Table of Contents

1 Introduction

In this report, we examine the performance of four machine learning algorithms on the BloodMNIST dataset. The BloodMNIST dataset [1] contains a total of 17092 color images of individual peripheral blood cells captured on a blood film, which is then organized into eight different classes in accordance with the eight categorizations of the peripheral blood cell [2].

The analysis and classification of peripheral blood cells is done by medical doctors to assess a patient's condition [3]. This analysis provides insight not only into the patient's overall conditions, but also assesses how well a specific bodily function performs, and how well a treatment is working on the patient [3]. However, these tests can only be performed in highly specialized laboratories, and it exposes the performing specialist to hazards and infections [3].

Looking at the importance of the classification task in the medical field, we are planning to experiment on four machine learning algorithms: Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Random Forest, and Support Vector Machine (SVM). We will tune the best parameters for each model and compare the classification accuracy of those four algorithms using accuracy and F1-score metrics. We hope that the outcome of this report can contribute not only to the automation of blood analysis, but also to the analysis of machine learning performance in image classification tasks.

2 Data

The original dataset from block [2] contains 17.092 images with the dimensions 360×363 of .JPEG format, but for this report, we derive the processed version which is called the BloodMNIST dataset from block [1]. This version of the dataset contains the same number of images that have been cropped to center each cell(s) of interest. Additionally, each image's dimension has been reduced to 28x28, and the image format has been changed to .NPY, which further helps to reduce computational cost.

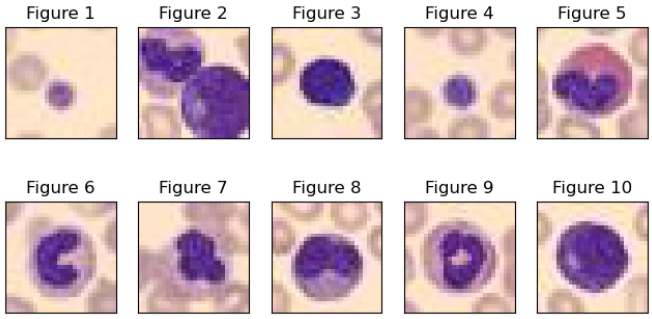

Figure 1: A brief look at the first ten entries on the BloodMNIST dataset.

The BloodMNIST dataset consists of two sets: a training dataset and a testing dataset. Each dataset is further split into two, with each set containing an input data (X) and a label data (Y) of which we will predict our model's outcome against. Simply, we end up with the following four datasets: X (training data), Y (training data), X (testing data), and Y (testing data). The training subset of the data makes up 80% of the data (13,673) while the testing subset makes up 20% of the data (3,419).

The X data is a four-dimensional array with the following structure: (number of images, image dimension, image dimension, number of color channel). We can see in Figure 1 that when plotted, X data consists of the actual images of each cell(s) of interest.

The Y data on the other hand is a one-dimensional array of integers which values range from 0 to 7. Each value signifies which class the cell(s) in each image belongs to. These labels were manually given to each image by clinical pathologists when the original dataset was first created [2]. From block [2], we can also note that the label distribution in the BloodMNIST dataset is not uniform, with four of the eight labels representing an average of 17% of the dataset and the other four representing an average of 8% of the dataset.

Table 1: shows the label that corresponds to each cell type.

Cell Type (Class)	Label
Neutrophil	0
Eosinophil	1
Basophil	2
Lymphocyte	3
Monocyte	4
Immature Granulocyte	5
Erythroblasts	6
Platelets	7

2.1 Preprocessing

There are a few preprocessing techniques that we can use for images. The provider of the BloodMNIST dataset has already resized, cropped, and centered the images from the original dataset. However, we will additionally apply the following techniques to further reduce computational costs and provide better insight for our machine learning algorithms.

2.1.1 Image Augmentation

Recall that in Figure 1, we can immediately infer that the images in BloodMNIST dataset are uniform in terms of color, contrast, and placement of each cell of interest. By introducing variations in our dataset, we can reduce the likelihood of our models overfitting or over-relying on irrelevant features in the dataset. Image augmentation is a technique to synthetically add variations and disturbances to a dataset, and in most cases, it also adds more synthetic samples to the dataset. In our experiment, we will not be adding new samples to prevent high computational cost.

We will be performing the following augmentations to our dataset: image flipping, adjusting of image contrast, adjusting of image brightness, adjusting of image hue and saturation values, image rotation. An external library called Albumentations [11] will be used to perform the augmentations.

2.1.2 Normalization

Normalization is the process of reducing the range of an image's pixel intensity value. We use a straightforward method of dividing all pixel values by 255 to reduce the image's pixel intensity range from [0, 255] to [0, 1]. By placing the values in a much smaller range, we can reduce the computational costs of our machine learning algorithms. This is a common approach especially for deep neural networks, as computation of high numeric values may become complex.

2.1.3 Image Reshaping

As previously mentioned, the data X is a four-dimensional array. However, the python library we are using for our random forest and support vector machine models only accept two-dimensional arrays for the input data. Specifically for these two models, we will reshape the structure of X using the numpy python library with the following calculation:

$$New\ shape = (X, Y), with:$$

$$X = number\ of\ samples$$

$$Y = number\ of\ color\ channels \times image\ height \times image\ width$$

It is important to note that with every step of preprocessing, there always exists a risk that we are omitting potentially important information as we transform our input data. Many image preprocessing methods include converting its full-color image datasets to a greyscale one to reduce computational costs. However, there is not enough evidence to suggest that color images would significantly impact the performance of a machine learning algorithm [5] [6]. Therefore, to preserve as much information from the dataset, we have decided to keep the BloodMNIST dataset as color images in our experiment.

3 Methods

In this experiment, we will focus on four machine learning algorithms: Fully Connected Neural Network (MLP), Convolutional Neural Network (CNN), Random Forest Ensemble, and Support Vector Machine (SVM). At the end of this section, we will describe the strength and weakness of these four algorithms compared to each other, and the architecture and hyperparameters of each algorithm.

3.1 Fully Connected Neural Network (MLP)

Multilayer Perceptron (MLP) is one of the most popular artificial neural networks. This network consists of one input layer, at least one hidden layer, and one output layer. These hidden layers help perceptron to solve non-linear classification problems, which will be passed to the output layer that determines the final classification.

MLP works based on the minimization of errors between the predicted outputs and the expected outputs. To achieve this result, MLP utilizes supervised training in its algorithm. One of the simplest and most general methods used for supervised training is backpropagation, which uses the gradient descent method to adjust the weights of the network to minimize the errors.

This method can be divided into two steps: feedforward and backpropagation. In the feedforward step, input patterns are applied to the input layer and their effect propagates through the hidden layers until an output is defined. After that, the errors will be calculated according to the predicted and expected outputs. These errors then were transmitted backwards from the output layer through the hidden layers back to the input layer while it updates the network weights. This process is repeated to reduce the network error.

3.2 Convolutional Neural Network (CNN)

Convolutional neural network (CNN) is a deep learning technique derived from the traditional MLP. Instead of using fully connected layers like its predecessor, CNN uses special network

structure, alternating between convolution and subsampling layers. Blocks of CNN consist of convolutional layer, pooling layer, and ending in fully connected layer.

Convolution layer is the core layer of CNN. This layer applies several different image filters known as image kernel to the inputs. The filters might extract features like the edge object or the color that differentiate each class of images. This kernel starts from top left of an image and move to the left to right and top to bottom, pixel by pixel, applying a mathematical operation at each coordinate of the image. Depending on the image number of channels (1 for greyscale, and 3 for RGB), the image inputs are divided into matrices. In this layer, we also apply non-linear activation function the same as MLP.

In pooling layer, we reduce the size of the matrices acquired from the previous convolution layer. This is done to avoid overfitting due to large dimensionality, especially in large dataset that requires numerous filters to find the pattern. The fully connected layer connects every neuron in one layer to every neuron in other layer. The concept is similar to MLP, where flattened matrices go through the fully connected layer to predict the images' labels.

3.3 Ensemble Method: Random Forest

An ensemble method refers to the combination of a few machine learning algorithms to predict a certain outcome. This method is often preferred as the combined model tends to have lower error rate than a singular machine learning model, under the condition that each algorithm in the ensemble has an error rate of lower than 50%. The outcome of the method is decided by majority vote (classification task) or average value (regression task), but there are other techniques that can be used as well. In our report, we will be using the Random Forest ensemble. Let us explore each algorithm in the ensemble:

Decision Tree

Appropriately, a Random Forest algorithm is made up of numerous Decision Trees. A decision tree is essentially a sequence of tests, where an algorithm will try to find an optimal way to divide the input data at each test, to determine which category the input data belongs to. How the splitting is decided depends on the chosen splitting criteria.

There are three components that make up a decision tree: non-leaf nodes, branches, and leaf nodes. A non-leaf node represents the test, a branch represents an attribute value, and a leaf node represents a class.

Bagging

Bootstrap aggregation, also known as bagging, is a method where a certain number of samples are randomly sampled from the original training set and fitted to the same number of machine learning models. The bagging method is a way to create a machine learning model that is less sensitive to outliers and overfitting as its outcome stems from the majority voting of predictions in its ensemble.

The random forest algorithm tends to be a good starting point for most machine learning problems as it runs at a reasonable time, has a decent performance rate, and perhaps most importantly, a decision tree's logical sequence is very easy to interpret. Drawing a graph for the first decision tree of the random forest, we found out that each image seems to be separable at certain pixel

values on the two-dimensional image axis. From this information, we can try using the Support Vector Machine for our next algorithm to better group each feature.

3.4 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm which premise is to find the best hyperplane that can separate the input data into its intended categories without occurring a significant overfit. In other words, it aims to separate the data with a hyperplane that has the maximum distance between the two categories of data. This hyperplane is computed through a kernel function.

Sometimes, data cannot be separated directly when it is two dimensional. Depending on its kernel function, the SVM is able to project the input data into a higher dimension to find the best hyperplane that can separate these input data. However, computing all the higher dimensions the SVM has generated would be computationally expensive. Instead of explicitly transforming the data into the higher dimension, the kernel function computes the similarity between pairs of points in the projected feature space. This method is called the kernel trick.

The decision to use a SVM was mainly led by the fact that the decision trees in the previous random forest ensemble had found that it is possible to categorize the BloodMNIST training set through specific values in the dataset. We were hoping that the SVM model would be able to perform these separations better since it is built to do such tasks.

3.5 Evaluation Metrics

In order to decide the best model for this dataset, we will perform evaluation to our algorithms. We will be using two metrics: accuracy and F1 Score.

3.5.1 Accuracy

Accuracy is a basic metric that measures the ratio of correctly classified data to the total number of images. This is a useful metric to understand how well the model is performing overall. However, this might not be a good metric when faced with imbalanced dataset, where one label outnumbered the other by large margins.

3.5.2 F1 Score

One metric that is useful to use in an unbalanced distribution of labels is the F1 score. F1 score is calculated from precision (ratio of true positives over false positives) and recall (ratio of true positives over false negative). It is meant to symmetrically represent both precision and recall into one metric and represents exactly how precise our models are in predicting the correct labels for each case.

3.6 Algorithms Comparison: Strength and Weakness

In this section, we will compare all four algorithms in terms of their robustness, tendency to overfit, training time, model complexity, and classification accuracy.

Table 2: Theoretical Algorithms Comparison

	MLP	CNN	Random Forest	SVM
Robustness; Tendency to overfit	Depends on the dataset and problem. Their fully connected architecture makes it more susceptible to variations of input. However, deep learning techniques aim to minimize prediction errors and are more likely to be influenced by noise or outliers' data and overfit.	Depends on the dataset and problem. It is often more robust to spatial variations. However, deep learning techniques aim to minimize prediction errors and are more likely to be influenced by noise or outliers' data and overfit.	It is relatively robust to noise due to the random sampling of bootstrap aggregating. However, without intervention through parameter tuning, it is very easy for a random forest algorithm to reach near-perfect accuracy on its given training set.	The robustness of SVM cannot be easily generalized as its robustness against noise heavily depends on its regularization parameters and the chosen kernel function.
Training time; Model complexity	Faster to train compared to CNN because it has simpler architecture and fewer parameters. However, it still has more parameters than the other two algorithms.	The CNN model has more complex architecture (involves many matrices operations) and more parameters to tune, requiring more time to fit and has a more complex model than three other algorithms.	Very cheap to run compared to the other three models. Hyperparameters are easily visualized compared to the abstract parameters of the neural networks and SVM.	The training time of the SVM model increases exponentially along with the given number of training samples. Although it only has three significant parameters, the SVM model is not easily visualized due to how it often projects to much higher dimensions.
Classification accuracy	Only take flattened vector input. This can be crucial for datasets with important spatial information. It can outperform random forest or SVM when faced with complex data that requires feature learning.	Exceptional performance in image-related tasks due to their ability to take spatial information.	Good performance can be expected given that the testing set has similar features to the training set.	Although the SVM model is often said to perform well in classification tasks, its accuracy is heavily dependent on the quality of the training data and its model's hyperparameters.

3.7 Architecture and Hyperparameters

There are more than three parameters to tune in each algorithm. However, we have time limitation and unable to test all the parameters, therefore we are going to choose 3 parameters from each algorithm to tune that we think make a significant difference to the results.

Table 3: Architecture and Hyperparameters in Each Algorithm

MLP	CNN	Random Forest	SVM
Activation function. Choosing this function can be crucial for MLP model since higher degree polynomial functions can cause overfitting while simpler functions can cause underfitting, depending on how complex our dataset is. There are several functions used in MLP, but we will choose the most used functions: sigmoid function, hyperbolic tangent function, and rectified linear unit.	**Number of layers.** Small number of layers will have limited capacity to capture intricate patterns in the data. This leads the model to underfit the training data, only capturing simple features such as edges or simple textures. On the other hand, a large number of layers can capture more complex features. However, a high number of layers might result in overfitting, as they might mistakenly take noise as feature in training data.	**Criterion.** The splitting criteria for each test in the decision tree. Generally, a decision tree will want to perform a splitting that results in highest purity or least variations for each split. This is preferable as it means that the test has perfectly split the data into two distinct groups. There are then three ways to measure the purity of each data split: entropy, Gini index, and log loss.	**Kernel.** How a SVM's decision boundary is constructed depends on its kernel function. For our experiments, we will only be considering the most well-known kernel functions: linear, polynomial, and radial-basis kernel functions. Only polynomial and radial-basis functions are able to project input data into a higher feature space.
Optimizer's learning rate. Smaller learning rate typically results in more stable training process and help the model converge to a better local minimum, however, if it is too small, the model might get stuck in local minimum, and vice versa.	**Number of filters.** Filters in this algorithm are responsible for capturing features in the inputs. The expected effect on the model is similar to number of layers parameter.	**Maximum leaf nodes.** The maximum amount of leaf nodes. It is possible for a decision tree to find multiple ways to categorize a class in a single decision tree. Limiting the maximum leaf node would encourage overfitting to occur less.	**C.** Determines how harsh the margin in the hyperplane is in punishing misclassification. The higher the value of C, the less strict the decision boundary.
Number of hidden neurons. The number of hidden neurons in each layer signifies the depth of our model. Less neurons make it more likely for our model to underfit the training data but has a faster training time and vice versa.	**Batch size.** This parameter signifies the number of data samples used in each forward and backward propagation during the training process. A higher batch size will have faster convergence and faster training process. However, high batch size might lead to overfitting.	**Maximum depth.** The maximum amount of splitting allowed to occur on a single branch of the decision tree. Limiting the depth of a decision tree also helps to lessen the chance of overfitting.	**Gamma.** This variable only applies to non-linear kernel functions. The gamma variable determines the smoothness of the decision boundary, with a lower gamma value signifying a smoother curve in the decision boundary.

In addition to hyperparameter tuning, we are going to use early stopping from Keras library for MLP and CNN to help us avoid overfitting. With the maximum epochs of 100, we will set the early stopping patience to 5 epochs, meaning that if the validation loss of the model doesn't significantly improve for 5 epochs it will automatically stop the fitting.

Additionally, for the SVM model, we will be utilizing a low number of maximum iterations to reduce computational costs. The maximum iteration is a variable that determines how many iterations the SVM model is allowed to attempt to reach convergence in its decision boundary until it is forcibly stopped. Another way to implement a premature stopping criterion is to utilize a higher tolerance coefficient.

4 Results and Discussion

In this section, we will report our results for each algorithm, including its average accuracy score and average fit time during hyperparameter tuning, as well as the results of each algorithm with their best parameters using accuracy and F1-score metrics with the help of scikit-learn library.

We will be using scikit-learn library's grid search cross validation function to perform our parameter tuning. The grid search function works by exhaustively applying each parameter combination on the machine learning algorithm model, and then cross validating the results with each other. For our experiment, we will be performing three sets of cross validation for each parameter combination.

4.1 Fully Connected Neural Network (MLP)

After the hyperparameter tuning we've done using grid search CV, we get that MLP model using activation function tanh with 300 hidden neurons and learning rate of 0.1 give us the best accuracy result, with average accuracy of 66% and average fit time of 23.8831.

Figure 2: MLP Hyperparameter Tuning Results. Grey bar chart represents the average accuracy and parameters with highest average accuracy is highlighted in red. The blue line represents the average fit time. Further details can be found in notebook since this is an interactive visualization.

As seen from the results, the tanh activation function consistently produces better average results compared to other activation functions, indicating that our dataset's patterns align well with the characteristics of the tanh function. Notably, there's no significant difference in fitting time for the three activation functions.

On the other hand, the number of hidden neurons significantly impacts the fitting time. For instance, 100 hidden neurons result in an average fitting time of 11 seconds, whereas 200 hidden

neurons lead to an average fitting time of 17 seconds, and 300 hidden neurons increase the average fitting time to 23 seconds. However, this increase in model complexity doesn't yield a significant boost in accuracy, implying that we have reached a point of diminishing returns, and the dataset may not require an overly complex model to capture its hidden patterns.

The most influential factor affecting accuracy is the optimizer's learning rate. Particularly for the sigmoid and tanh activation functions, a learning rate of 0.1 consistently outperforms lower learning rates such as 0.001 and 0.0001. This suggests that a larger learning rate benefits the model's convergence and helps in navigating the loss more efficiently.

4.2 Convolutional Neural Network (CNN)

The parameters with the highest accuracy score in our CNN model are 2 layers, 16 filters, and 128 batch size, with average accuracy of 78% and average fit time of 36.214.

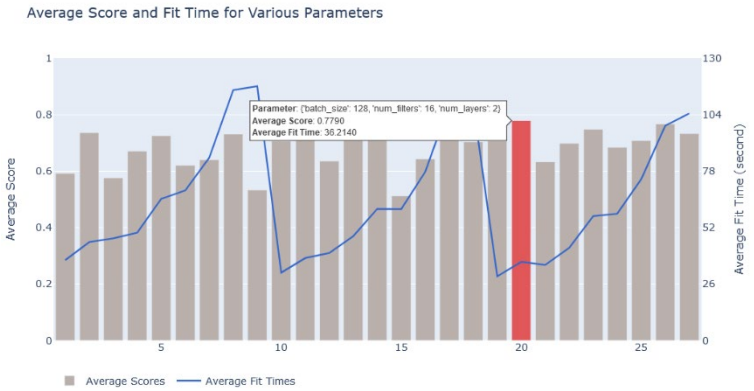

Figure 3: CNN Hyperparameter Tuning Results. Grey bar chart represents the average accuracy and parameters with highest average accuracy is highlighted in red. The blue line represents the average fit time. Further details can be found in notebook since this is an interactive visualization.

As seen in the results, both number of layers and number of filters affect the fit time as expected. Out of the number of layers we tune, 2 layers give us the best accuracy result. This indicates that 2 layers strike an optimal balance for our dataset. More layers give us a decrease in accuracy, making it less effective especially looking at how it increases the fit time significantly.

On the other hand, we don't achieve any significant boost in accuracy when using a larger number of filters. At several instances, they even result in a decrease in accuracy. This indicates that the dataset may not need an overly complex model to capture its patterns.

In addition to offering faster training time, larger batch size gives us a higher accuracy score than the other. Large batch size gives us a more stable convergence and consistency.

4.3 Ensemble Method: Random Forest

We can observe that the grid search function has determined the following parameters to result in the best accuracy of 75%: splitting criterion of Gini index, maximum tree depth of 25, and number of decision trees in the forest to be 125.

We can see from the interactive graph in the Jupyter Notebook that a low number of estimators in the ensemble consistently result in lower average accuracy score and fitting time. Generally, we would like to have a lot of estimators in our ensemble to have a robust machine learning model.

Given how surprisingly similar these results are, we have to consider if the loss of efficiency is worth the miniscule advantage on the average accuracy score. Additionally, we also must consider if the similarity in accuracy scores is because random forest ensemble tends to be very good at adapting to its training set.

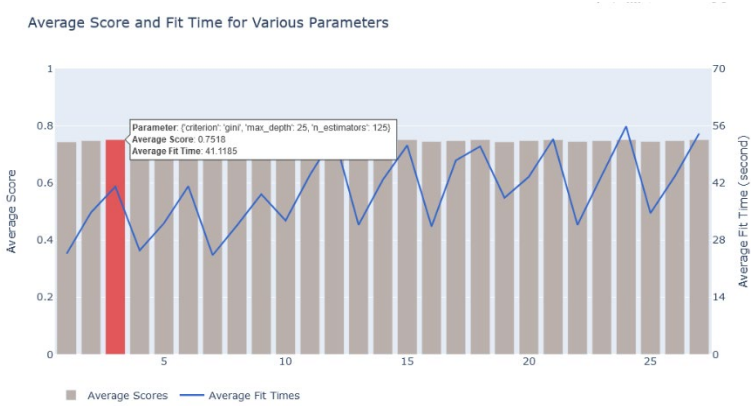

Figure 4: Random Forest Ensemble Hyperparameter Tuning Results. The height of each bar chart represents its average accuracy, with the red bar highlighting the best average accuracy score. Each bar will show its hyperparameter combination when hovered. The blue line represents average fit time. Please refer to the Jupyter Notebook for this interactive graph.

Curiously, splitting criterion doesn't seem to significantly affect the accuracy of the ensemble. The maximum tree depth doesn't seem to have a significant impact on the ensemble, either. With the best maximum tree depth of 25, this seems to further support that limiting the growth of decision trees works very well to avoid overfitting.

4.4 Support Vector Machine (SVM)

Our grid search reached 74% average accuracy with the following combination of hyperparameters: C at 5, Gamma value of scale (very small number of one divided by number of features and variance of input data), and the radial-basis kernel function.

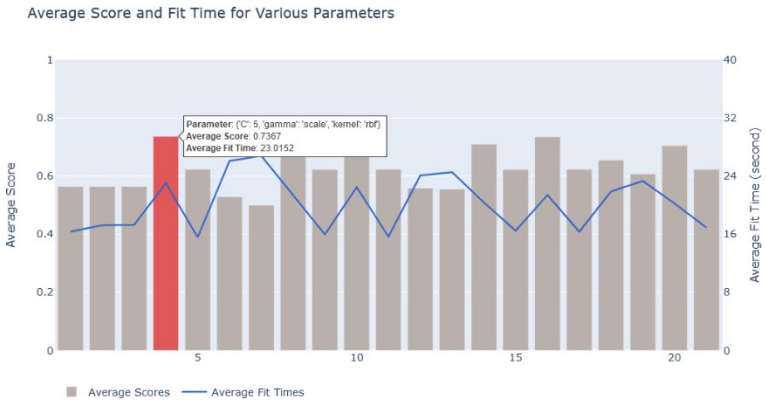

Figure 5: Support Vector Machine Hyperparameter Tuning Results. The height of each bar chart represents its average accuracy, with the red bar highlighting the best average accuracy score. Each bar will show its hyperparameter combination when hovered. The blue line represents average fit time. Please refer to the Jupyter Notebook for this interactive graph.

In the interactive graph, we can see that the radial-basis function consistently performs the best amongst the three kernel functions. We can roughly deduce that the distribution of our data is not linear, considering the low scores of the linear function. We initially expected that the SVM model would be very sensitive to its C and Gamma values, but as there is no discernible pattern that we have found on the graph, more investigation would be needed to see the effect these two variables. The polynomial kernel seems to perform well with high values of C, however.

4.5 Algorithms Comparison

Table 4: Algorithms Results on Testing Data

Algorithm	Hyperparameters	Accuracy	F1-Score	Fit Time (second)
MLP	Activation function: tanh; Hidden neurons: 300; Learning rate: 0.1	0.611	0.5688	29.8802
CNN	Batch size: 128; filters: 16; layers: 2	0.7961	0.7418	148.8725
Random Forest	Criterion: gini; Max depth: 25; Estimators: 125	0.7646	0.7086	88.2153
SVM	C: 5; Gamma: scale; Kernel: rbf	0.7175	0.6957	40.6656

Neural network algorithms such as MLP and CNN are more sensitive to noise and outliers. Since we were doing augmentation (adding noise) to our dataset, MLP got lower accuracy and F1 score despite being a neural network algorithm. On the contrary, the random forest ensemble did relatively well when used to predict the clean testing data. This supports the theory that random forest models are said to be quite robust to noise and outliers due to their bagging method. In the end, CNN has the highest accuracy out of four algorithms; our result further proves that CNN has an outstanding performance for image dataset because of its ability to take spatial information.

In this experiment, our SVM model appears to have performed quite poorly considering its reputation to be well-suited for classification tasks. We have a theory as to why this is the case. We found out that the fitting time and the accuracy of a SVM model is dependent not only on its C and Gamma parameters, but also on how many iterations it is allowed to reach convergence in its decision boundary. We have chosen the maximum iterations number to be 250, as anything higher than this is computationally very expensive and not feasible for us. Considering that the maximum iteration the SVM is allowed by default is infinite, and that other variations on the SVM in scikit-learn library allow the value of the maximum iteration to be 1000 or more, we suspect that our chosen maximum iteration number may be very low. If this is true, it would explain our model's comparatively poor performance.

In terms of fitting time, neural networks generally have higher fit time due to their supervised learning method and complex architecture, as they have more parameters. The fitting time will vary depending on how many epochs the models are doing. Since we are using Keras early stopping, the number of learning is done according to the validation loss in order to avoid overfitting the training data.

On an additional note, the random forest ensemble performs surprisingly well despite the simplicity of its algorithm. From its properties, we can infer that if the random forest gets to cover types of noises in its training samples, it would be able to perform very well.

At the end, we did a smaller experiment with only 5 epochs on neural networks about how they work on non-augmented dataset (can be seen in notebook's appendix), and they unsurprisingly have higher accuracy. This further proves how neural networks don't work well with noise and outliers in the data. However, larger experiments are needed to further study this phenomenon.

5 Conclusion

This study provides valuable insights to the performance of different machine learning algorithms, particularly in the context of image classification. The findings underscore the remarkable performance of CNN over the other 3 algorithms, achieving accuracy score close to 80% and F1 score of 74%. Despite the augmentation preprocessing leading to decrease in accuracy for neural network algorithms, it still outperforms other algorithms due to its ability to take spatial information. However, it is worth noting that its slow training time remains a consideration. Random Forest algorithm gives us close score to CNN, having the accuracy of 76% and F1-Score of 71% with almost half the training time. This might be a better choice for image classification tasks that requires smaller computational time.

While our study focused on a limited set of hyperparameters, it is important to remember that there are more parameters that define a machine learning algorithm. However, due to the time and hardware limitations, we couldn't truly conduct a comprehensive and exhaustive hyperparameter tuning for all algorithms, which might offer valuable insights and further enhance the model's performance.

In the future, we propose conducting larger experiments by combining both augmented and non-augmented dataset, to expand the scope and gain a deeper understanding of how these algorithms performances are and to see whether the trends observed in this study persist in a broader range of data or not. In addition, we can conduct more experiments with pre-made CNN with more complex architectures such as AlexNet, VGGNet, GoogLeNet, and many more, and see how the complexity impacts the predictions accuracy.

In real-world applications, where datasets are filled with all kinds of noise, the importance of data quality enhancement such as image denoising cannot be overstated. This is especially evident in critical domains like biomedical, where even a minor misclassification can result in significant consequences.

6 Reflection

Doing this assignment has given us the following insights. We learned the importance of hyperparameters and how each parameter affects each algorithm model. For example, we've seen how important an epoch is to MLP and CNN; we can directly observe how accuracy improves the more epoch is allowed for these algorithms. More complex parameters or algorithms don't necessarily give us a better result and might even decrease the prediction's accuracy.

In addition, we've also learned how important it is to be familiar with an algorithm's structure and theoretical foundations to make a sound decision in which parameters to tune. For example, it's quite beneficial to be familiar with a Support Vector Machine's kernel functions to determine which ones to include and how one would go on about regularizing them.

7 References

[1] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, B. Ni. MedMNIST v2 - A large-scale lightweight benchmark for 2D and 3D biomedical image classification. In: Scientific Data (2023).

[2] A. Acevedo, A. Merino, S. Alférez, A. Molina, L. Boldú, J. Rodellar. A dataset of microscopic peripheral blood cell images for development of automatic recognition systems. In: Data in Brief (2020).

[3] R. Al-qudah, C. Y. Suen. Improving blood cells classification in peripheral blood smears using enhanced incremental training. In: Computers in Biology and Medicine (2021).

[4] Z. Li, F. Liu, W. Yang, S. Peng, J. Zhou. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. In: IEEE Transactions on Neural Networks and Learning Systems (2022).

[5] X. Pei, Y. Zhao, L. Chen, Q. Guo, Z. Duan, Y. Pan, H. Hou. Robustness of machine learning to color, size change, normalization, and image enhancement on micrograph datasets with large sample differences. In: Materials & Design (2023).

[6] C. Kanan, G. W. Cottrell. Color-to-Grayscale: Does the Method Matter in Image Recognition? In: PLoS One (2021).

[7] M. Dovbnych, M.P. Wójcik, A Comparison of Conventional and Deep Learning Methods of Image Classification. In: JCSI 21 303-308 (2021).

[8] S. B. Driss, M. Soua, R. Kachouri, M. Akil. A Comparison Study Between MLP and Convolutional Neural Network Models for Character Recognition. In: SPIE Conference on Real Time Image and Video Processing 10.1117/12.2262589 hal-01525504 (2017).

[9] V.E. Balas, N.E. Mastorakis. Multilayer Perceptron and Neural Networks. In: WSEAS Transactions on Circuit and Systems (2009).

[10] Z. Li, F. Liu, W. Yang, S. Peng, J. Zhou. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. In: IEEE Transactions on Neural Networks and Learning Systems (2022).

[11] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, A. A. Kalinin. Albumentations: Fast and Flexible Image Augmentations. In: Information (2018).