1. Introduction

This project is a blood cell image classification task using machine learning models. In the modern healthcare industry, the importance of integrating artificial intelligence with medical resources has been soaring. This project creates four different machine learning models to classify blood cell images into eight distinct classes, analysing the strengths and weaknesses of the algorithms and comparing their results.
Our aim is to automate and enhance the process of identifying blood cell types, offering significant potential for the medical community. Beyond simple classification, this project involves image recognition, pattern analysis, and accelerates various medical diagnoses, ultimately improving the efficiency and accuracy of haematology disease detection. Moreover, by harnessing the power of machine learning, we not only utilise the model with the highest accuracy but also consider additional factors such as runtime according to the purpose of this classification task, ensuring efficient decision-making.

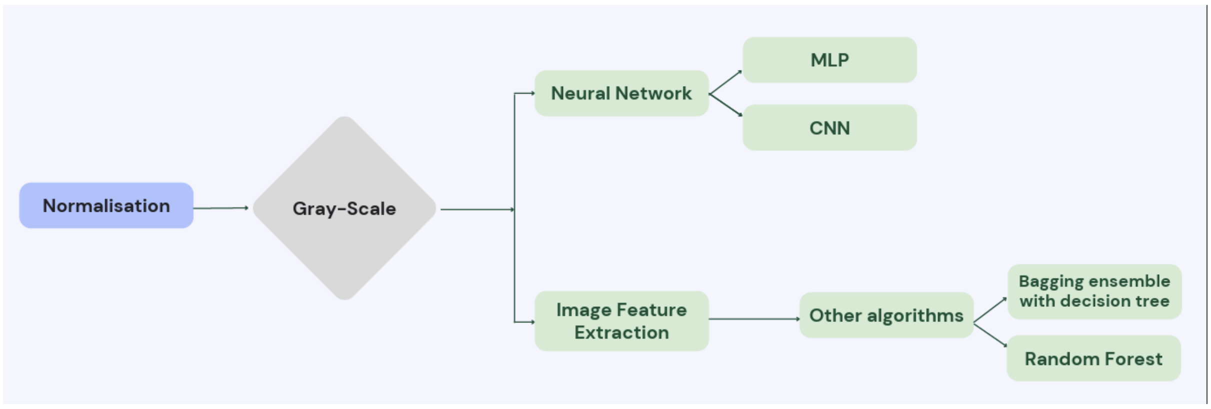


Figure A : Flowchart

2. Data

Dataset Description

In the given dataset, we have 17,092 images in this given dataset which consists of a training set of 13,673 images and a test set of 3,419 images. There are 8 classes in the provided dataset. The original source of the images is derived from BloodMNIST. The images are represented as a 28x28 grid. Each image has 3 channels (Red, Green and Blue) which means there's three values per pixel. From the output of the data type, we can see that each pixel intensity is represented as a byte (unit8) which can be represented by 256 values from 0 to 255. According to the pie chart, it shows corresponding classes of images including labels.

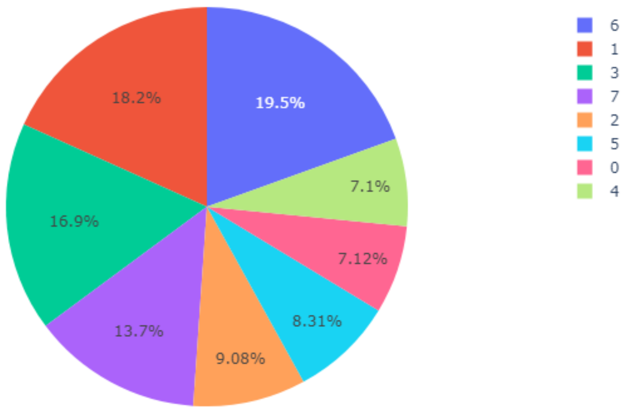


Figure B : Pie Chart

Class 0 corresponds to basophils, accounting for 7.12% of all cell types. Class 1 corresponds to eosinophils, accounting for 18.24%. Class 2 corresponds to erythroblasts, accounting for 9.08%. Class 3 corresponds to immature granulocytes, accounting for 16.94%. Class 4 corresponds to lymphocytes, accounting for 7.10%. Class 5 corresponds to monocytes, accounting for 8.31%. Class 6 corresponds to neutrophils, accounting for 19.48%. Class 7 corresponds to platelets (thrombocytes), accounting for 13.74% (Yang et al., 2023).

Data EDA & Preprocessing

In the initial stage, a thorough review of images representing individual blood cell classes was conducted. This crucial step was accompanied by the essential preprocessing procedure of data normalisation, which is outlined in appendix Figure 1. Throughout this undertaking, image data normalisation process was performed. Upon examining the images for each class, it shows that not only do the sizes differ but also the internal structures of blood cells vary. Given that our primary objective is to identify the distinctive features of blood cells and classify the images, it is critical to gain a thorough understanding of these characteristics. To address this, we conducted grayscale conversion and image feature extraction.

Preprocessing 1: Normalisation

Definition

Normalisation is the technique of adjusting and scaling data to a standard range or distribution between 0 and 1 (Patro & Sahu 2015).

Purpose

Since some pixel values of image data are high and some pixel values are low, we decided to apply normalisation to the image data. We divided images by 255 to make the values normalised between 0 and 1. In this case, this provides equity across all images. It helps in removing variations and ensuring that data is on a common scale for accurate analysis and modelling (Patro & Sahu 2015).

Insight

Based on our examination of the images, it is evident that the images are centred and size-normalised, with uniform dimensions (28x28 pixels).

Preprocessing 2: GrayScale Conversion

Definition

Grayscale conversion is a process of transforming a colour image into a grayscale image, which contains only shades of grey rather than colour.

Purpose

We discovered that the colour had little impact when classifying images, and that the grayscale, which depicts contrast and brightness, was more important. As a result, we chose to transform the photographs from colour to black and white, hence decreasing the pixel count.

Insight

This reduced the complexity of the data, making it easier for machine learning algorithms to process and led to faster training and inference times.

Preprocessing 3: Image Feature Extraction

3.1) Pixel Intensity

Definition

Our initial step involved the assessment of "Pixel Intensity." This represents the level of brightness or darkness of each pixel. Our grayscaled images effectively represent the brightness of grayscale tones in black and white images.

Purpose

This pixel intensity information helps extract features of an image and classify the image.

Insight

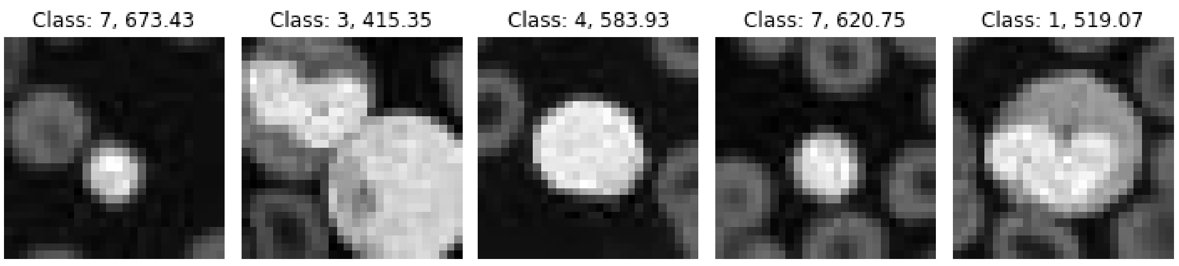


Figure C : Examples of Pixel Intensity

Through this analysis, it became evident that higher pixel intensity values correspond to smaller-sized blood cells.

3.2) GLCM

Definition

The GLCM (Gray-Level Co-occurrence Matrix) technique computes key statistical metrics such as mean and correlation based on the associations between pixel intensities in the current pixel and its neighbours. These calculated values are then assigned as new intensity values for the central pixel within a kernel, representing partial texture properties of the input image. In this project, we employed five key variables, 'correlation,' 'homogeneity,' 'contrast,' 'energy,' and 'dissimilarity,' for describing the texture and patterns within the images (Mall, Singh & Yadav 2019).
Correlation: It measures an association between two pixels.
Homogeneity: It measures the similarity among pixels within a texture.
Contrast: It represents the contrast between neighbouring pixel values.
Energy: It represents the uniformity of pixels.
Dissimilarity: It measures the variation of pixel values.

Purpose

It is used to quantify the spatial relationships of pixel intensities in a grayscale image. GLCM provides statistical information about how often different combinations of pixel values, or grey levels, occur in proximity to each other within an image.

Insight

In the proposed method total, 13 numbers of features were extracted from a BloodMNIST image dataset using a GLCM.

Insights from all preprocessing techniques

In order to compare and analyse the distribution by classes, histogram and density plot were shown in one plot per every variable using a preprocessed dataset. This could lead to gaining insights into the visual differences in distribution among various classes and conducting a comparative assessment of the characteristic distributions for each class. Features like Energy, Homogeneity are highly right-skewed. Features like Size, dissimilarity, contrast, and Correlation is relatively distributed across different values. Therefore, we constructed a dataframe incorporating the variables 'correlation,' 'dissimilarity,' 'contrast,' and 'size,' which demonstrated effective class differentiation. Plots are described in Appendix Figure 2.

3. Methods

Model 1 - Fully Connected Neural Network

Theory:
A fully connected neural network is a neural network where each neuron in one layer is connected to every neuron in the next layer. Each connection has an associated numerical weight. During the training process, the weights are adjusted, in order to learn to predict a correct class (Mahajan, 2020).

Strength:
Flexibility is core of the system, as there are no specific requirements needed on the input data.There are no constraints on the data type and the data format. The input data could be images or text etc.

weakness:
It is time consuming and requires a lot of computational time to run the model due to its complexity of the model. Weaker performance compared to other specialised models such as convolutional neural network which is designed to solve a specific problem.

Architecture:
The fully connected neural network model was created by initialising keras sequential. The input data was flatten and the input size is 28*28 = 784 for each image. There are 784 input neurons in the input layer and 500 neurons in the first hidden layer, this means that there are 784*500=392000 weights along with 500 biases between the input layer and the first hidden layer. The total parameters in this model are 647,008.
For the fully connected neural network, it consists of:
Layers: 4 layers in total including one input layer, two hidden layers and one output layers
Neurons: Input layer has 784 neurons (28*28) for each image.The first hidden layer and the second hidden layer have 500 neurons respectively, and the output layer has 8 neurons for the 8 classes.
Activation Functions: ReLU for hidden layers as it introduces non-linearity into the model and is able to present complex relationships between inputs and outputs. For the output layer, softmax was used as it is appropriate for multi-class classification problems.

Compilation: Adam optimizer and sparse categorical cross entropy loss function for multi-class classification.

Hyperparameters

Fixed parameters:

- **Batch_size:**
  The training data is divided into samples with a batch size of 100. It controls the amount of training data in the training process for each batch. Larger batch size means faster training process as larger data will be processed in each batch. Smaller batch size will lead to higher variance due to the training size in each batch being too small.
- **Epochs:**
  The model is trained 10 times. It controls the number of times the model will be trained. Higher epochs means more opportunities for the model to learn from the training data. Less epochs will prevent the model being overfitting, but might result in underfitting.
- **Number of neurons in the hidden layers:**
  It controls the number of neurons in each hidden layer. The higher the number of neurons in the hidden layers, the more complex the model is. This may lead to overfitting. On the other hand, having too few neurons in the hidden layers, this may lead to underfitting.
- **Activation_function:**
  It is the mathematical functions applied on the neurons in the hidden layers.

Hyperparameters to be tuned:

- **N_hidden_neurons:**
  It is the number of neurons in the hidden layers.
- **Activation function:**
  It is the mathematical function to be applied on the neurons in the hidden layers and determines the output of the neural network. In this case, non-linear functions are more appropriate, so sigmoid and Relu are listed in the hyperparameter tuning section.
- **Optimizer:**
  Different mathematical algorithms to adjust the weights during the training phase. It affects the speed of convergence of gradient descent. For example, when Adam is used as an optimizer, the model reaches its stable point more quickly compared to the basic Stochastic Gradient Descent .

Model 2 - Convolutional Neural Network

Theory:
A Convolutional Neural Network (CNN) is a specialised type of multilayer neural network trained with the backpropagation algorithm. It uses layers of convolutional operations to automatically learn and extract hierarchical features from input images (Seo & Shin 2019).

Strength:
CNNs excel in handling patterns with high variability and are robust to distortions and geometric transformations, making them ideal for applications such as handwritten character recognition, face detection, and image classification.

Weakness:
Data Intensiveness: CNNs require a large amount of labelled training data to perform effectively in order to prevent overfitting (Zhang et al. 2016).

Architecture:
A CNN model using Keras's sequential api was created. This model is designed for a one-dimensional convolutional operation (Conv1D) and is configured as such due to the utilisation of grayscale data.
The model starts with a Conv1D layer with 32 filters, a filter size of 3, and ReLU activation. This layer is suitable for processing one-dimensional data.
A Dropout layer with a dropout rate of 0.5 is added immediately after the first Conv1D layer. Dropout helps prevent overfitting by randomly deactivating some neurons during training.
A Flatten layer is added to convert the output from the convolutional layers into a flat vector for input to the fully connected layers.
A Dense (fully connected) layer with 128 units and ReLU activation is added.
The output layer is a Dense layer with 8 units and a sigmoid activation function. This indicates that the model is designed for multi-class classification with 8 classes.
The model is compiled using the Adam optimizer, 'sparse_categorical_crossentropy' as the loss function, and 'accuracy' as the metric for evaluation. This configuration is suitable for multi-class classification tasks.

Hyperparameters

Fixed parameters:

1. **Convolutional Layer Hyperparameters**:
    - filters in the first Conv1D layer: **32** learnable convolutional filters used to extract features from the input data.
    - filter size: **3** as a filter size of the convolutional filter is specified.
    - activation function: **Rectified Linear Unit (ReLU)** is used as the activation function in the convolutional layer.
2. **Model Compilation Hyperparameters**:
    - optimizer: The **Adam** optimizer is used for gradient-based optimization during training.
    - metric: The model's performance is evaluated using **accuracy** as the metric.
3. **Model Training Hyperparameters**:
    - epochs: The model is trained for a total of **10** epochs, meaning it goes through the entire training dataset 10 times.
    - batch size: During each training iteration, the model is updated using a batch of **100** samples from the training dataset.
    - Validation split: **20%** of the training data is used for validation during training.

Hyperparameters to be Tuned:

- **n_hidden_neurons**:
  It determines the number of neurons in the neural network's hidden layers. It is controlled to increase the model's ability to capture some patterns.
- **activation_function**:
  It affects the degree of how well the model can be learned. It is controlled to find the best function.
- **dropout**:
  It regulates the rate of dropout. It is used to prevent overfitting.
- **optimizer**:
  It adjusts the weights of models in order to minimise the loss. It is controlled to find the best optimisation strategy, which can reduce the training speed and increase the ability to find optimal weight values.

Model 3 - Bagging Ensemble with a Decision Tree

Theory:
Bagging ensemble with a decision tree is an ensemble of decision trees for classification or regression. Bagging means applying decision trees on randomly selected sample subset with replacement and then aggregating the decisions to avoid variance (Chelliah, 2021).

Strengths:
The strengths of the bagging ensemble with a decision tree are**:**
- Reduce variance and better accuracy:
  Since individual decision trees will be applied to different subsets, this results in less overfitting and better generalisation on unseen data.
- Reliable predictions:
  we combine the predictions from multiple trees, therefore, it leads to reliable outcomes.

Weaknesses:
The weaknesses of the bagging ensemble with a decision tree are**:**
- Running Time:
  it requires more time to train the model compared to other algorithms
- Correlated trees:
  If one feature has a higher impact on the target value, all the individual decision trees will be split based on that strongest feature, which means all trees are correlated. This will not reduce the variance.

Architecture:
A decision tree algorithm was applied on the dataset by initialising a decision tree classifier and a bagging ensemble method was used by initialising a bagging classifier, with the decision tree classifier serving as the base estimator. The model was trained using the ttt_all (X training data) and y training data, the ttt_all_test (X test data) was applied to the model to predict the y values, and then the predicted y value was compared with the y test data.
The reason this algorithm is chosen is because it's a classification problem and the decision tree tends to be overfitting. However, by implementing a bagging ensemble with a decision tree, it results in less overfitting and a more generalised model as it combines multiple models based on the randomly selected data from the training set with replacement.

Hyperparameters to be tuned:
- N_estimators:
  it controls the total number of individual trees created. The higher numbers of individual trees created will increase overfitting and lead to higher generalisation error to unseen data.
- Max_features:
  when splitting a node, it controls the maximum number of features the model accepts for each base estimator. A larger number will lead to higher chance of overfitting as the model is more complex.
- Max_samples:
  it controls the proportion of the samples drawn from the training data for each base estimator. This creates randomness on the training data and reduces overfitting.

Model 4 - Random Forest

Theory:
A Random Forest is an ensemble learning method that combines decision trees using feature subsets and majority voting, reducing overfitting and improving model generalisation (Pal 2005).

Strengths:
- High Predictive Accuracy:

It combines decision trees using feature subsets and majority voting, reducing overfitting and improving model generalisation

- Feature Importance Ranking:
  It measures the impact of each feature on performances and rank based on the importance of features in the dataset. This could lead to gaining insights and making decisions.

Weaknesses:

- Computational Complexity:
  When training Random Forest with large amounts of trees and features could require long training time and large memory. This can be less suitable based on the environment.

Architecture:

Random Forest is applied because it is a powerful ensemble algorithm especially for classification tasks. The classifier is trained with hyperparameters and evaluated for accuracy.
Random Forest combines multiple decision trees for predictions. Hence, it could ensure robust performance. Furthermore, it can reduce overfitting effectively by learning via bootstrap sampling and randomly selecting attributes on each node. Finally, it can measure the importance of each attribute, assisting in choosing the best model.

Hyperparameters to be tuned:

- **n_estimators:**
  It is the number of decision trees ensembled. Higher values can lead to better accuracy, so various numbers from low to high are controlled.
- **max_depth:**
  It is the maximum depth of each decision tree that is ensembled. Higher values normally lower the complexity but might increase overfitting. Hence, the limitation of depth is controlled.
- **max_features:**
  It is the maximum number of features considered for each split. Number of features is controlled in a range that prevents overfitting.
- **max_samples:**
  It determines the portion of the dataset that will be used for training. It is controlled not being overfitted.

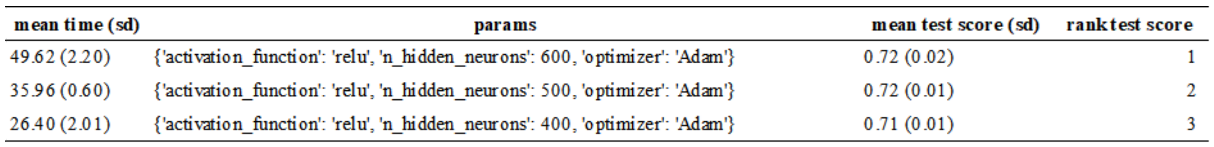
By fixing the random seed, in this project 0, it makes the result of code consistent, allowing to reproduce and verify the validity of the findings. Also, stratified cross validation was used to tune the model.

4. Results & Discussion

Result & Discussion

1. **hyperparameter tuning results**

<Fully Connected Neural Network>



mean time (sd)	params	mean test score (sd)	rank test score
49.62 (2.20)	{'activation_function': 'relu', 'n_hidden_neurons': 600, 'optimizer': 'Adam'}	0.72 (0.02)	1
35.96 (0.60)	{'activation_function': 'relu', 'n_hidden_neurons': 500, 'optimizer': 'Adam'}	0.72 (0.01)	2
26.40 (2.01)	{'activation_function': 'relu', 'n_hidden_neurons': 400, 'optimizer': 'Adam'}	0.71 (0.01)	3

<Convolutional Neural Network>

mean time (sd)	params	mean test score (sd)	rank test score
4.93 (0.53)	{'activation_function': 'relu', 'dropout': 0.1, 'n_hidden_neurons': 300, 'optimizer': 'Adam'}	0.78 (0.01)	1
5.57 (0.38)	{'activation_function': 'relu', 'dropout': 0.1, 'n_hidden_neurons': 200, 'optimizer': 'Adam'}	0.78 (0.01)	2
4.39 (0.18)	{'activation_function': 'relu', 'dropout': 0.1, 'n_hidden_neurons': 100, 'optimizer': 'Adam'}	0.77 (0.00)	3

<Bagging Ensemble with a Decision Tree>

mean time (sd)	params	mean test score (sd)	rank test score
15.67 (0.04)	{'max_features': 1.0, 'max_samples': 1000, 'n_estimators': 800}	0.66 (0.01)	1
7.84 (0.02)	{'max_features': 1.0, 'max_samples': 1000, 'n_estimators': 400}	0.65 (0.01)	2
11.77 (0.05)	{'max_features': 1.0, 'max_samples': 1000, 'n_estimators': 600}	0.65 (0.01)	3

<Random Forest>

mean time (sd)	params	mean test score (sd)	rank test score
5.25 (0.01)	{'max_depth': 15, 'max_features': 'sqrt', 'max_samples': 0.8, 'n_estimators': 300}	0.66 (0.01)	1
5.26 (0.01)	{'max_depth': 15, 'max_features': 'log2', 'max_samples': 0.8, 'n_estimators': 300}	0.66 (0.01)	1
28.16 (0.04)	{'max_depth': 15, 'max_features': None, 'max_samples': 0.9, 'n_estimators': 400}	0.66 (0.01)	3

2. The trends and explanations for observations

Heatmaps were plotted to show the patterns of the mean test score between the selected two hyperparameters. The darker the colour is in the heatmap, the higher the mean test score is in relation to the combinations of the two selected hyperparameters. This is shown in Appendix Figure 3.

Model 1: Fully Connected Neural Network

From the heatmap (n_hidden_neurons vs optimizer):
It indicates that using Adam as the optimizer and higher number of hidden neurons tend to have higher mean test scores. The model performs better with mean test score 0.72 when the number of hidden neurons is 600 and the optimizer is Adam.

From the heatmap (optimizer vs activation function):
It indicates that the model performs best with mean test score 0.71 when the optimizer is Adam and the activation function is Sigmoid.

From the heatmap (activation function vs n_hidden_neurons):
It indicates that the mean test score starts increasing when the number of hidden neurons gets larger and using Relu as the activation function leads to higher score. The score starts to drop when the hidden number of neurons is greater than 600. The model performs best with mean test score 0.61 when the activation function is Relu and n_hidden neurons are between 400 and 500.

Model 2: Convolutional Neural Network

From the heatmap (n_hidden_neurons vs activation function):
The findings indicate that when using activation functions such as ReLU and sigmoid, a higher number of hidden neurons tends to lead to better results. Specifically, the model performed best with a mean test score of 0.59 when ReLU was used as the activation function, and 300 hidden neurons were approximately 300.

From the heatmap (n_hidden_neurons vs dropout):
Regardless of the dropout rate, it showed that a higher number of hidden neurons leads to an increase in the mean test score. Among the results, the model performed best with a mean test score of 0.51 when the dropout rate was set at 0.1, and approximately 300 hidden neurons were used.

From the heatmap (n_hidden_neurons vs optimizer):
The findings suggest that using Adam or RMSprop as the optimizer tends to perform better results compared to Adadelta or SGD. Overall, a higher number of hidden neurons generally leads to improved results. Specifically, when employing the Adam optimizer and having over 100 hidden neurons, the model achieved its best performance with a mean test score of approximately 0.66.

From the heatmap (activation function vs dropout):
The findings indicate that using ReLU or sigmoid as the activation function tends to result in better performance compared to softmax, especially when the dropout rate is low. The best result, with a mean test score of 0.6, was achieved when utilising a dropout rate of 0.1 in combination with the ReLU activation function.

From the heatmap (activation function vs optimizer):
The findings demonstrate that, overall, applying ReLU and sigmoid as activation functions yielded favourable results. Furthermore, combining them with the Adam or RMSprop optimizer led to relatively superior performance. Among these results, the highest mean test score reached 0.75 when both the Adam optimizer and the ReLU activation function were employed.

From the heatmap (dropout vs optimizer):
Upon examining the overall trend, it became evident that applying the Adam and RMSprop optimizers, along with low dropout rates, consistently resulted in favourable outcomes. Notably, the combination of Adam and a 0.1 dropout rate yielded the highest mean test score, reaching 0.67.

Considering the comprehensive analysis of all the results, the trend shows that the combination of ReLU as the activation function and Adam as the optimizer, along with a lower dropout rate and a higher number of hidden neurons, consistently leads to better test results.

Model 3: Bagging Ensemble with a Decision Tree

From the heatmap (max_samples vs n_estimator):
It indicates that the model has the same performance when max_samples is in a range between 500 and 1000 and n_estimator is between 400 and 800.

From the heatmap (max_samples vs max_features):
It indicates that higher max_features and higher max_samples will have better performance. The model performs the best when max_features is 1.0 and max_sample is 1000.

From the heatmap (n_estimator vs max_features):
It indicates that higher max_features tend to have higher mean test score, and the model has the best performance with mean test score 0.65 when max_features is 1.0 and n_estimators are between 400 and 800.

Based on the result, the results are aligned with the predictors as we can see that the mean test score gets to a point when it stops increasing. The higher the values of the hyperparameters we set does not mean that the model will perform the best. This is because higher values of hyperparameters may lead to overfitting and not generalise to the unseen data. Low values lead to underfitting that the model will not learn enough from the training data. Therefore, the results aligned with the predictions.

Model 4: Random Forest

From the heatmap (n_estimator vs max_depth):

Regardless of the n_estimators value, larger max_depth sizes consistently led to better results. The best outcome was achieved with the largest max_depth value within the specified range when n_estimators was set to 200, showing an accuracy score of 0.51.

From the heatmap (n_estimator vs max_features):
After performing Grid Search with the values of both variables, all combinations had the same values. This means that the model's performance is not affected by variations in these parameters.

From the heatmap (n_estimator vs max_samples):
When max_samples are 0.8 and 0.9, the mean test scores are all identical regardless of the values of n_estimators. Furthermore, it showed very low scores when 1.0 max_samples are applied.

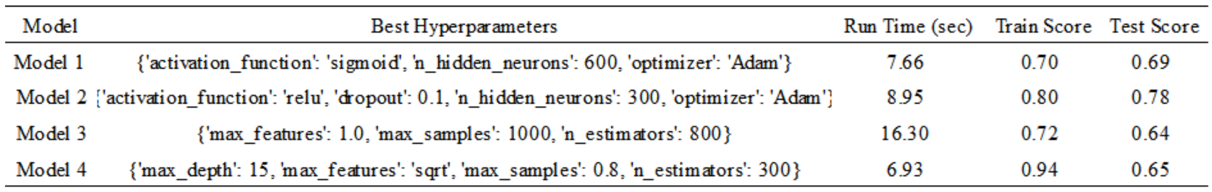
From the heatmap (max_depth vs max_features):
The larger max_depth value is, the higher score mean test scores perform. However, the difference is not prominently noticeable compared to the other values.

From the heatmap (max_depth vs max_features):
When max_depth values increase, smaller max_smples consistently lead to better results. The best outcomes were achieved with the largest max_depth value within the specified range when max_samples were set to 0.8 and 0.9, showing an accuracy score of 0.66.

From the heatmap (max_features vs max_samples):
When max_samples are 0.8 and 0.9, the mean test scores are all identical regardless of the type of max_features. Furthermore, it showed very low scores when 1.0 max_samples are applied.

Considering the comprehensive analysis of all the results, the trend suggests that max depth and max_samples appear to have a more significant impact on the outcome compared to n_estimators or max_features. A lower max_samples and a higher max_depth, consistently leads to better test results.

3. a comparison of the results for the four different models with their best hyperparameters



Model	Best Hyperparameters	Run Time (sec)	Train Score	Test Score
Model 1	{'activation_function': 'sigmoid', 'n_hidden_neurons': 600, 'optimizer': 'Adam'}	7.66	0.70	0.69
Model 2	{'activation_function': 'relu', 'dropout': 0.1, 'n_hidden_neurons': 300, 'optimizer': 'Adam'}	8.95	0.80	0.78
Model 3	{'max_features': 1.0, 'max_samples': 1000, 'n_estimators': 800}	16.30	0.72	0.64
Model 4	{'max_depth': 15, 'max_features': 'sqrt', 'max_samples': 0.8, 'n_estimators': 300}	6.93	0.94	0.65

4. Analysis and discussion of the results

Model 1 has the highest mean test score (0.72) with Adam optimizer, sigmoid activation function and 600 hidden neurons. Adam's higher convergence speed in gradient descent compared to others and sigmoid is suitable for the binary classification problem aligned with our expectation.
In comparison to model 2, model 1 has high computational cost and lower precision, F1 score and recall values for each class (0-7) because the convolutional neural network is more specialised and efficient for image classification problems. For example, for class 7, the precision of model 1 is 0.97 whereas model 2 has precision 0.99.
Model 3 bagging ensemble with a decision tree has the highest mean test score (0.66) with max_features: 1.0, max_samples: 1000, n_estimators: 800. In comparison with model 4 random forest, model 3 has a slightly lower accuracy test score 0.641 compared to 0.649, and lower precision, recall and F1 score. For example, for class 7, precision for model 3 is 0.98 whereas 1 for model 4. This result aligned with our expectation as model 3 has less randomness on selected sample subsets (with replacement), which makes model 4 outperforms model 3.

Overall, the convolutional neural network performs the best amongst all 4 models that were implemented on the provided dataset. CNN can be applied where accuracy is more important than other aspects such as runtime.
In situations where accuracy is paramount, such as disease diagnosis or patient monitoring, applying a Convolutional Neural Network (CNN) model is a viable choice. However, for data analysis in medical fields beyond direct life-critical decision-making, a different approach may be suitable. In these cases, a Fully Connected Neural Networkcan be employed, offering a trade-off with slightly lower accuracy but faster runtime.

5. CONCLUSION

Summary of main findings and identification of study limitations

In this study, we implemented four machine learning algorithms for the blood cell image classification task and we have found the following findings:

- Fully connected neural network: mean test score 0.72. It has high flexibility on input data and is able to handle different data types. However, its tradeoff is high computational cost.
- Convolutional neural network: mean test score 0.78. It performs outstanding on image classification problems due to its different architecture design of neurons.
- Bagging ensemble with a decision tree: mean test score 0.66. It has less overfitting and better generalisation on unseen data.
- Random forest: mean test score 0.67. It’s an ensemble method of decision trees. It has relatively low runtime compared to other models.

The limitations of this study are:

- Data: the study is working on a relatively not large size of sample data. Having a more representative and larger sample data with different class distribution would have a more reliable and unbiased model performance result.
- Computational cost: fully connected neural network and convolutional neural network takes much longer time to run the model.

Future work suggestions

- Select a more balanced and representative data of classes.
- Exploring other types of algorithms such as ensemble methods due to its much lower computational cost.
- Investigate more on the hyperparameter tuning part to find the best hyperparameters.

Considering the above suggestions will enhance the accuracy and efficiency of our algorithms.

6. REFLECTION

Our takeaways were comprehending the end to end process of machine learning, from processing the data processing and implementing various algorithms to hyperparameters tuning and finally having the final model. In addition,we now had a better understanding of the strengths and weaknesses of different algorithms. Overall, completing this assignment provides a great experience on applying machine learning theories into real-world applications.

7. References

Chelliah, I. (2021, September 6). *Bagging Decision Trees — Clearly Explained*. Medium. https://towardsdatascience.com/bagging-decision-trees-clearly-explained-57d4d19ed2d3

Mahajan, P. (2020, October 23). *Fully Connected vs Convolutional Neural Networks*. Medium. https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5

Mall, P. K., Singh, P. K., & Yadav, D. (2019, December 1). *GLCM Based Feature Extraction and Medical X-RAY Image Classification using Machine Learning Techniques*. IEEE Xplore. https://doi.org/10.1109/CICT48419.2019.9066263

Pal, M. (2005). Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, *26*(1), 217–222. https://doi.org/10.1080/01431160412331269698

Patro, S. G. K., & Sahu, K. K. (2015). Normalization: A Preprocessing Stage. *ArXiv:1503.06462 [Cs]*. https://arxiv.org/abs/1503.06462

Seo, Y., & Shin, K. (2019). Hierarchical convolutional neural networks for fashion image classification. *Expert Systems with Applications*, *116*, 328–339. https://doi.org/10.1016/j.eswa.2018.09.022

Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., & Ni, B. (2023). MedMNIST v2 - A large-scale lightweight benchmark for 2D and 3D biomedical image classification. *Scientific Data*, *10*(1), 41. https://doi.org/10.1038/s41597-022-01721-8

Zhang, L., Yang, F., Daniel Zhang, Y., & Zhu, Y. J. (2016, September 1). *Road crack detection using deep convolutional neural network*. IEEE Xplore. https://doi.org/10.1109/ICIP.2016.7533052

8. Appendix

Appendix: Supplementary Figures

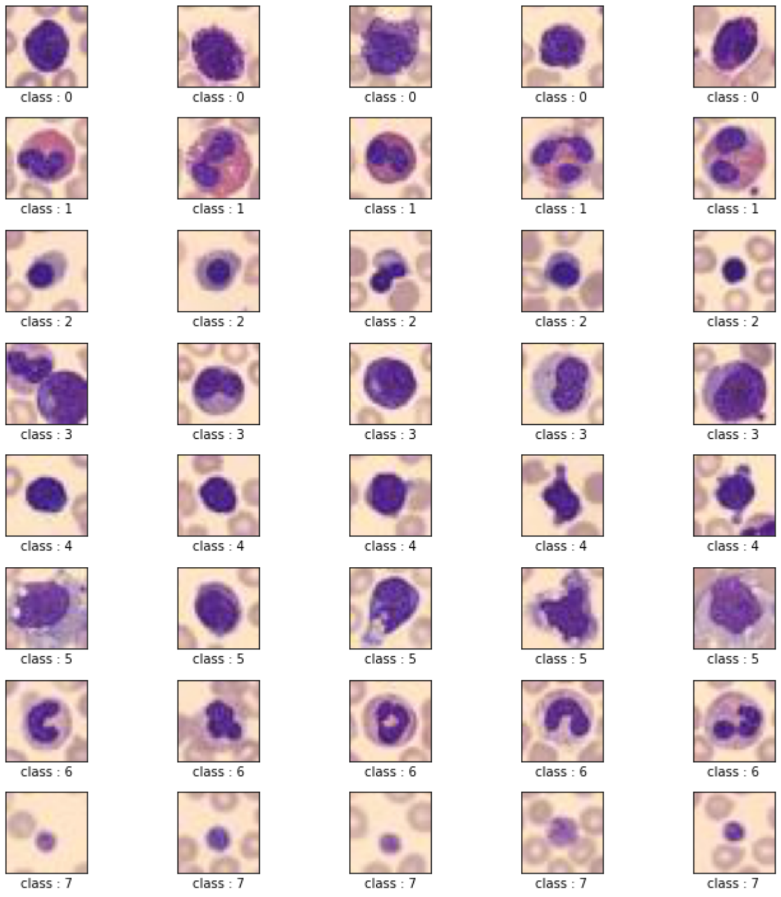


Figure 1 : Blood Cell Image by Class

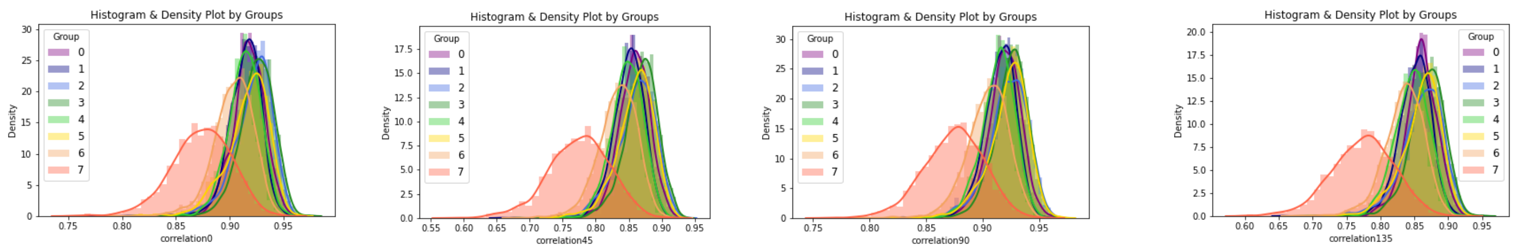


Figure 2.1 : Correlation Density Plot & Histogram

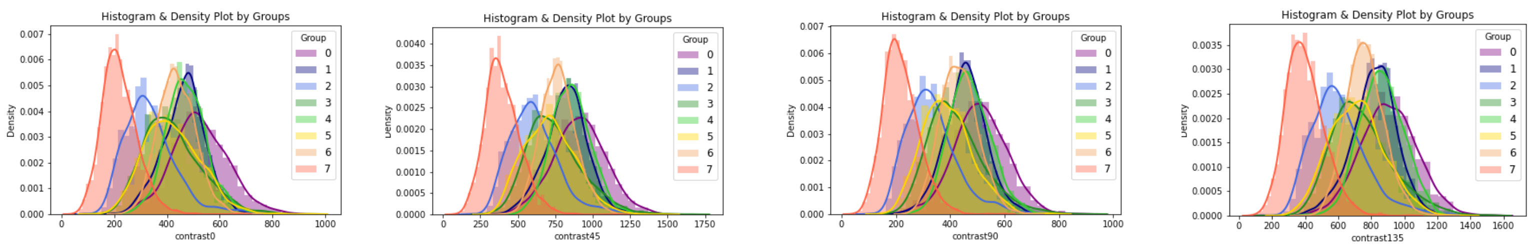


Figure 2.2 : Contrast Density Plot & Histogram

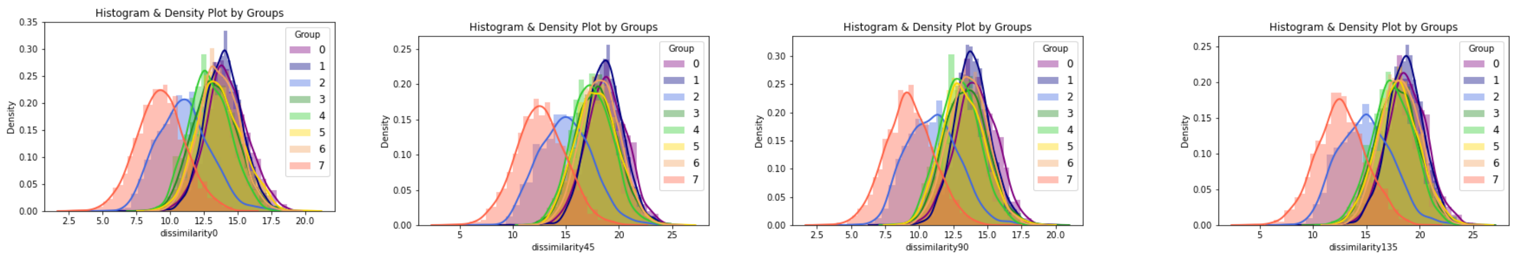


Figure 2.3 : Dissimilarity Density Plot & Histogram

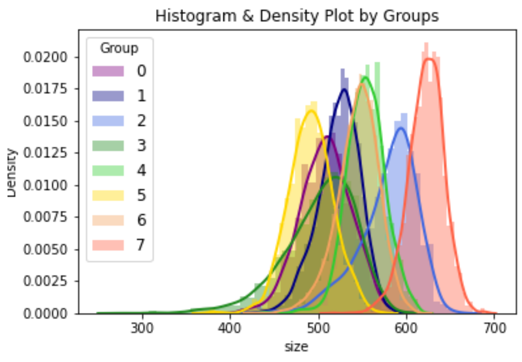


Figure 2.4 : Size Density Plot & Histogram

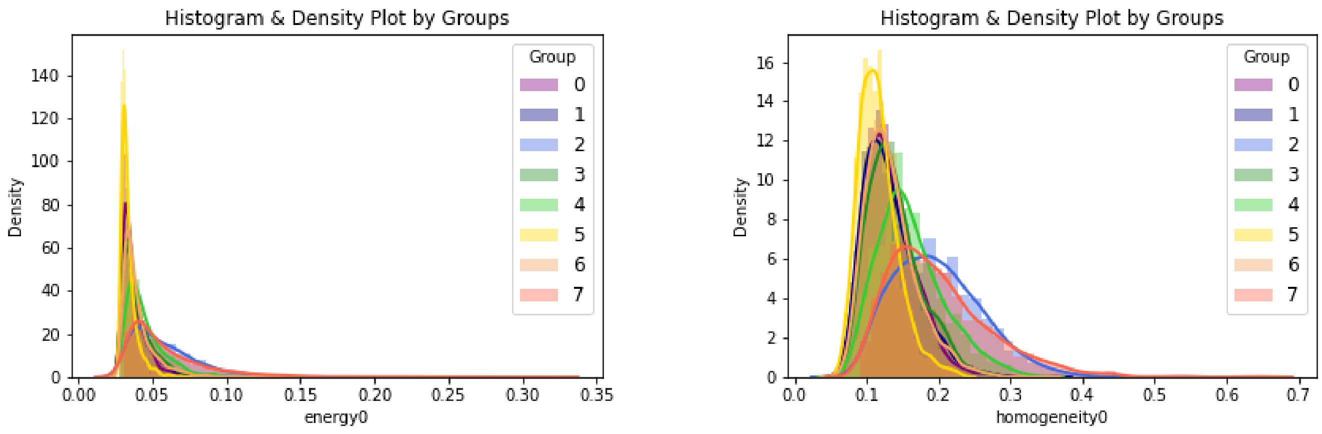


Figure 2.5 : Energy & Homogeneity Density Plot & Histogram : Reason not selected

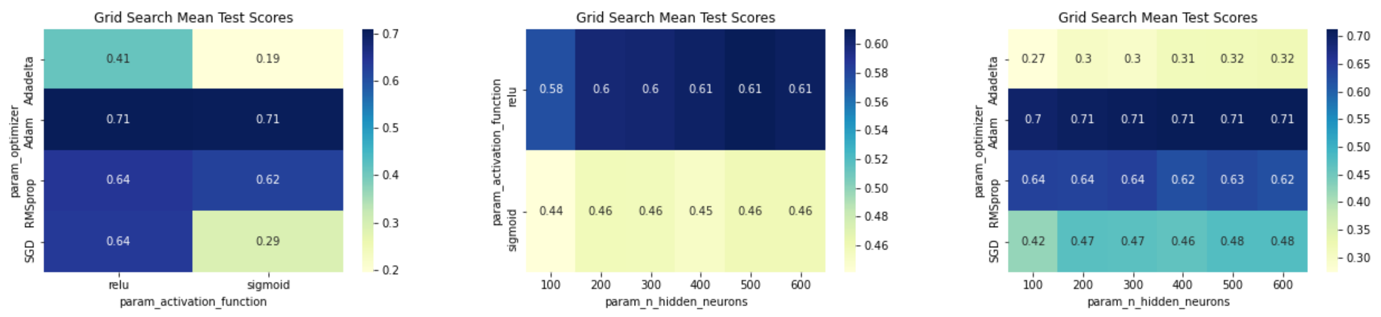


Figure 3.1 : Fully Connected Neural Network Heatmap

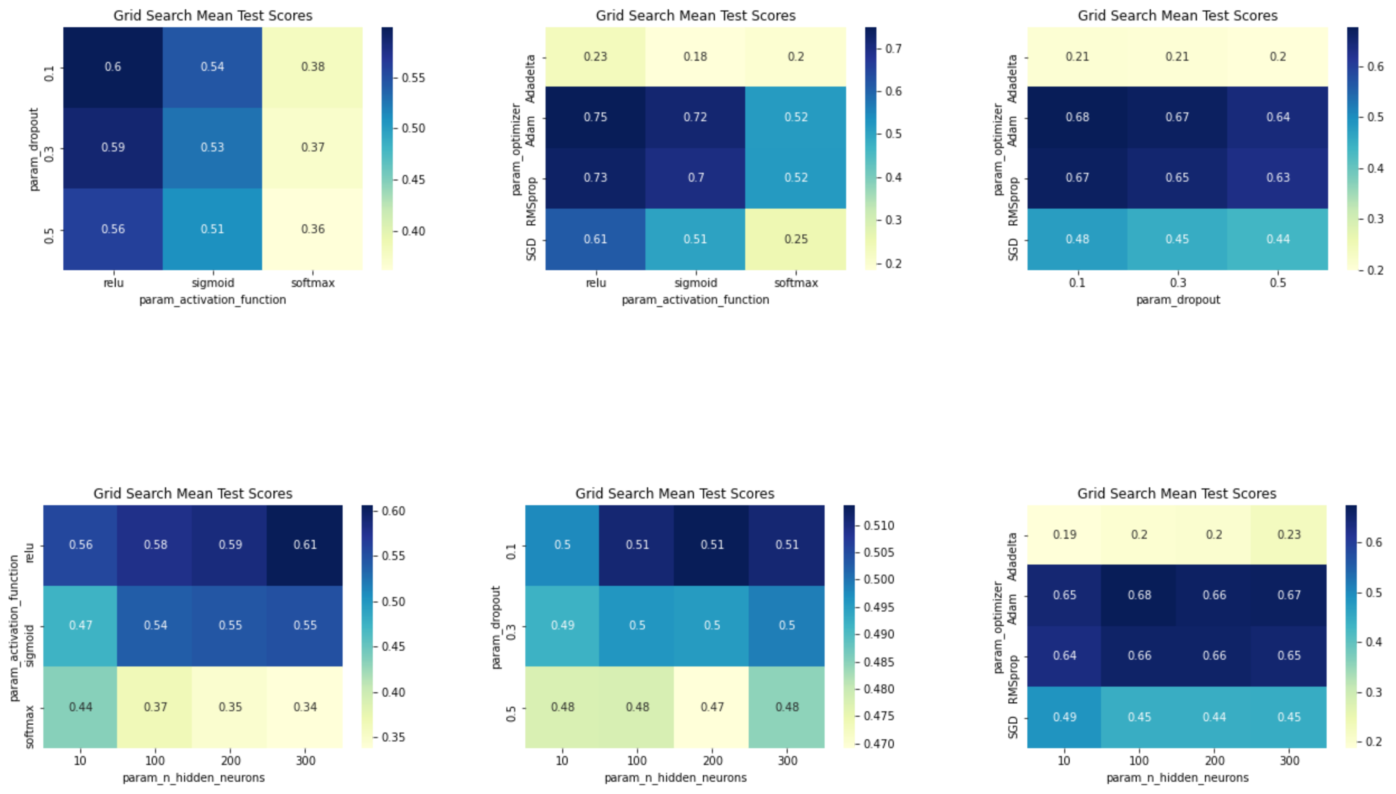


Figure 3.2 : Convolutional Neural Network Heatmap

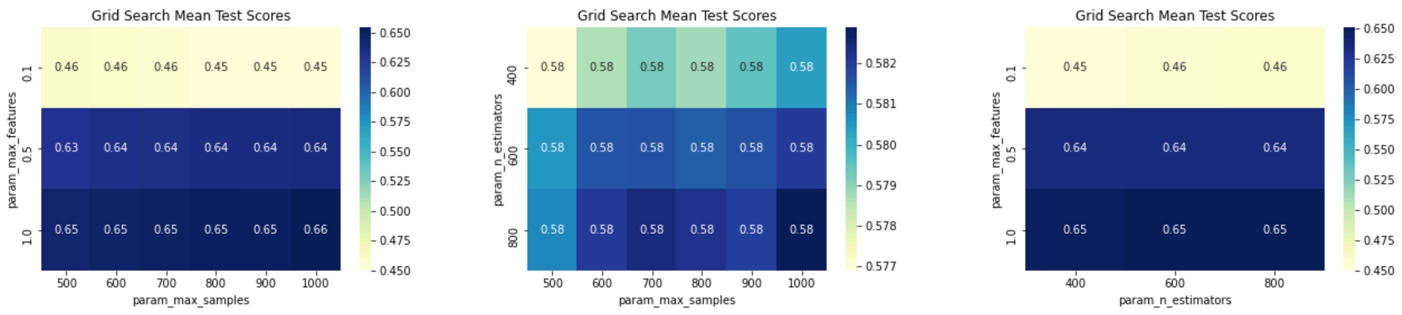


Figure 3.3 : Bagging Ensemble with a Decision Tree Heatmap

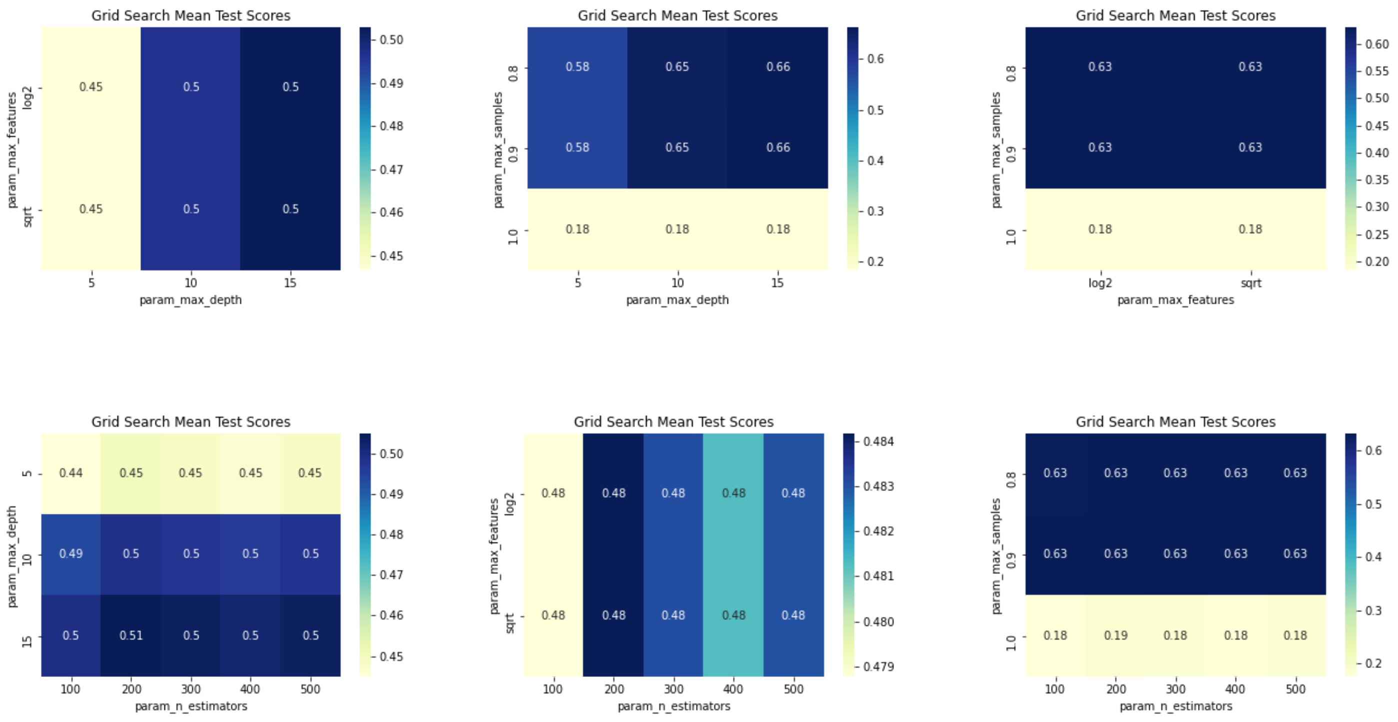
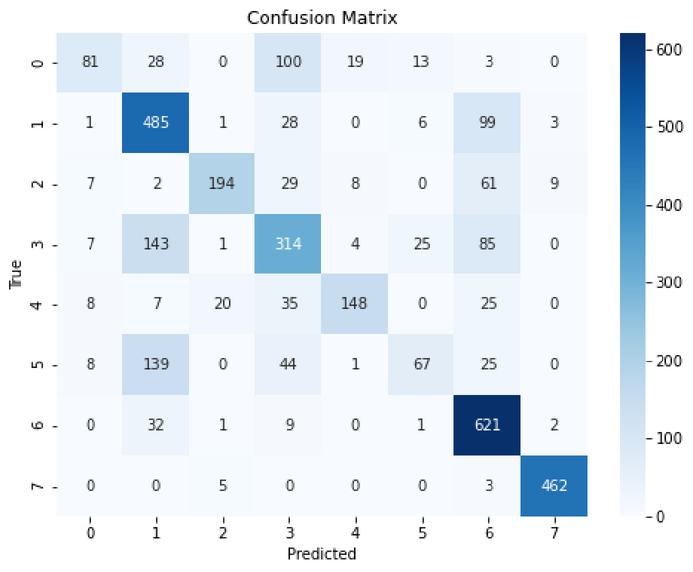


Figure 3.4 : Random Forest Heatmap



True \ Predicted	0	1	2	3	4	5	6	7
0	81	28	0	100	19	13	3	0
1	1	485	1	28	0	6	99	3
2	7	2	194	29	8	0	61	9
3	7	143	1	314	4	25	85	0
4	8	7	20	35	148	0	25	0
5	8	139	0	44	1	67	25	0
6	0	32	1	9	0	1	621	2
7	0	0	5	0	0	0	3	462

Figure 4.1 : Final Fully Connected Neural Network Confusion Matrix

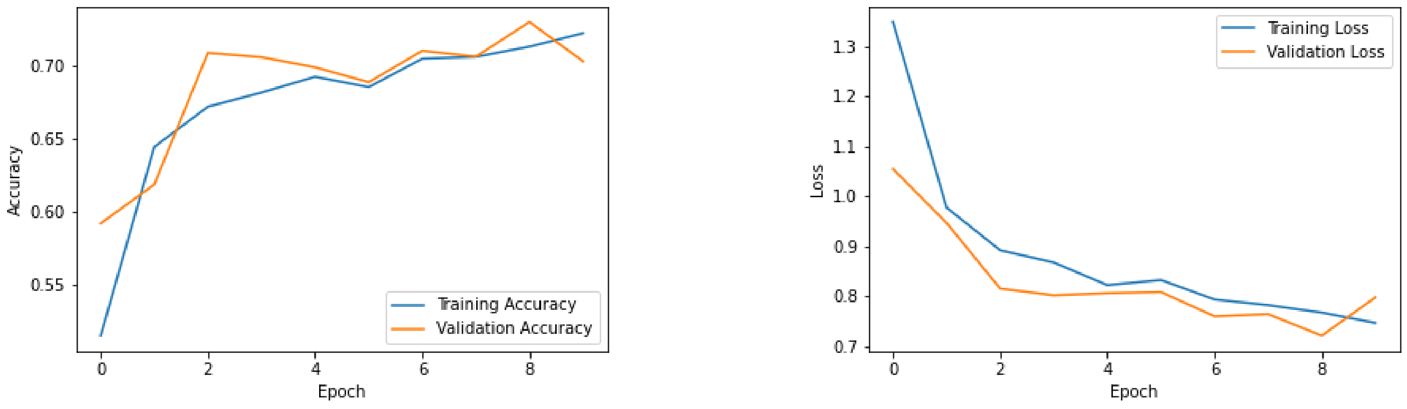


Figure 4.2 : Final Fully Connected Neural Network Training/Validation Accuracy & Loss Graph

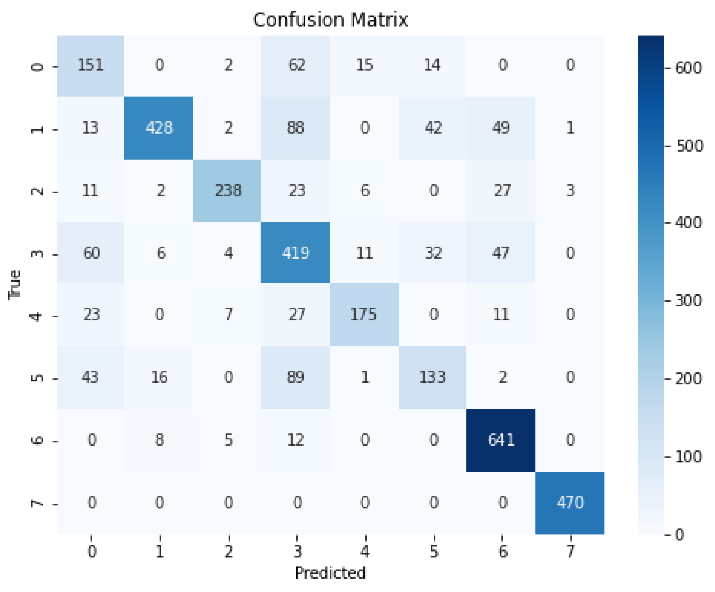


Figure 4.3 : Final Convolutional Neural Network Confusion Matrix

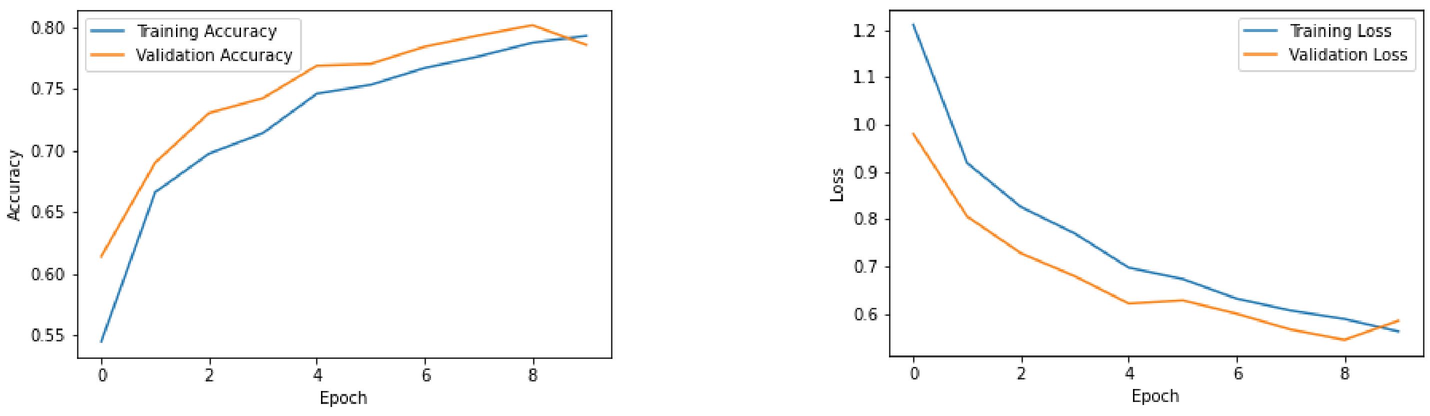


Figure 4.4 : Final Convolutional Neural Network Training/Validation Accuracy & Loss Graph

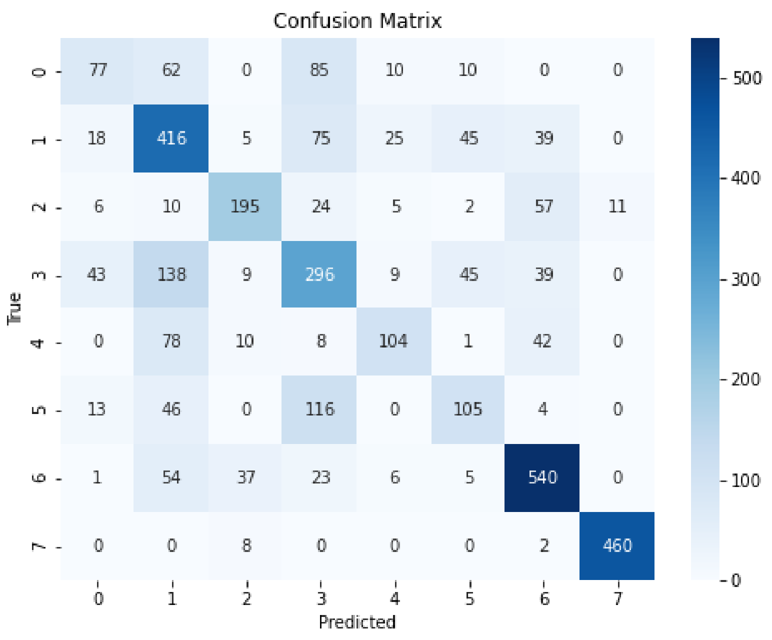


Figure 4.5 : Final Bagging Ensemble with a Decision Tree Confusion Matrix

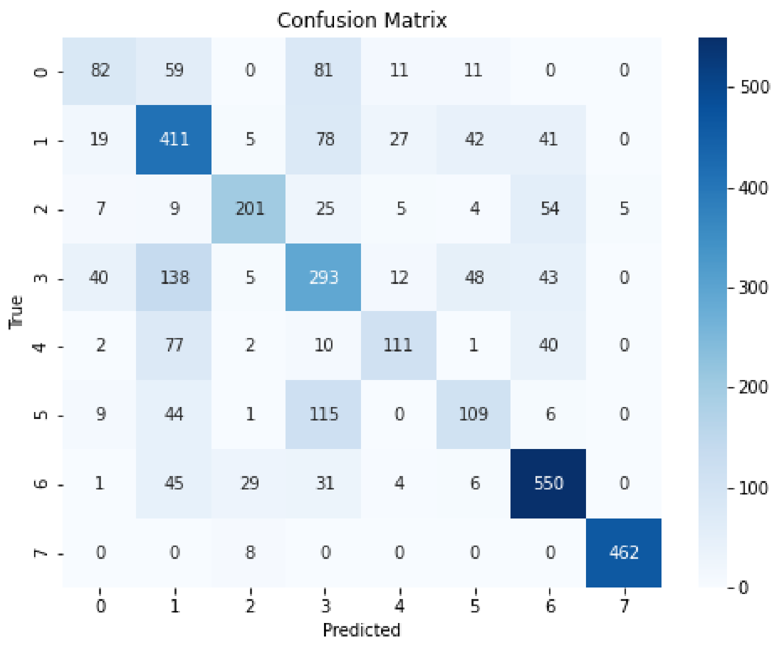


Figure 4.5 : Final Random Forest Confusion Matrix