data augmentation. Different from the data augmentation in this report, which generates new images by directly photoshopping the original images, GAN is a learning algorithm that generates samples by iteratively learning better images. It has a Generator and a Discriminator. Generator will try to generate better samples to pass the test of Discriminator and Discriminator is learning to identify “fake” samples (Evans 2017). After some iterations, some “good” enough “fake” images are produced and fed into chosen neural networks. With more data inputs, algorithms are expected to perform better.

Methods

The study has been conducted through the use of Python 3 and Jupyter notebook/Google Colab for the purposes of data exploration, analysis, as well as algorithm execution, along with the relevant libraries required. Specifically, the Python Imaging Library (PIL) has been used to read through all the images within the directories, with the label/class being assigned based on the directory in which the image is located.

Colour Channels

One of the primary design decisions made during the processing of images is the conversion of the coloured images into grayscale, denoted as “L” by the Python Imaging Library. The images were converted into grayscale to allow for faster processing, as this provides us with less features required to feed into the model by removing the RGB channels normally provided and instead only utilising a monochromatic greyscale channel. With regards to image processing by the classification algorithms, it is not expected to have any material impact with the performance of popular classification algorithms such as CNN (Diaz et al. #)

Image Dimension and Scaling

During the conversion of the PNG images to a Numpy array, where each pixel is converted into greyscale format with values ranging from 0 to 255 as outlined above, it is then required to resize the images into a uniform dimension to allow it to fit into a singular N x M array. This array dimension represents the N number of images we have in our dataset, with M representing the total number of pixels which can be calculated by the height x width of our images. As a result, two different design decisions have to be made with regards to the conversion of our PNG images into a numpy array, namely the resize dimension of the images, and the true aspect ratio of the image should be retained during the handling of the resize.

In order to determine this, the dimensions for all images are taken with the median height and width of 68 and 46 pixels respectively; with the minimum being 9 x 3 and maximum image dimensions being 648 x 536. Since upscaling small images to a larger size results in a greater loss of detail, and expect a greater penalty towards the performance of our classification models (Dennanni), we have therefore chosen to avoid stretching the small images to a larger size, in favour of downscaling the majority of larger images to a smaller size. Moreover, large images such as the one mentioned above are greater than 12,000 times bigger than the smallest image we have within the dataset. Therefore, in order to strike the balance between avoiding upscaling small images and downscaling large images too much, we have chosen the square dimension of 48 x 48, which is close to our median dimensions, but leaning towards the smaller size. Square dimensions are also safe and easy to work with CNN (Leopd 2020).

It is also important to consider how the images will be resized, which covers the second part of the design decisions within this subsection. The simplest method available would have been to stretch the image to fit the 48x48 dimension specified above, resulting in non-square images to stretch or shrink in a manner that loses it's true aspect ratio. This is important since general intuition is that the resizing of images that results in the loss of it's aspect ratio can, at extreme cases, deform it beyond recognition. However, there doesn't seem to be concrete evidence on the impact of scaling an image that is not true of it's aspect ratio with performance of machine learning algorithms. As such, we have chosen to generate two sets of datasets to compare performance between algorithms, one which is scaled with respect to it's aspect ratio, with the remaining space padded by white pixels, and the second dataset containing the stretched image. For the purposes of this experiment, we will be using the stretched dataset as the base parameter.

Image Mask
Finally the mask is applied to each corresponding image. The masks for each image are provided in a separate folder "Msk". The masks aim to filter out the unwanted background information and help the algorithm to focus on the actual character. Pixels are black (with value 0 after grayscale) for background and white (with value 255) for areas containing objects in the masks. Then masks are applied by setting values in background areas of the original image to 0. Masked data will be used instead of original images.

Classification Labels
The label data is converted into integers as some of the models we will use will only accept numerical label data, such as CNN and XGBoost.

Transformations and Scaling
The dataset is already within the range of 0 to 255, as such normalisation is not required to be considered (due to the fact that all normalisation does is control the dataset to be within 0 and 1). It is no different to our data being between 0 to 255. Standardisation on the other hand is quite different and can be considered, and is required to conduct for PCA anyway.

Measure of Performance
The comparison between classification models will be measured through a variety of measures, however will predominantly be against the top-1 accuracy score attained through it's 10-fold cross-validation.

Splitting

The data was then split using a train/test ratio of 0.75:0.25 using **stratify = labels and shuffle = True** as arguments of the train_test_split function in order to randomise the order of the input data (it was read in order from the files). The training set will be used as the input of algorithms to do experiments, hyperparameters tuning, model selection. The test set will not be used until the final model evaluation.

Machine Learning Algorithms
As outlined within the introduction section, the following classification algorithms are considered within the study, and will be compared against each other through a set of performance measures, to check suitability against the nature of our dataset. This section will briefly explain the relevance of the classifier, as well as its key advantages that makes it a viable choice for the problem at hand.

1. Support Vector Machines (SVM), Histogram of Oriented Gradients (HOG)

The essence of the Support Vector Machine algorithm for a linear binary classification problem is to try to find a hyperplane(line in 2D space) that will separate the feature space into 2 regions or halfspaces, in a way that each halfspace only contains a single class of data. Samples of each class closest to the hyperplane will constitute the Support Vectors and these are the . If the dataset is not linearly separable, the algorithm states that the original feature space can always be mapped to some
higher-dimensional feature space where the training set is separable. SVM then uses a kernel function to implicitly map data to a higher dimensional space where the data is linearly separable.

Support Vector Machine was chosen as one of the algorithms as the algorithm can be applied to complex data types by varying or designing the kernel function. The algorithm provides versatility to solve for image data in this experiment and many other types of structured or semi structured data in general due to the ability to tailor the kernel. It is often used in image recognition or classification problems
Although not intrinsically capable of multi-class classification, this can be worked around by splitting the multi-class classification problem into multiple binary class classification problems. In sklearn this is done using the SVC() function as well as setting the parameter **decision_function_shape = 'ovo'** or one versus one.

Histogram of Oriented Gradients (HOG) is a feature engineering technique that is used by many papers in computer vision, including Triggs, 2005), (Yash Sinha, 2016) , and (Greeshma K V, 2020). In these instances HOG is used on input images, and the result is fed into a non-Neural Network based algorithm such as KNN and SVM.
The technique breaks down an image into smaller parts and calculates gradient vectors as well as magnitude and direction of features on each pixel within these localised parts of the image. This means the magnitude and direction of small changes in x and y values is calculated from 1 pixel to the next. A sharp change in magnitude and direction from one pixel to the next, for example, may represent an edge. By doing this over small portions of the image, we can get a representation of the shape and structure of objects within the image. A histogram for each localised region is then generated, and the magnitude of the gradients is binned into a number of pre-specified buckets. These gradient values are calculated and binned iteratively over the entire image in pre-set cell block sizes. The resulting histograms values are concatenated into a 1 dimensional vector and normalised to a unit weight (Weng, 2017) (SINGH, 2019)
The function is available in several libraries including sklearn and openCV. In this experiment, HOG function within sklearn is used, resulting in a HOG feature descriptor vector of length 72 per image.

2. k-Nearest Neighbours

K Nearest Neighbours uses the majority class of ‘k’ nearest neighbors to determine the class of a given data point. The distance to the nearest neighbours can be calculated according to various distance, similarity or proximity measures such as Euclidean or Minkowski distance.
KNN algorithm was chosen for this experiment as it is simple to implement, intuitive to understand for multi class classification problems with few parameters to tune, and provides a good accuracy on image without much pre-processing. It usually works well on data that is not linearly separable, such as image data. However KNN is a lazy learner, ie, it uses training data to perform classification. For higher dimensions of data, KNN models can become slow, unstable or inaccurate.

3. Random Forest

Amongst the classification models considered is Random Forest. Although generally associated with datasets containing nominal/ordinal data, the use of random forest is growing in areas of applications that contain continuous data such as satellite imagery (Horning #). The main advantage of the use of Random Forest is it’s robustness with the ability to generalise the dataset and avoidance of overfitting, which is the primary caveat of traditional decision trees. This is in large, thanks to the model making use of a tree ensemble, and in particular, using a tree ensemble type of bagging as opposed to boosting, which will be covered in the next algorithm. The superiority of bagging tree ensemble algorithms when compared to a singular decision tree is its use of the wisdom of crowds, whereby a collection of uncorrelated/low-correlation decision trees will each have it’s own fault and bias when it comes to predicting; however as a whole, will (most of the time) arrive at the correct conclusion. In this instance, random forest implements majority voting, whereby the different trees resembling the random forest will each have its own prediction, with the majority class being assigned as the prediction for the test instance.

In order to truly understand how random forests are able to achieve little to no-correlation between the decision trees contained within, it’s important to understand how a decision tree is constructed and how random forest compares to it. Within decision trees, internal nodes are constructed on the basis of statistics or heuristic with the goal of splitting the dataset, such that the resulting leaf nodes will contain pure samples. In order to achieve this, scores are calculated against all of the available features within the dataset, with the feature that will achieve this split the best, being chosen as the internal node for that split. The calculation of this score is also against all the training dataset available, which is also partly why it’s prone to overfitting, which is that it always looks at the same training dataset. As a result, the outcome of the decision tree will be a largely low bias but high variance model, with class imbalances within the training dataset placing a greater importance within the model and certain features having more prediction power than it should. In contrast, random forest generates decision trees in parallel. The main difference is that the number of available features for selection on each node splitting will be less than the total number of features available. This results in decision trees where the features used are not always the same and hence contributing to it’s lower correlation.

Apart from the subsampling of features, bootstrapping is also used for the generation of the training dataset at each node split. This means that as opposed to using the same dataset at each node split to calculate the statistic for each of the subsampled features, a new dataset is instead generated every time. The way the bootstrapped dataset is created is through repeated sampling of the original dataset, with replacement, until the size is the same as the original dataset. The primary difference is that we can select the same entry more than once, hence allowing us to potentially reduce the effects of class imbalances.

4. eXtreme Gradient Boosted (XGBoost) Classifier

Similar to Random Forest, eXtreme Gradient Boosted trees is a classification model that belongs to the ensemble trees method and is recognised as one of the most powerful supervised machine learning algorithms available, based on recent performances within Kaggle competitions (Hadidi 2016). Due to its popularity amongst the data science community, as well as its relation to random forests, this algorithm will be considered within this study. In comparison to random forests, XGBoost is an ensemble method that performs it through boosting. This means that instead of creating trees in parallel, XGBoost instead creates one decision tree at a time, however with each boosting sequence, the tree attributes are adjusted through its eta such that previous mis-classifications are accounted for in the creation of the next boosted tree. What results is a sequence of trees being created, with subsequent trees being an improved version of its predecessor.

However, similar to decision trees, XGBoost can also be prone to overfitting when run with suboptimal hyperparameters. This is due to the fact that each boosting sequence will result in the next tree based on the mistakes of the previous tree. As a result, it is important to consider in the designing of the model to make use of regularisation, boosting, tree pruning and/or other similar techniques that prevents the model from leaning too much into the provided training set.

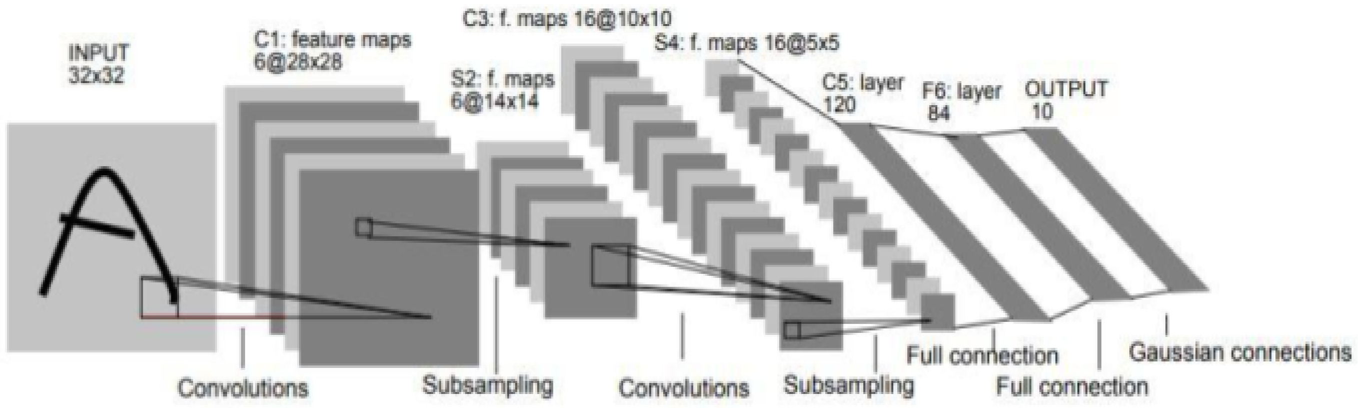
5. Convolutional Neural Networks

Convolutional neural networks(CNN) are so powerful that recent winners of ILSVRC(Image Large Scale Visual Recognition Challenge) are all variations of CNN. CNN is intuitively useful for image recognition since it imitates the process of how humans recognise patterns in an image. Humans break down an image to several parts that are recognisable and classify the image according to experience. For example, in a hand-written digits task, humans would classify any digits with two circles as the number “8” even if it is badly written (e.g oval shaped circles, disjoint circles). Because it is the only digit that possesses the feature of two circles.

Convolutional neural networks work in a similar way. It passes filters that would pick up certain features on the whole image. The filters are of small sizes (e.g 3 × 3 or 5× 5) compared with image sizes (e.g 48×48). When the filters are sliding over the image, features that fit the filters will be extracted and stored within a feature map. This operation is called a convolutional layer. CNNs are essentially models with multiple layers. A second type of layer is the pooling layers. Pooling layers will pool the values within the window of pool size together by taking the mean or maximum values in the pool, depending on the type of the pool layer. This layer would continue to compress the image size and still retain useful information. However, this is not the main reason for pooling. Pooling is needed to make features translation invariant, which means features are still recognisable when the image is stretched, compressed or rotated. Take the example of digit "8" again. After pooling, the model will even classify "∞" as "8", given that "∞" is not one of the target classes. Another must-have layer is the Dense layer. This fully connected layer ensures that classification is possible by mapping the results of previous layers to predicted class labels. Before the Dense layer, one Flatten layer is needed to flatten the outputs of convolution layers to get a vector of probabilities.

With these basic concepts in mind, a baseline CNN model is built. This simple CNN model has only one convolutional layer: a 2D convolutional layer with 64 feature maps computed by 5×5 kernels (i.e filters), following a subsampling layer. The subsampling layer will take the output from the previous layer and pool together the best value among the 2×2 pool window. Then with flattened data, two Dense layers are applied to perform classification. The latter Dense layer always has exactly 62 units for this 62 classes classification task. The former Dense layer's aim is to create information bottleneck. The number of units need to be larger than 62 and smaller than the output shape of the previous layer (33,856 in this baseline model). Activation function used in this model and most models discussed in this report is the rectified linear unit activation function, also known as 'relu'. The exception is the last Dense layer of every model, which uses 'softmax' to obtain categorical probabilities. This model is trained using the Adam optimiser, which is the go-to choice of optimisation method that adapts learning rates to reduce training costs (Brownlee 2017). The loss value is measured by sparse categorical cross-entropy because the internal labels used are integers.

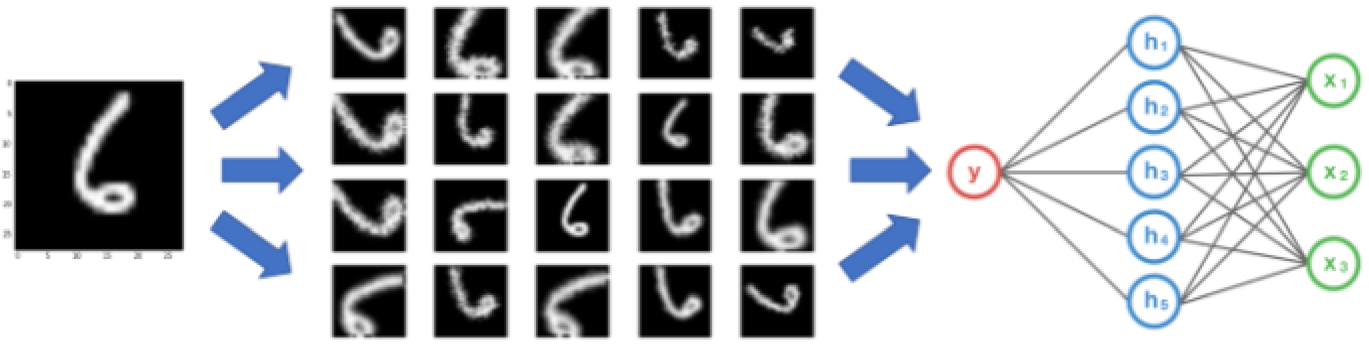
A second model with one more hidden layer is constructed to possibly improve performance. It is a common approach to increase the depth of neural networks to extract more important features. LeNet-5 (created by Yann LeCun) with two sets of convolutional layers and subsampling layers are suggested. The first Layer C1 has 6 feature maps that are generated by sliding non-overlapping kernels of size 5×5. The term "non-overlapping" indicates that the stride per slide is the same size as the kernel. Then the subsampling layer S2 will maxpool with pool size 5×5. The next set of hidden layers would be very similar except the doubled number of feature maps in the convolutional layer. The additional number of feature maps would help the model to retain more structures while downsampling the input size. Then the model finishes off with a standard Flatten and two Dense layers structure (Maitra, Bhattacharya, and Parui 2015, 1021-1025).



LeNet-5 Architecture ("LeNet-5 – A Classic CNN Architecture" 2018)

After the exploration of two simpler CNN models, Keras tuner is used to grid search through some hyperparameters of CNN. As suggested in the literature of GoogLeNet (Tripathy 2016), concatenating layers of convolutions along with pooling and small kernel size could exploit local information of data. This approach should fit the task of image recognition very well as local information on an image is highly correlated. Keras tuner was given the choice of zero to three convolutional layers stacking together before a max pooling layer. Two sets of this structure is possible with a Dropout layer in between to reduce overfitting. The units in the former Dense layer are also made as a hyperparameter to control the extent of information bottleneck.

Another technique called data augmentation is applied to the tuned model. Data augmentation feeds slightly altered input images into the network. This method increased the sample size in an indirect way. It lets the model see not only the original data (when augmentation is hardly done) but also the modified data that potentially come from the same distribution. Also, as we explained in the dataset description, despite the large number training samples, the number of classes is also big. With various train and validation splits, some classes in the split might have few samples. Data augmentation could solve the above problem by randomly generating data. (Gandhi 2018)



Data Augmentation Demonstration (Gandhi 2018)

In the fitting of the final CNN model, a feature in Keras called "ReduceLROnPlateau" is used to help the algorithm to reach local minimum. It is set to reduce learning rate by 0.1 if the validation accuracy stops to show any improvement for consecutive 3 epochs, which possibly indicates the presence of a local minimum. Without this feature, local minimum might be missed and the true strength of an algorithm is not shown.

The evaluation of a given CNN model will not use K-fold cross validation although it was suggested and experimented. The runtime of a complicated model like CNN is affordable in a single run. But K-fold cross validation implies that the runtime will be K times longer. Sacrificing the runtime just to get a good estimate of test accuracy is not worthwhile in my opinion, especially without the help of GPU. Hence K-fold is not used in actual evaluation of any models. Instead, a validation set is splitted from the training set, at a ratio of 8:2 for train and validation respectively.

Experiments and Discussions

Support Vector Machine/K Nearest Neighbours

The approach was to run both algorithms with and without pre-processing (PCA and HOG) applied using 10 fold CV to tune hyperparameters(using only training/validation data and with the exception of SVM with no pre-processing as it took a very long time just to run 1 iteration), and then use the best hyperparameters to train the model, and evaluate performance against test data.

The below table summarizes the results of the experimental setup. The best performing algorithm was SVM after using the HOG feature descriptor to extract features of a square image. The following combinations were tested:

- SVM on a flattened numpy array of masked image data - this is marked as no pre-processing (as there is no experiment specific pre-processing
- KNN on a flattened numpy array of masked image data
- HOG feature descriptor (array of length 72) used as input for SVM
- HOG feature descriptor used as input for KNN
- PCA on masked flattened image (threshold is variance explained ~95%), the dimension reduced 1 D array used as input for SVM
- PCA on masked flattened image (threshold is variance explained ~95%), the dimension reduced 1 D array used as input for KNN

Preprocessing	Technique	algorithm	accuracy	Best parameters after 10 fold CV
No	N/A	SVM	7%	
No	N/A	KNN	46.1%	K = 2
Yes	PCA	SVM	8%	C = 10, gamma = 0.1, kernel = rbf
Yes	PCA	KNN	4%	K = 2, distance = ‘minkowski’
Yes	HOG	SVM	74%	C=10, gamma = 0.1, kernel = rbf
Yes	HOG	KNN	66%	K = 5, metric = ‘manhattan’

The parameters used for the HOG feature descriptor are:
Orientations = 8, pixels per cell = (16, 16), cells per block = (1,1)
The input for the HOG descriptor was a 48 x 48 square masked image.
The parameters that were tuned for SVM are - C, gamma and kernel.
The parameters that were tuned for KNN are - k, distance metric.

PCA provides the lowest accuracies for the data, it could be because PCA on a flattened array may cause important data about the image to be lost when the dimensions are reduced. It would be interesting to compare the results if PCA is applied on the 48 x 48 square image as an example.
HOG + SVM provides the highest accuracy, which is consistent with the results seen in many previous works.
KNN without any pre-processing provides a higher accuracy than SVM, and SVM with high dimension data takes very long to run (possibly as it is being modelled as multiple binary class classification problems, which may take longer to iterate). It would also be interesting to see if using the 'one versus many' option in SVM may provide faster processing time and what impact that may have on accuracy.

Tree Ensembles

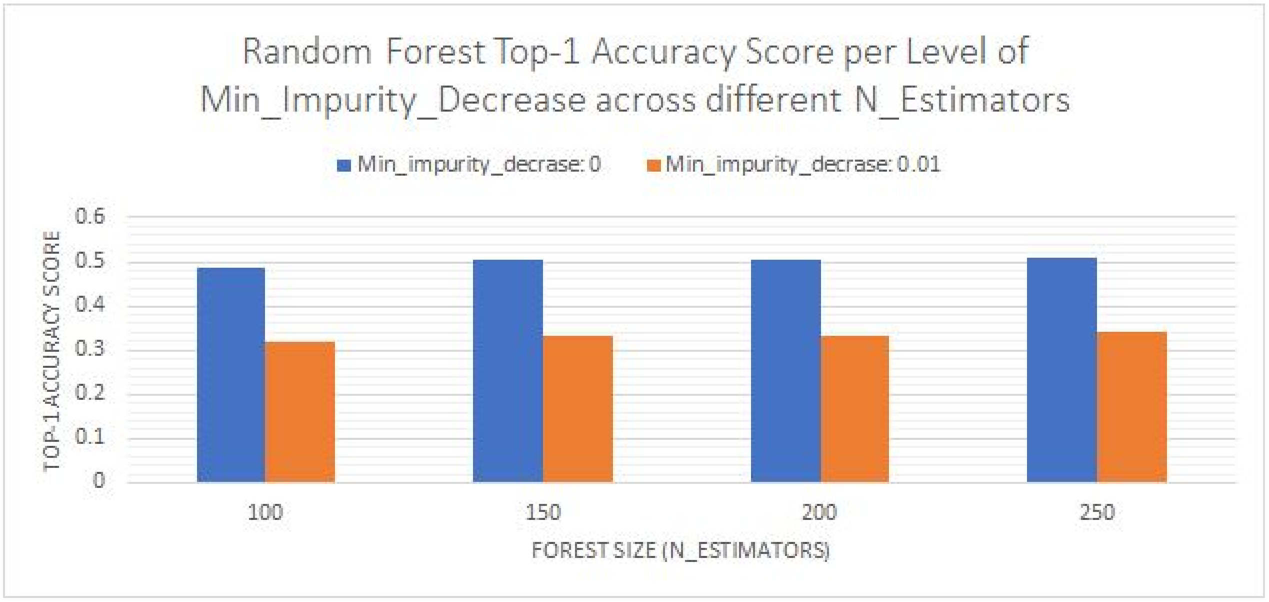
In conducting the experiment for the two tree ensemble methods, namely Random Forest and XGBoost, the following approach has been followed in hyperparameter optimisation. The two classifiers will be run against two different datasets outlined within the Methods section of the report, which are the "stretched" and "true aspect ratio" data. The notion behind running the algorithms against the two sets is to determine the effects that image transformation has, if any, on the performance of the model. Moreover, two sets of runs will be performed for each classifier for each dataset, one with no pre-processing applied, and the second with standardisation and PCA applied on the dataset. This allows us to evaluate whether dimension reduction, as a by-product of PCA, results in positive performance gain within tree ensembles in the context of our problem.

The following four tests are run for each algorithm and is summarised as below, noting that we will be using the stretched dataset without PCA as the base performance for each model:

1. (base) Stretched dataset without PCA
2. True aspect ratio dataset without PCA
3. Stretched dataset with standardisation and PCA
4. True aspect ratio dataset with standardisation and PCA

The consideration of which hyperparameters to optimise for each of the tree ensemble methods has to be carefully chosen, due to computing capacity constraints as well as the number of hyperparameter optimisation runs we'll be required to conduct to make the experiment fair for each set, as outlined above.

For Random Forest, one of the main design decisions made is to set the parameter max_depth, which controls the maximum allowed number of levels within a single tree, to None. Doing so therefore places no limit as to how much the trees within the forest can grow, instead relying on the parameters max_impurity_decrease as well as min_samples_split for preventing overfitting and ensuring that the trees are still reasonably generalised. The parameter min_impurity_decrease is the minimum threshold required on how much impurity will decrease before a node can be considered to be splitted, and we've opted for a conservative selection for our hyperparameter optimisation of either 0, which is the default, and 0.01. The value of 0 is considered since the use of this parameter to control overfitting may not be as favourable as simply controlling the other parameter which is min_samples_split. Min_samples_split is the number of weights, where each instance in our model represents one weight, required to split an internal node. Again, a conservative parameter selection has been used with the inclusion of the default value up to 5.

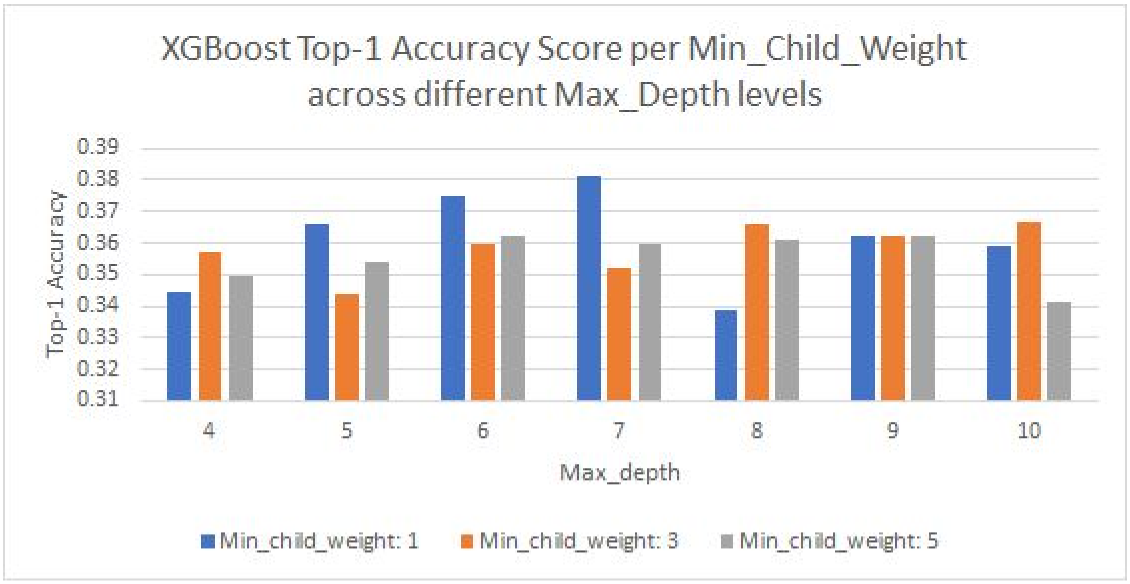


As illustrated on the graph above, setting the value of min_impurity_decrease to 0.01 does result in a significant decrease in accuracy on our model across different levels of n_estimators. This is therefore reflected within the summary table below, with the final set of best parameters having a min_impurity_decrease of 0, but with varying levels of min_samples_split across different n_estimators.

Another important consideration is how random forest considers the split for each node. This is controlled by the parameter "criterion" with either values of entropy or gini index. Our experiment has concluded on a mix between the two for different tests, however it is interesting to find that entropy was consistently higher when used on the stretched dataset, and gini being the winner for the true-ratio dataset. Other parameters considered are class_weight, which is included in an attempt to control class_imbalance within the dataset, and also n_estimators which determine the size of the random forest. Initial expectations is that a balanced class_weight would prevail due to better generalisation of the training set, as well as the selection of the largest n_estimators due to the wisdom of the crowds, explained in the previous section. The expectation on the number of n_estimator turned out to be true, obtaining n=250 for all batches of hyperparameter optimisation, however class_weight of balanced was only found as the best parameter for our base model, which is also our best tree ensemble method.

With regards to XGBoost, the constant parameters selected is to have the objective set to multi-classification with softmax, with subsampling of our dataset set to 75%, and tree_method of gpu_hist to enable GPU implementation of XGBoost for performance gain. This reflects the nature of the problem, as having an objective set to softmax allows us to obtain the classification with the highest probability, and subsampling of the training dataset at each boosting iteration simply ensures that our tree does not overfit.

As previously mentioned within the Methods section of this paper, overfitting can be a problem with XGBoost without the correct set of parameters. In order to control this, the use of early stopping is included in the design of the model, as well as the optimisation of the parameters max_depth and min_child_weight within our gridsearch. The implementation of early_stopping_rounds works by ensuring that each boosting sequence results in an improvement of the performance metric score against the evaluation set specified within the model. In our implementation, we have decided to further split the X_train dataset, to create X_train_2 which is then fed into the 10-fold GridSearchCV object for creating the model, and a separate X_val dataset which will be left unseen by the XGBoost model, but is what we're gonna use to determine if the model should perform an early stop during its boosting sequence. Therefore, if the accuracy has not increased after 10 rounds of boosting sequences, our XGBoost model will stop iterating. The other controls we've added, max_depth is similar to max_depth within Random Forest, and min_child_weight which is similar to min_samples_split in RandomForest, except this is the minimum weight/support needed for the child node to contain for a split to be considered. Even though min_child_weight has been set to the default value of 1 for all instances of the XGBoos configurations that we've tested, the graph below illustrates that different levels of min_child_weight can provide the best accuracy score at different values of max_depth, as was the case for one of the configurations we've tested.



The final outcome of the four different configurations on Random Forest and XGBoost classifiers are summarised below. On both tree ensemble algorithms, the base setup of using the "stretched" dataset (also referred to as "original" dataset within the code file) has prevailed with the highest accuracy amongst the other configurations, with Random Forest achieving the highest tree ensemble score of 54% accuracy, with XGBoost achieving 49.71%.

Dataset	**Pre-processing**	**Technique**	**Algorithm**	**10-fold Cross-val Accuracy**	**Best Parameters**
Stretched	No	N/A	Random Forest	**54.52%**	*class_weight*: balanced, *criterion*: entropy, *min_impurity_decrease*: 0 *min_samples_split*: 5,

					n_estimators: 250
Stretched	No	N/A	XGBoost	49.71%	*learning_rate*: 0.1, *max_depth*: 7, *min_child_weight*: 1
True aspect ratio	No	N/A	Random Forest	**43.42%**	*class_weight*: None, *criterion*: gini, *min_impurity_decrease*: 0, *min_samples_split*: 2, *n_estimators*: 250
True aspect ratio	No	N/A	XGBoost	42.21%	*learning_rate*: 0.05, *max_depth*: 9, *min_child_weight*: 1
Stretched	Standardise	PCA	Random Forest	**43.20%**	*class_weight*: None, *criterion*: 'entropy', *min_impurity_decrease*: 0, *min_samples_split*: 3, *n_estimators*: 250
Stretched	Standardise	PCA	XGBoost*	41.76%	*learning_rate*: 0.05, *max_depth*: 9, *min_child_weight*: 1
True aspect ratio	Standardise	PCA	Random Forest	29.54%	*class_weight*: None, *criterion*: 'gini', *min_impurity_decrease*: 0, *min_samples_split*: 4, *n_estimators*: 250
True aspect ratio	Standardise	PCA	XGBoost*	**30.18%**	*learning_rate*: 0.05, *max_depth*: 9, *min_child_weight*: 1

*It is important to note that the implementation of XGBoost with PCA applied is different in one of the parameters which could have drastically altered the possible outcome of the result. The difference in question is the lack of ability to use early_stopping_rounds within XGBoost with PCA due to lack of support in being able to access the pca.transform function which was required to be used on the cross-validation fold for the evaluation parameter. As a result, the tree being constructed by XGBoost had to continue throughout it's full number of sequences which is set to the default of 100 boosting rounds. The possible effects of this is the overfitting of the model towards the training dataset resulting in high variance on the testing data.

Moreover, the use of the stretched dataset consistently outperformed the dataset that has been scaled by it's aspect ratio. This result is consistent across all instances, with an approximately 10 percentage point difference across results, and does go against the initial hypothesis, which is that stretching would cause image deformation that may result in it being beyond recognition. It is also worth pointing out that the application of standardisation with PCA has also consistently resulted in a lower accuracy score for all configurations of both models. We will therefore rule that running with PCA with the number of components that explains at least 95% of the variance results in a worse result for both Random forest and XGBoost.

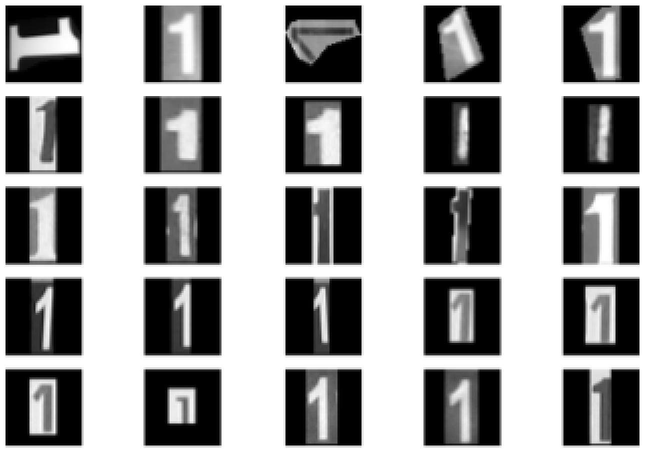
Convolutional Neural Networks

As explained in methods, four main models are built. With the structure from simple to more complex, a gradual increase trend is seen in all metrics. The baseline model with only one convolution layer achieved 0.4524 accuracy. The performance is decent given its simple design. The LeNet-5 model increased all metrics equally by 10 to 20 percent. This model is simple, effective and efficient. The time used is only a half of the time for the baseline model. This is because the published model is carefully designed so that it has less feature maps (6 in the first convolution layer and 12 in the second layer) and a slightly deeper network. The sights are: like expected, a deeper neural network could achieve better results even with less feature maps. The second insight is the number of feature maps in a sequence of convolution and max pooling layer could be designed increasingly without the large increment in time. This makes sense because with one set of convolution and max pooling the size of input shrunk by half as well as the training cost. Then more feature maps could make use of the cost saved.

It is noticeable that evaluation metrics are quite balanced: no apparent skew indicates that all classifiers have appropriate balanced performance in all classes.

CNN classifiers	Accuracy	Precision(macro average)	Recall(macro average)	F1-score(macro average)	Runtime(second)
Baseline model	45.24%	0.4200	0.3000	0.3200	77.0
LeNet-5	63.24%	0.5800	0.5100	0.5200	26.7
Tuned CNN without Data Augmentation	71.97%	0.6800	0.6200	0.6300	169.0 (exclude tuning)
Tuned CNN with Data Augmentation	83.48%	0.8100	0.7800	0.7800	328 (exclude tuning)

With the help of Keras tuner, the effect of stacking convolutional layers before a pooling is proven with the accuracy improved from 0.6324 to 0.7197. Other measures increased by an equal amount of 0.1100. The biggest boost is realised by Data augmentation. Data is set to randomly performing rotation clockwise or anticlockwise by 10 degrees, or zoom in/out by maximum 10%, or shift width/height by maximum 10%. These action ranges are chosen and proven to work well because it describes the variation already seen in datasets (See figure below). When testing with different data augmentation methods, dramatic drops in accuracy are seen when these deformations are not present in the original dataset. For example, it doesn't make sense to flip horizontally and vertically for a dataset containing digits and alphabets and ask the machine to recognise correctly. The accuracy dropped even below the accuracy without any data augmentation. Therefore, the exact usage of data augmentation should be chosen carefully to fit the original data distribution.



Variation in the Original Dataset

Data Augmentation used for Tuned CNN	Accuracy
None	71.97%
Rotation:10 Width shift :0.1 Height shift: 0.1 Zoom range : 0.1	81.23%
Flip: horizontal & vertical Rotation: 10	58.72%
Flip: horizontal Rotation :10	61.20%
Flip: horizontal Rotation :10 Height shift:0.1 Height shift:0.1	63.66%

The tuned CNN with data augmentation is then evaluated on the test set, yielding 82.62% accuracy and macro average precision of 0.78, recall of 0.74, F1-score of 0.75. This is our final result for this task.

Conclusions and Future Works

Overall, our testing for tree ensemble methods has resulted in a win for Random Forest over XGBoost. While XGBoost did obtain an overall higher accuracy score for the configuration using the true-ratio dataset with standardisation and PCA applied, the margin between the classification performance of Random Forest and XGBoost can be seen as materially sufficient to label the bagging ensemble method better suited for this problem over the boosting method. It is often said that the collective opinion will always prevail over a single expert, and this seems to be the case with random forest, while may be considered as a set of relatively weaker decision trees having the power in numbers over a singular tree who has honed its knowledge over multiple iterations.

Additionally, future work and enhancements for Random Forest should increase the hyperparameter range for n_estimators above 250, since we have continually obtained 250 as the best parameter across all configurations. There is good reason to believe that a higher n_estimator, which would result in more trees being created, would further attain a higher score at the expense of computation power required to train the model. Moreover, an element that is yet to be explored is combining both random forest and gradient boosting to create gradient boosted trees. Unfortunately, this has not been covered within the scope of this study, due to the computation power required to create enough parallel trees to provide a generalised gradient boosted forest. With the current setup, the best configuration for tree ensemble is no match for CNN.

Convolutional neural networks have shown great advantages in image classification. With moderate training time and model complexity, highest performance is achieved, in terms of accuracy, precision, recall and F1-scores, with the help of data augmentation. With a vast amount of literature in image recognition using similar datasets, it is usually suggested to adopt a deeper neural network to boost performance. These networks will take tremendous time to tune free parameters but there are many public pre-trained models with adjusted weights available. These models are proven to be transferable to datasets with even different image characteristics (Maitra, Bhattacharya, and Parui 2015, 1021-1025). Another direction of improvement is to solve the problem of insufficient high-quality data. Because the more good quality samples, the better the network could adapt to the features. Images could be hand-cropped to the same dimensions with less unused pixels. Additional samples could be collected manually or artificially generated. Class imbalance should also be taken care of, by maintaining the same number of samples in each class. These could also be done by replicating images in classes with fewer samples.

CNN provides the best accuracy for this classification problem, with HOG + SVM giving approx 74% accuracy. Future work could involve expanding the experimental setup to use multiple feature extraction techniques such as SIFT, Edge extraction, as well as tuning the parameters of HOG. Since SVM allows for kernels to be designed, another direction to take the experiment is to develop a bespoke kernel that may improve the accuracy of the SVM classifier. Another experiment would be to compare the accuracy of the grayscale images (that have been used in this assignment) to that of the original coloured images and how the processing and compute power required varies.

Bibliography

Brownlee, Jason. 2017. “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.” Machine Learning Mastery. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=Adam%20is%20an%20optimization%20algorithm,iterative%20based%20in%20training%20data.&text=The%20algorithm%20is%20called%20Adam,derived%20from%20adaptive%20moment%20est.

Dennanni, Adriano. 2018. “How to deal with image resizing in Deep Learning.” Neuronio.ai. https://medium.com/neuronio/how-to-deal-with-image-resizing-in-deep-learning-e5177fad7d89.

Diaz, Javier, Carlos A. Lopera, Juan David C. Mena, and Lina Quintero. 2019. “The Effect of Color Channel Representations on the Transferability of Convolutional Neural Networks.” *Computer vision conference CVC* 04:10. https://www.researchgate.net/publication/332622912_The_Effect_of_Color_Channel_Representations_on_the_Transferability_of_Convolutional_Neural_Networks.

Evans, David. 2017. “Practical Futures: Chars74k, GANs, and Why Deep Learning Training Sets Matter.” https://withintent.uncorkedstudios.com/chars74k-and-gans-and-training-sets-b3647c676b8e.

Gandhi, Arun. 2018. Data Augmentation | How to use Deep Learning when you have Limited Data — Part 2. https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

Hadidi, Sally. 2016. “Anthony Goldbloom gives you the Secret to winning Kaggle competitions.” KDnuggets. https://www.kdnuggets.com/2016/01/anthony-goldbloom-secret-winning-kaggle-competitions.html.

Horning, Ned. 2010. “Random Forests : An algorithm for image classification and generation of continuous fields data sets.” *International Conference on Geoinformatics for Spatial Infrastructure Development in*

Earth and Allied Sciences n/a (n/a): 1. http://wgrass.media.osaka-cu.ac.jp/gisideas10/viewpaper.php?id=342.

“LeNet-5 – A Classic CNN Architecture.” 2018. https://engmrk.com/lenet-5-a-classic-cnn-architecture/.

Leopd. 2020. “Non-square images for image classification.” Stack Exchange. https://stats.stackexchange.com/questions/240690/non-square-images-for-image-classification.

Maitra, Durjoy S., Ujjwal Bhattacharya, and Swapan K. Parui. 2015. “CNN Based Common Approach to Handwritten Character Recognition of Multiple Scripts.” *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, 1021-1025. 10.1109/ICDAR.2015.7333916.

Tripathy, Auro. 2016. “GoogLeNet Insights.” https://www.slideshare.net/aurot/googlenet-insights.

University of Surrey. 2012. *The Chars74K dataset*. N.p.: The University of Surrey. http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/.

M. Oquab, L. Bottou, I. Laptev and J. Sivic, “Learning and Transferring Mid-level Image Representations Using Convolutional Neural Net- works”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1717-1724, 2014.

Yash Sinha, P. J. a. N. K., 2016. Comparative Study of Preprocessing and Classification Methods in Character Recognition of Natural Scene Images. Pilani: Springer India 2016.

T. E. de Campos, B. R. B. a. M. V., 2009. CHARACTER RECOGNITION IN NATURAL IMAGES. Lisbon: Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP).

Vishnu Sundaresan, J. L., 2015. Recognizing Handwritten Digits and Characters. Stanford: Stanford University.

Weng, L., 2017. Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS. [Online] Available at: *https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html*

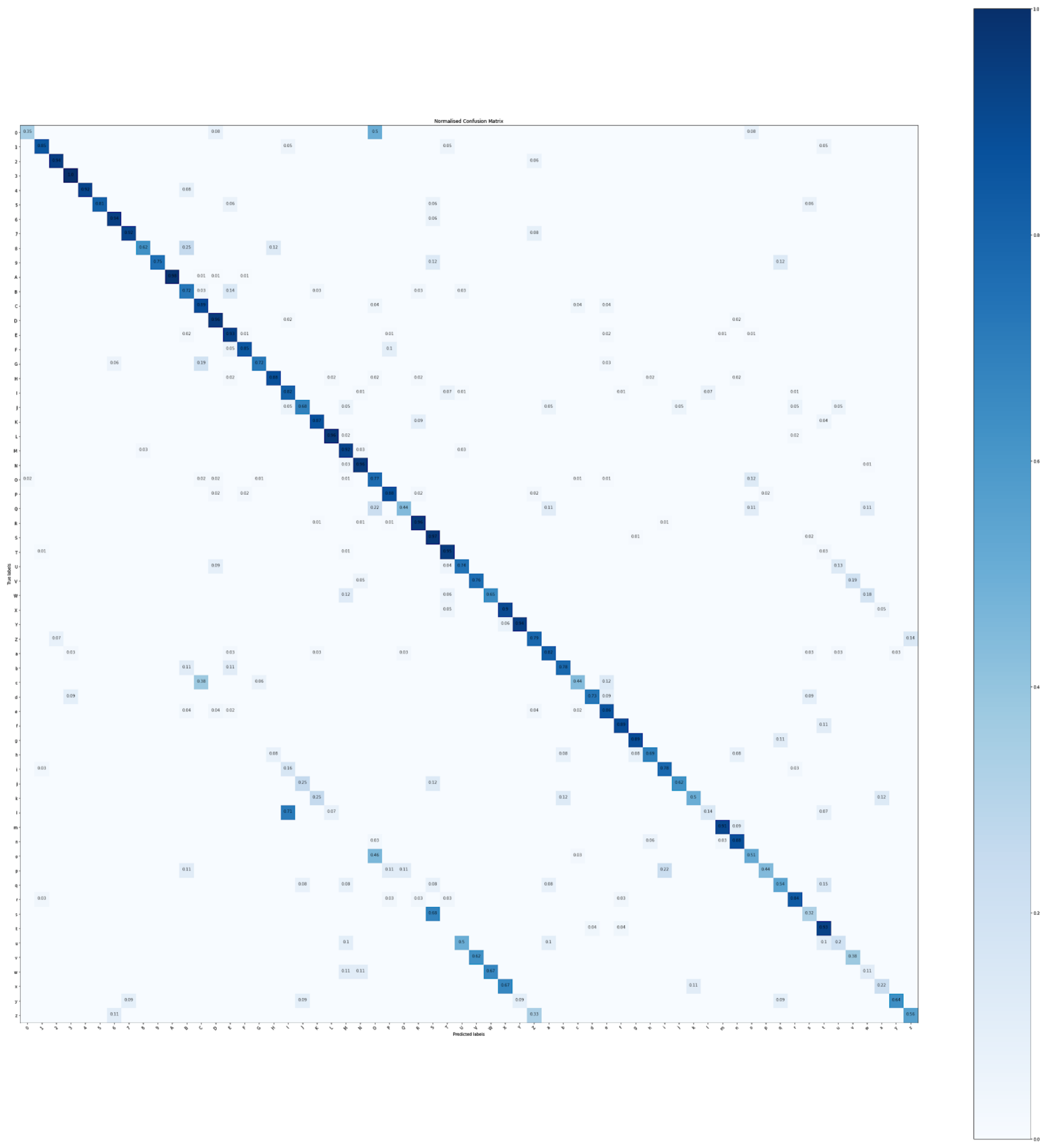
[Accessed 12 November 2020]. SINGH, A., 2019. Feature Engineering for Images: A Valuable Introduction to the HOG Feature Descriptor. [Online] Available at: *https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/*

[Accessed 10 November 2020].Greeshma K V, Dr. J. Viji Gripsy, 2020, Image Classification using HOG and LBP Feature Descriptors with SVM and CNN, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NSDARM – 2020 (Volume 8 – Issue 04)

Appendices

Appendix A: Related Figures

Figure 1. Confusion Matrix for the Best Model (Tuned CNN with Appropriate Data Augmentation)


Normalised Confusion Matrix
True labels
Predicted labels
0.0
0.2
0.4
0.6
0.8
1.0

Appendix B: Contributions

The workload is split according to classifiers. Each person is responsible for his/her part of the report and code file. Joint efforts are made to compile them together. Mimansa Rana used SVM and KNN as her classifiers. Jesse Narvasa wrote the tree ensemble methods. Yuxi Shen is responsible for CNN models.

Appendix C: Instruction to run code

Dataset Download

Dataset should be downloaded here: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/ . Scrolling down to the “Download” section, English dataset with file name “EnglishImg.tgz” (size: 127.9 MB) should be downloaded.

SVM/KNN

The SVM/KNN code was run on Google CoLab. The entire code can be run sequentially, however, in CoLab the data takes over 2 hours to load from a file in Google Drive. Therefore the data was saved in Google Drive as a .npz file that could be read quicker.

Tree Ensembles

The code file was run on Google CoLab. The entire code can be sequentially run, although the INPUT_DIR and OUTPUT_DIR static variables will need to be set accordingly. Please note that the code file contains the model for each four configurations of Random Forest and XGBoost, resulting in eight different models. The best models for Random Forest and XGBoost are mentioned within markup text within the document.

CNN

The CNN code file is run on a desktop with RAM 16GB, CPU: AMD Ryzen 7 3700X 8-Core Processor. The graphic card is NVIDIA GeFore RTX 2060. However, only the CPU is used to run the code. You should run all code chunks sequentially. The whole code file runtime is around half an hour. 90% of the time is used in the CNN tuning stage.