Introduction

The main aim for this study is to implement four machines learning algorithms to solve a cellule classification task. By utilizing the dataset, experiments on four algorithms in order to evaluate their suitability, performance, strengths and weaknesses for practical medical problems. This report could potentially inform some experiences for cellular recognitions. For instance, which machines learning algorithms have high accuracy and low runtime for identifying a cellule. In addition, this report could help future medical classification tasks. Such as choose the right algorithms for capturing tiny features of virus by applying the algorithms on the virus image. On the other hand, given that this study uses image data, the evaluation of different machines learning algorithms can demonstrate which one is more suitable for processing the image data. This following content provides potential insights for research on the integration of machine learning and medicine. Validate the advantages and disadvantages of four machine learning algorithms based on their theoretical ideas.

Data

The input data is a type of image, each with an input size of 28x28x3, which indicates that there are 28 by 28 matrix pixels (resolution) with three channels of colour red, green, and blue. According to the **current** study by (Yang et al., 2023), the original images have a resolution of 3x360x360 pixels, with centre-cropped into 3x200x200, and then resized into 3x28x28. With python data exploration techniques, there are 13637 rows for training data and 3419 rows for testing data with the target 8 unique classes. For each pixel, the value is in the range 0 to 255, which represents the corresponding RGB value. The following is a sample image for each class.

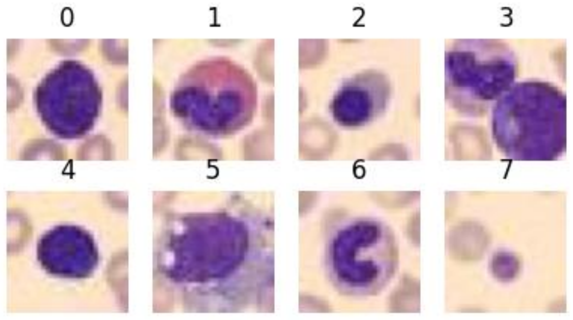


Fig. 1

For CNN and MLP, we chose three data augment techniques, which are saturation adjust, contrast adjust, and central crop. Referring to the above content and Fig. 1, we know that the images were modified to relatively small resolution, which may have the potential risk of losing tiny features. For example, in class 7, the whole cellule tends to be similar to the one besides it. By adjusting the saturation and contrast in Fig. 2, we can find that the boundary between the nucleus and cytoplasm becomes more identical. By central cropping, small cells' features, like those in the sample of class 7, can be enhanced. However, there is one more preprocessing technique we may apply: flipping the image horizontally. For instance, in Fig. 3 below, the cells from the same class 6 may be flipped to each other in the dataset. However, the network may treat one of them as an unseen image even though it has learned similar features before moving to another one. We considered doing the flip preprocessing during the model's training, but it costs much more runtime, up to 12 hours (Appendix Fig. 1). And we tried the random flip left and right with random seed 42 before input into the model, but there was a small effect on the accuracy. Therefore, we omitted this preprocessing technique.

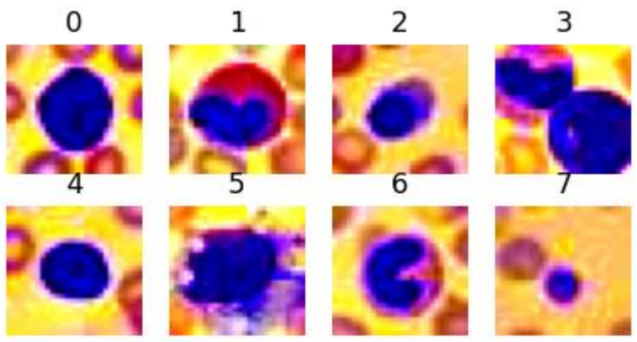


Fig 2

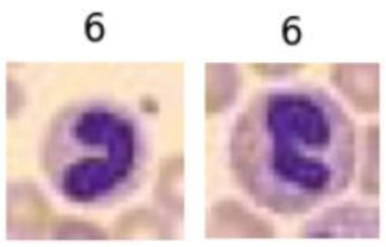


Fig 3

Methods:

MLP & CNN Theories:

The multi-layer perceptron (MLP) is a basic artificial neural network used for disposing the classification tasks, also known as feedforward neural networks. The fundamental architecture of a MLP consists of three layers: the input layer, hidden layers, and output layer. As the above exploration of data shows, there are 28x28 pixels with three RGB values. The following are explanations:

- **Input layer**: the input neurones will receive the original feature data. For the image data set, the data will be in the range of 0-255, which indicates the corresponding value in RGB.
- **Hidden layers**: at least one layer that receives the data from the input layer for processing. Neurones in this layer will be fully connected with the input layer, and they are also connected with the next upcoming hidden layer for passing the computational results. For setting up the numbers of hidden layers, the number of neurones in each layer usually indicates the architecture building. Activation functions are indispensable elements in neural networks. While training data, there are problems like vanishing and exploding gradients, which reduce the efficiency of learning complex data (new features). They help the neural networks tackle complicated tasks through non-linear calculations. For this assignment, Relu and Tanh are used to find the most accurate test accuracy.
    a. **Tanh** (Hyperbolic Tangent): the output values are in the range of -1 and 1. By using Tanh, the gradients will not be biassed, which can help the MLP optimise more efficiently. However, the Tanh function causes the vanishing gradient problem during classifying the cell images while the growing numbers of hidden layers.
    b. **Relu** (Rectified Linear Unit): as the result of the Relu computation will be in range 0 to 1, this activation function has the advantage of dealing with vanishing gradient problems that may be generated by the Tanh function. What's more, the Relu has higher computational efficiency than Tanh because the thresholding starts at 0 which makes the model to train data faster.
- **Output layer**: this layer receives the output from the last hidden layer for processing the classifications of the data. The numbers of neurones in this layer will be the numbers of target classes. Activation function will also be utilised in this layer, and it will usually be Softmax which is for calculating the probabilities for each class in the form of one-hot vectors.

The above theoretical ideas are the basic forward propagation process for a MLP, there are two more theoretical ideas, which are loss functions and backpropagation. Humans usually learn from mistakes, and so does the MLP. Loss function calculates the errors between predictions and actual

classes. Let the MLP learns from the errors by standard optimization method and then back propagate (update) the biases and weights to reduce the errors. Gradient descent algorithms such as Adam or SGD are used to optimize the MLP to minimise prediction errors. The weight initialisation is crucial for MLP since a bad initialisation could disturb the gradient while the computation process to the deeper dense (Simonyan & Zisserman, 2015).
The convolutional neural network (CNN) is a more effective network for dealing with image data. The fundamental architecture is similar with MLP, which includes the three basic layers: the input layer, the hidden layers, and the out layer. Besides that, the CNN has a more efficient hidden layer, which consists of convolution, activation, and pooling. After all, flatten the output from the last layer before transferring the data to the fully connected layer.

- **Convolution layer**: contains numbers of filters for detecting the features. Calculating the filter matrix into a feature map that explicitly represents the input image. The feature map helps to improve the network's efficiency for the following calculation of activation function. As the activation function will no longer calculate directly on the input data but instead calculate on more representative data, which is the feature map.
- **Pooling layer**: two strategies to reduce the number of parameters in the network are max pooling and average pooling (Week 8 lab). Respectively, take the max and mean of the pooling matrix. The selection of types of pooling depends on the missions for CNN. For instance, max pooling can protrude the most representative features of data, like high-contrast colours. On the other hand, average pooling is suitable for learning overall features of data that retain most of the original colours.
- **Fully connected layer**: the last layer for receiving the output of the pooling layer or convolution layer. This layer is mainly used to produce the classification of probabilities for each class and usually uses the Softmax activation function. However, some architectures may have an extended Relu or Tanh hidden layer before the output layer to robust results from pooling layers and convolution layers.

As the MLP and CNN have theoretical similarities, three formats of terms for introducing the strengths and weaknesses will be utilized: "Common Strengths" and "Common Weaknesses" will represent themselves for both MLP and CNN. Respectively, "MLP (CNN) Strengths (Weaknesses)" will indicate exclusive strengths or weaknesses for MLP or CNN.

Common Strengths

- **Flexibility**: arbitrary decisions of numbers of hidden layers and neurones can be customized regarding to specific tasks. Especially when tackling overfitting or underfitting issues while fitting on huge data volumes. Reducing the number of hidden layers can save computation performance. Nevertheless, the small number of hidden layers may lead to underfitting of test data and vice versa. A large number of hidden layers has a high risk to be overfitted with test data. For the number of neurones, it restricts the calculating range of fitting the activation function on sparse data. For instance, the reduction of neurones can be interpreted as a reduction of the input amount for layers, which reduces flexibility in terms of fewer weights between the input nodes and the nodes to hidden layers (Lavine & Blank, 2009). It can reduce the risk of the overfitting by controlling the number of neurones.
- **Non-linearity**: by applying a non-linear activation function, the neural network can perform various tasks. The features of the data should not be similar between before and after learning. Otherwise, the neural network will lose its functionality. As shown by the linear function for feature processing (Appendix Formular 1), the features learned from the numbers of hidden layers will be the same as one-layer feature processing. Non-linearity can help the perceptron learn more detailed and complex relationships from the data. For

example, the output of the Relu function can be either 0 or 1, which provides the perceptron with more decisions to learn from the data.

MLP Strengths:

- Low architecture complexity: the MLP is more concise than other neural networks, such as CNN. The hidden layers are simply assembled by the fully connected layers exclusively. Which makes the MLP's overall interpretability higher than others because of the plain architecture.
- High adaptability to different missions: the MLP does not require spatial data relationships like the CNN. The input data for MLP usually needs to be flattened into a one-dimensional vector, which has no consideration of data positions. This characteristic gives MLP the ability to learn from different types of data, which include image data and tabular data. Due to the common strength flexibility, the MLP can also accept corresponding activation functions for dealing with regression or clustering problems.

CNN Strengths

- More efficient for spatial data classification: in order to generate a recognizable image, the image composes RGB colour pixels with invariant positions. According to the theory of CNN, it has the convolutional layer and pooling layer. With compared to the architecture of MLP, CNN process the data from convolutional layer and pooling layer instead of flatten data initially. The CNN utilize the convolutional layer to capture the representative features in an area of features and pooling them for refinement. Along with the filter strides through the input data, the original position of pixels retained mainly and become more representative. Because the input data has not be flattened, the CNN can learn new features without disrupting the positions of pixels. As the relationships among pixels matters, marginal pixels have same weights (share weights) as all other pixels because they are important to classifying the image as well. Now the padding can help the filters to capture marginal pixels better while the stride tends to move beyond the image. In general, the convolutional layer with padding and pooling layer helps the CNN learn image features on the original data. This is the main reason why CNN has more efficiency when dealing with the spatial data like image.

Common Weaknesses

- Low interpretability in hidden layer: for large number of hidden layers neural networks, the calculations will become hard to trace and probe. Due to the non-linearity of neural network, it tends to be complicated for human understand exactly what decisions the layer made. Furthermore, it hard to estimate whether the optimizer is at which local minimum. As well as when the vanishing gradient problem occurs, its hard form human to catch the issue where it happens at which propagation step.
- Long run time: due the architecture of neural networks, they usually consist with big number of perceptron or many layers network. For instance, a simple residual network can be 18 layers. There is a linear relationship between the complexity of networks and runtime. Furthermore, the neural network involves more hyperparameters which occupies more run time while searching the best parameter combination while tuning. Additionally, the volume of data used to learn by networks tends large scale which will increase the calculated quantity for hardware. Another weakness derives from the long run time which is hard to tuning the hyperparameters. The runtime for CNN number of hyperparameters increases exponentially.
- Overfitting issue: as explained in the long run time weakness, the network's architecture usually tends to be more complex in order to learn more features. It can a double-edged sword, a well-designed MLP architecture may overfitting while training a small volume of

data. Because a complex architecture may learn the noise or false-positive data that are not the actual predictions. With compared to CNN, despite it has the convolutional layer and pooling layer with anti-overfitting activation function, overfitting still occurs for same reason.

MLP Weaknesses

- Easy to get vanishing gradient problem: the MLP is sensitive to initial weight and bias. Because the hidden layer will back be propagating the weight base on each new bias. With regarding to dense of an architecture, inappropriate weight initialization (very small value) may impede the network to capture enough features. Big value of bias initialization can also lead to vanishing gradient on a relative shallow dese network.

Random Forest

The ensemble method chosen is Random Forest, which combines bagging and decision tree. Random Forest will run iteratively, in each iteration, it randomly selects a subset of features from the training data, and build a decision tree. Decision tree is a classification method, it consists of nodes that help divide data into different classes according to their attributes. In each node splitting, it will select the best attributes to split on, and the attributes are the ones that with highest information gain. When predicting, Random Forest will combine the predictions from all decision trees and output the final prediction.

Strength

- Improved accuracy compared to decision tree: Random Forest combines predictions from individual trees by majority voting. The accuracy is often better then single decision tree.
- Overfitting prevention: With the use of bagging and randomness, it can help reduce the amount of overfitting at each individual decision tree.
- Good interpretability: There are method to visualize the decision path and attributes of each split.

Weakness

- Computationally expensive: Random Forest requires great amount of computational power because it needs to build multiple decision trees. If using traditional CPU to compute the algorithm, the runtime might be long.
- Difficult to separate similar classes: If patterns between classes are quite similar, Random Forest might confuse them with each other, failed to separate similar classes result in a lower accuracy (Amini & Shalbaf, 2022).

Support Vector Machine

In classification problem, SVM can find decision boundary to separate data into different classes using hyperplane with the maximum margin. Not just linear decision boundary, SVM can also find non-linear boundaries by transforming the data to a higher dimensional space where a linear decision boundary can be found. And uses kernel trick to do calculation in original dimension to reduce the computational complexity.

Strength

- Capable of handling complex data: SVM can find linear and non-linear boundaries, it has the ability to handle some complex (multi-dimensional) classification problem like image classification.
- Can reduce the risk of overfitting: Can control the generalisation of the model by adjusting parameter and reduce the risk of overfitting (Auria & Moro, 2008).

Weakness

- Hyperparameter tuning can be difficult: There are many hyperparameters to be choose from, and each will have great impact on the performance. And there are multiple kernels as well, might need to try out every kernel and check whether it fits the data well.

Architecture and hyperparameters

- MLP Architecture
  - Initially, the image will be flattened before it passed into the first hidden layer. As the input size for us to process is (28 x 28 x 3) RGB image, a vector after flattening will be generated with the length 2352 for each image. The first hidden layer may start from 1024 neurones, that's because we want enough neurones to capture general features for better generalization ability. We used the Kaiming He's weight initialization method for our model (Kaiming, Xiangyu, Shaoqing & Jian, 2015). The weights will be initialized according to a Gaussian distribution which is suitable for the Relu activations. We also set the bias initialization to zero as we want a relative balance position to start the model. On the other hand, we tried not to set the similar number of neurones like the vector length. Because it is for MLP to capture some features we do not expect like noises. Then we can decrease the number of neurones progressively by divided by 2 in order to capture more representative features. Due to the issue of that MLP is easy to get overfitting, we designed the architecture based on the fundamental three layers outline with one dropout layer. Mainly use the Relu function for hidden layer and Softmax function for output layer. According to the data exploration, we set the output layer with 8 neurones for receiving the classes. Thus, we chose the sparse categorical cross entropy as our loss function because the labels were integer encoded.
- CNN Architecture
  - We designed a relative less deep CNN architecture base on the Visual Geometry Group (Simonyan & Zisserman, 2015) architecture. Due to the lacking performance of our local machine, we could not afford a full version of VGG-16 or VGG-19 layers neural networks and we assumed that the grid search tuning will cost much time. We limited the number of our hidden layers (two convolutional layers with following one pooling layer) to be maximum 4 layers. As the input size of the image is 28x28x3, we chose a small convolutional filter size 3x3 with one padding. Thus, the filter should have total 10 strides on both horizontal and vertical respectively. We used the max pooling for our pooling layer, that's because we preprocess the input data by adjusting the contrast and saturation. We want the max value in order to capture the significant colours. Furthermore, two fully connected layers followed by all above convolutional layers and pooling layers for classifying and learning all features from above. After all, last fully connected layer with 8 neurones to calculate the target class.
- MLP and CNN common Hyperparameters
  - **optimizer__lr**: it controls the learning rate of the optimizer. We expect faster training with higher learning rate, but we may not find a good global minimum in restrict epochs. Slower training with lower learning rate, but big chance to find a global minimum.
  - **batch_size**: it controls how many samples will be trained each time before the mode updates. We expect the model trains faster with larger batch size but high risk of overfitting. We should tune this hyperparameter with the epochs together.
  - **epochs**: it controls how many times the model trains on the entire dataset. We expect overfitting for many epochs like above 20 epochs. Small number of epochs may underfitting.

    - **activation_function**: it controls which activation function will be applied into the hidden layer in MLP or convolutional layer and pooling layer in CNN. We expect Relu can fix the vanishing gradient problem while the accuracy increases as we tune the hyperparameters. Which also provide us with a relative higher accuracy than Tanh activation function.
- MLP exclusive hyperparameters
    - **n_hidden_layers**: it controls how many hidden layers in the model. We expect more hidden layers and higher accuracy. At the beginning of training the model, we expect overfitting as we set more layers in the model.
    - **n_hidden_ neurones**: it controls how many neurones in each hidden layer. Dynamic changes with the n_hidden_layers (Divided by 2 while increase one hidden layer). We expect the algorithms can learn more features and overfitting while we start with large number of neurones such as 4096.
- CNN exclusive hyperparameters
    - **n_convolutional_layers**: it controls how many convolutional layers in the CNN. We expect more layers can learn more representative colour features from the input image. However, it should be overfitting if we set an extreme deeper network like more than 5 layers.
    - **n_filters**: it controls how many convolutional kernels in each layer. Higher number of filters can make the algorithm tend to overfitting.
    - **n_fully_connected_neurones**: it controls how many neurones in each fully connected layer. Higher number of fully connected neurones can receive most of the features from convolutional layers and pooling layers. But it can cause the overfitting problem and increase the algorithm runtime.
- For both MLP and CNN, we used the grid search method for tuning the hyperparameters. Because the grid search accepts several hyperparameters together and run them by combinations. The output will always be the best combination from the input hyperparameters which can free our time mostly. We can have comparisons among all combinations which help us to find out the boundary hyperparameters. That supports us with good understanding of our model by the results from the grid search. However, it also increases the difficulty to debug the code while running. Due to the black-box of neural network, we cannot distinguish the grid search is running normally or is stuck in somewhere, or if some parameters are making the computation growth explosively. Therefore, we chose the random grid search for CNN for reducing the overall runtime and risks of extreme long time running. This grid search method randomly picks some of the hyperparameters combinations instead of all combinations. It supports us with an opportunity to try more hyperparameters. However, we assumed that there would be some loss. As random search has probability not to reach the potential best combinations, we would run several times to experiment for finding out the most possible best combinations. In general, we chose the grid search mainly because we can get comparisons which give us overall information of a batch of hyperparameters. But we must be careful and rigorous when design the models to avoid waste of time because of undetectable bugs.

Random Forest hyperparameters

GridSearchCV with 3-fold cross validation is used to verify all combinations of hyperparameters. The parameters selected for RandomForestClassifier are ‘criterion’, ‘n_estimators’, and ‘min_samples_split’.

- **criterion**: defines how to measure the quality of a split during tree-building. There are two criterions to choose from, ‘gini’ which stands for Gini impurity and ‘entropy’ for Shannon information entropy.

- **n_estimators**: define the number of decision trees will be build, given that the dataset is relatively large, we choose four different scales: 128, 256, 512, and 640, to try to find out the difference of their performance and relation between different number of trees. More trees built might increase the performance of the model.
- **min_samples_split**: specifies the minimum number of samples required for a node before splitting. If the number is small, the tree will split more often, result in more complex trees. And a larger number result in simpler trees, which help prevent overfitting to a certain extent. Three different scales: 2, 6, and 10 were chosen, to find out how model perform in different splitting strategies.

One thing to note that we did not specify the ‘max_sample’, this parameter controls the maximum number of samples in each decision tree, if not specified, the whole training set will be used. We want that to increase the strength of each individual trees.

SVM hyperparameters

We use GridSearchCV with 3-fold cross validation to search through the best hyperparameter combination. When tunning parameters for SVM, first we need to determine which kernel to use. There are four kernel to choose from in SVM: linear, polynomial, RBF, and sigmoid. Their decision boundaries are different (see Fig 4). And the linear kernel is the only one that does not apply the kernel trick.

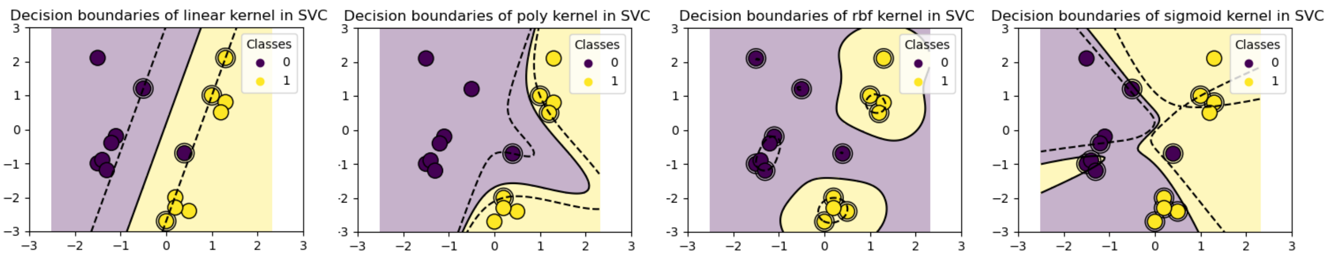


Fig 4. Decision boundaries of different kernel (Plot classification boundaries, n.d.)

Other parameters chosen are ‘C’, ‘gamma’, and ‘degree’.

- **C:** the regularization parameter, which controls the complexity of the model to prevent overfitting. When C is small, the model tends to choose a hyperplane with larger margin and allows for some misclassifications, which reduce the risk of overfitting. When C is large, the model will focus on reducing misclassifications, but might result in overfitting. We ran a few different scales on the dataset, and narrow down our selection to 6, 8, and 10.
- **gamma:** defines the kernel coefficient, not used for linear kernel. It regulates the impact of each training sample on the decision boundary. With smaller gamma, the model tends to create a smoother decision boundary. And with larger gamma, the model tends to create a more complex decision boundary. After testing different scales on the dataset, we narrow down the number to be under 0.02, and select 0.008, 0.01, 0.012, 0.014, and ‘scale’ (1 / (n_features * X.varience)) for comparison.
- **degree:** controls the degree of polynomial kernel. Changing the degree will change the shape of the decision boundary of polynomial kernel. 3, 4, 5 are selected for comparison.

Since there are some parameters that are unique to certain kernel, we decided to compare the kernel first, and choose the best kernel to run the GridSearchCV. We tested the four kernels on the dataset with their default parameter, and the polynomial and RBF kernel performs the best (see Fig 5).

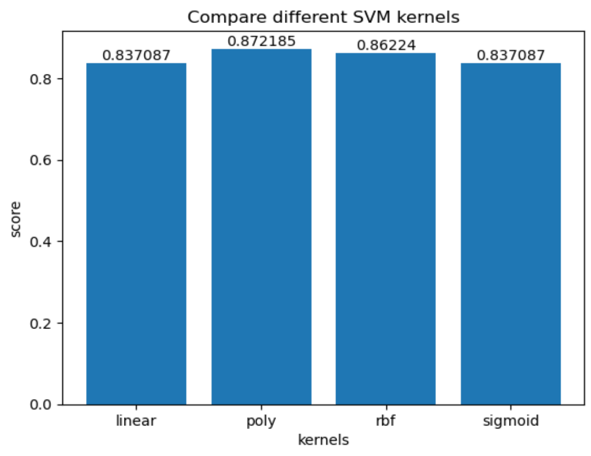
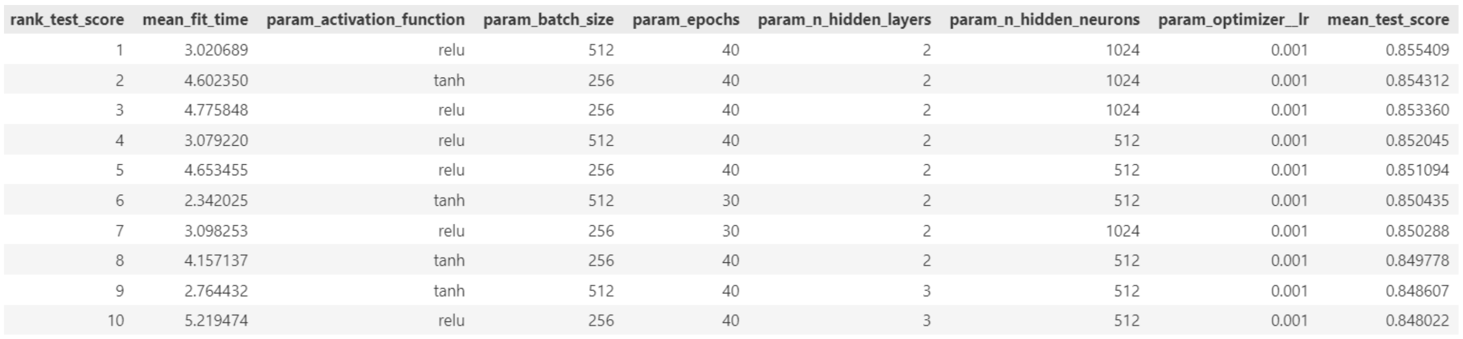


Fig. 5

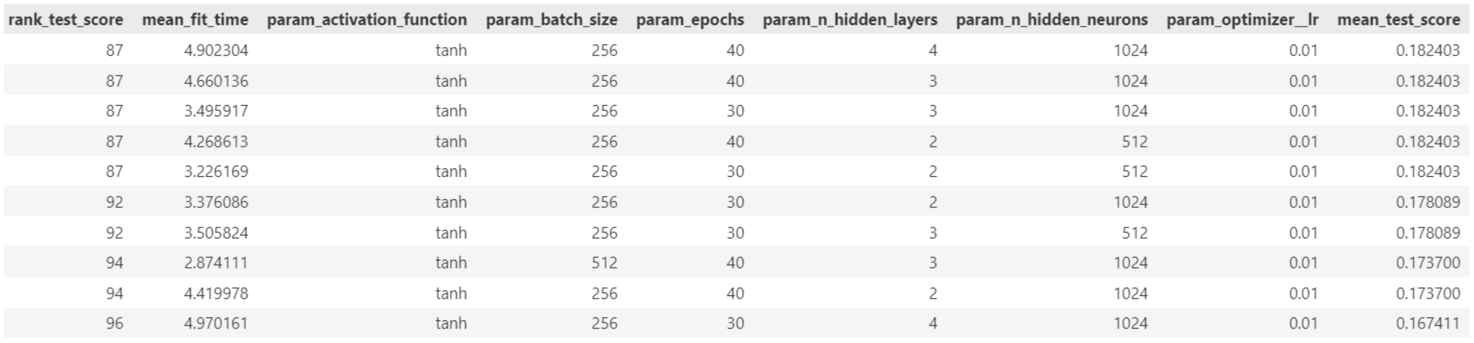
Therefore, we decided to choose polynomial and RBF kernel and run grid search on them. And since the ‘degree’ parameter is only for polynomial kernel, so we separate them into two grid searches then compare their best parameters and score afterwards.

Results and Discussion



rank_test_score	mean_fit_time	param_activation_function	param_batch_size	param_epochs	param_n_hidden_layers	param_n_hidden_neurons	param_optimizer__lr	mean_test_score
1	3.020689	relu	512	40	2	1024	0.001	0.855409
2	4.602350	tanh	256	40	2	1024	0.001	0.854312
3	4.775848	relu	256	40	2	1024	0.001	0.853360
4	3.079220	relu	512	40	2	512	0.001	0.852045
5	4.653455	relu	256	40	2	512	0.001	0.851094
6	2.342025	tanh	512	30	2	512	0.001	0.850435
7	3.098253	relu	256	30	2	1024	0.001	0.850288
8	4.157137	tanh	256	40	2	512	0.001	0.849778
9	2.764432	tanh	512	40	3	512	0.001	0.848607
10	5.219474	relu	256	40	3	512	0.001	0.848022

Fig. 6 Top 10 MLP Search Results



rank_test_score	mean_fit_time	param_activation_function	param_batch_size	param_epochs	param_n_hidden_layers	param_n_hidden_neurons	param_optimizer__lr	mean_test_score
87	4.902304	tanh	256	40	4	1024	0.01	0.182403
87	4.660136	tanh	256	40	3	1024	0.01	0.182403
87	3.495917	tanh	256	30	3	1024	0.01	0.182403
87	4.268613	tanh	256	40	2	512	0.01	0.182403
87	3.226169	tanh	256	30	2	512	0.01	0.182403
92	3.376086	tanh	256	30	2	1024	0.01	0.178089
92	3.505824	tanh	256	30	3	512	0.01	0.178089
94	2.874111	tanh	512	40	3	1024	0.01	0.173700
94	4.419978	tanh	256	40	2	1024	0.01	0.173700
96	4.970161	tanh	256	30	4	1024	0.01	0.167411

Fig. 7 Tail 10 MLP Search Results

According to the tabular above, we observe that the best combination for our MLP model is Relu activation function, 512 batch size, 40 epochs, 2 hidden layers and 1024 neurones at the first layer with decrease half number for the following layers. For the top 10 combinations (Fig. 6), 6 of them utilized the Relu activation function which satisfied our expect. As the MLP is sensitive to the weight and bias initialization, the Relu helps the model to reduce the effect of suffering the vanishing problem. That makes the model gain a relatively good performance and better accuracy score. The tail 10 result (Fig. 7) validates the idea that Tanh function is easier to suffer from the vanishing gradient problem. For general consideration, we hypothesized that more hidden layers would have higher score. But the table shows that accuracy of 3 and 4 hidden layers is under the one of 2 hidden layers. The issue would be overfitting as the model learnt some noise data while the network becomes deeper.

rank_test_score	mean_fit_time	param_n_fully_connected_neurons	param_n_filters	param_n_convolutional_layers	param_activation_function	param_batch_size	param_epochs	param_optimizer__lr	mean_test_score
1	13.836229	1024	64	2	relu	512	20	0.001	0.901266
2	23.153002	1024	64	4	relu	256	20	0.001	0.898924
3	19.482323	1024	64	3	tanh	512	20	0.001	0.889930
4	7.357376	1024	32	3	relu	256	15	0.001	0.887224
5	14.735374	1024	64	3	tanh	512	15	0.001	0.884517
6	7.983814	1024	32	4	sigmoid	512	15	0.01	0.194763
6	11.433093	2048	32	4	relu	256	20	0.01	0.194763
6	23.011941	2048	64	4	relu	256	20	0.01	0.194763
6	14.056819	1024	64	2	tanh	512	20	0.01	0.194763
6	7.334804	2048	32	2	sigmoid	256	15	0.001	0.194763

Fig. 8 All 10 CNN Search Results

According to the table above (Fig. 8), we observe that the Relu function still has relatively better performance on CNN. Notably, we find that the individual hyperparameter in the best hyperparameter combination are all minimum values within their corresponding groups (except for the activation function). This could indicate that the network has a large parameter interval or an insufficient amount of input data, resulting in the model suffering overfitting when large values of hyperparameters are tuned.

Random Forest Result

The best combination found for Random Forest is 'entropy' for criterion, min_samples_split=2, and n_estimator=640. With those, the model has the best cross validation score of 83.65%, and the accuracy on the testing set is 84.73%. We can see the result of different hyperparameter combinations (see Fig 9). Majority of the top 5 scores having their n_estimators equal to 512 and 640, and with min_samples_split equals to 2. While majority of the tail 5 scores having their n_estimators equal to 128, which is the lowest number we selected (see Fig 10). That maps our prediction, the model performs better with more complex trees. However, building and fitting a more complex tree will increase the fit time by a significant amount.

rank_test_score	param_criterion	param_min_samples_split	param_n_estimators	mean_fit_time	mean_test_score
1	entropy	2	640	169.035754	0.836466
2	entropy	2	512	138.142739	0.835881
3	gini	6	640	41.644914	0.833907
4	entropy	6	512	159.329389	0.833614
5	gini	2	256	32.298254	0.833395

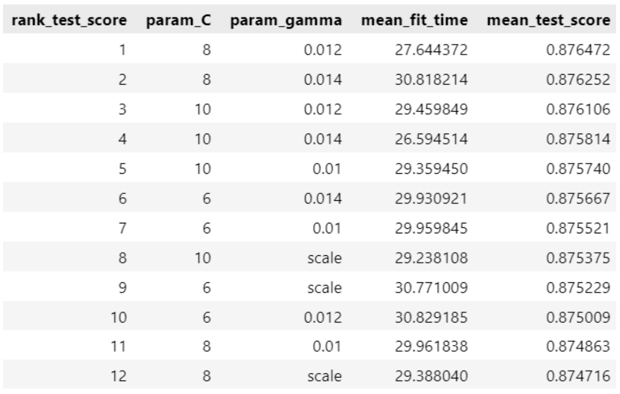
Fig. 9 Top 5 Parameter combination for Random Forest Classifier

20	gini	6	256	17.625843	0.829665
21	gini	6	128	10.029167	0.827763
22	entropy	6	128	37.140964	0.827105
23	gini	10	128	9.937080	0.826081
24	entropy	10	128	41.486339	0.825496

Fig. 10 Tail 5 Parameter combination for Random Forest Classifier

SVM Result

The best combination found for SVM uses RBF kernel, with C=8 and gamma=0.012. The model has the best cross validation score of 87.65%, and accuracy on the testing set is 89.97%. There are no significant differences between the top 5 scores, and their selection of gamma is relatively even, with C equals to 8 and 10 (see Fig 11). While the tail 5 scores tend to have smaller C and gamma. Result shows that model with larger C and gamma is likely to have a better performance, but their fit time is not having a big difference.

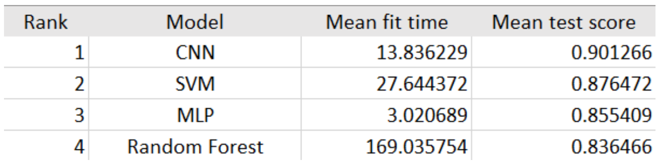


rank_test_score	param_C	param_gamma	mean_fit_time	mean_test_score
1	8	0.012	27.644372	0.876472
2	8	0.014	30.818214	0.876252
3	10	0.012	29.459849	0.876106
4	10	0.014	26.594514	0.875814
5	10	0.01	29.359450	0.875740
6	6	0.014	29.930921	0.875667
7	6	0.01	29.959845	0.875521
8	10	scale	29.238108	0.875375
9	6	scale	30.771009	0.875229
10	6	0.012	30.829185	0.875009
11	8	0.01	29.961838	0.874863
12	8	scale	29.388040	0.874716

Fig 11. Parameter combination for SVM

Discussion

As the result shows (see Fig 12), with the convolutional layer and pooling layer of CNN, it has distinct advantage when processing image data compared to other models, and naturally attains the highest score among the four.



Rank	Model	Mean fit time	Mean test score
1	CNN	13.836229	0.901266
2	SVM	27.644372	0.876472
3	MLP	3.020689	0.855409
4	Random Forest	169.035754	0.836466

Fig 12. Four models with their best result

And SVM used its advantage in handling multi-dimensional data, and coupled with the adjustable regularization parameter it has, that allows SVM to outer perform MLP in terms of score. CNN and MLP has more complex architectures, the architecture design often greatly affects their performance. But with the help of GPU, CNN and MLP are able to reduce a great amount of runtime, they have the lowest runtime among the four. In comparison, SVM and Random Forest only uses traditional CPU during processing, even the fastest SVM takes twice the time as CNN. Last but not least, affected by the lack of complexity and difficulty when distinguishing similar classes, Random Forest had scored the last. And the multiple decision tree's structure makes its runtime 'far ahead' of other models, which is over 10 times as CNN.

Conclusion

With the highest score and relatively short runtime, we consider CNN to be the best classification model for our image dataset. And Random Forest has the lowest score among the four, with very long runtime, we believe that Random Forest is not the first choice for this kind of image classification problem. MLP is having a great potential, its runtime is the lowest, and there is potential for further optimization.

Given the constrains of limited time and computing power, we had difficulty in balancing the research in models and research in optimization methods, therefore we have not managed to find a better way in model optimization. In the future, we can do more research on the architecture and optimization methods for CNN and MLP, to improve the robustness and performance of the models.

Reflection

This assignment offers us multitudinous experiences while utilizing the machine learning algorithms on real world problem. It provides us with a whole machine learning research pipeline, which gives us warm-up to modern machine learning study regarding the image classification task. Furthermore, it gives us the insights that neural networks such as MLP and CNN are good at learning image data due to their character of automatically weight and bias adjustment. Additionally, this assignment also gives us a deeper understanding of how these four machine learning algorithms work and how to pick the appropriate hyperparameter based on external factors such as input size. Finally, this assignment is exciting because we have not only used advanced machine learning algorithms for the same task but also compared them with traditional machine learning algorithms. It makes us excited about the future of AI learning.

Reference:

Amini, N., & Shalbaf, A. (2022). Automatic classification of severity of COVID-19 patients using texture feature and random forest based on computed tomography images. *International Journal of Imaging Systems and Technology*, *32*(1), 102–110. https://doi.org/10.1002/ima.22679

Auria, L., & Moro, R. A. (2008). Support Vector Machines (SVM) as a Technique for Solvency Analysis. *IDEAS Working Paper Series from RePEc*.

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv.org. https://doi.org/10.48550/arxiv.1409.1556

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. arXiv.org. https://doi.org/10.48550/arxiv.1502.01852

B.K. Lavine, T.R. Blank, 3.18 - Feed-Forward Neural Networks, Editor(s): Steven D. Brown, Romá Tauler, Beata Walczak, Comprehensive Chemometrics, Elsevier, 2009, Pages 571-586, ISBN 9780444527011, https://doi.org/10.1016/B978-044452701-1.00026-0.

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), 41–41. https://doi.org/10.1038/s41597-022-01721-8

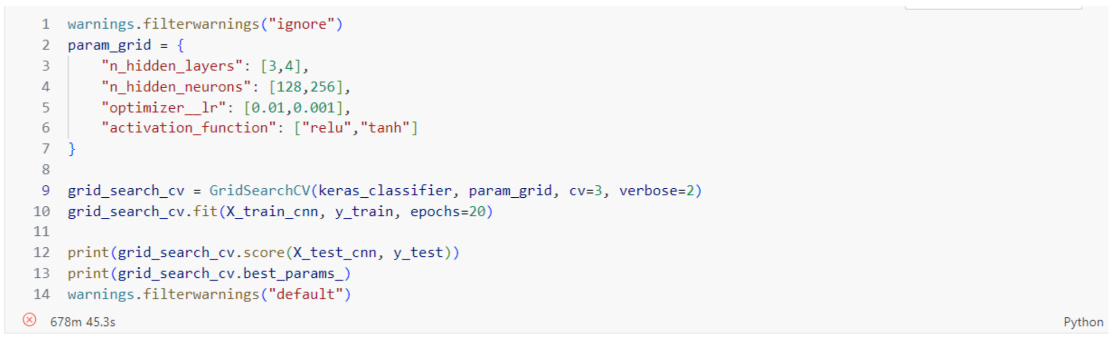
Plot classification boundaries with different SVM Kernels. (n.d.). Retrieved October 20, 2023, from: https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html#sphx-glr-auto-examples-svm-plot-svm-kernels-py

Appendix:

$$A^L = {W^L}^T A^{L-1} = {W^L}^T {W^{L-1}}^T \cdots {W^1}^T X$$

$$A^L = W^{total} X$$

[Formular 1]



```python
warnings.filterwarnings("ignore")
param_grid = {
    "n_hidden_layers": [3,4],
    "n_hidden_neurons": [128,256],
    "optimizer__lr": [0.01,0.001],
    "activation_function": ["relu","tanh"]
}

grid_search_cv = GridSearchCV(keras_classifier, param_grid, cv=3, verbose=2)
grid_search_cv.fit(X_train_cnn, y_train, epochs=20)

print(grid_search_cv.score(X_test_cnn, y_test))
print(grid_search_cv.best_params_)
warnings.filterwarnings("default")
```

Appendix Fig. 1