Image Classification Report

Author: Mengshen Guo (520074642), Yanjie Wu (520076163)

Abstract

This report introduces and demonstrates the use of random forest, multilayer perceptron and convolutional neural network and their hyperparameter tuning process. The advantages and disadvantages of the different algorithms are explored by comparing their performance on the Fashion_MNIST dataset. The purpose of this report is to realize an exploration of the experimental process of image classification and to provide important suggestions for a better choice of image classification algorithms as well as to demonstrate an understanding of the nature of the algorithms.

Contents

Abstract 1

1. Introduction 3

2. Data 3

2.1 Data description and exploration 3

2.2 Pre-processing 4

3. Methods 4

3.1 Summary 4

3.1.1 Random Forest Algorithm 4

3.1.2 Fully connected neural network 5

3.1.3 Convolutional neural network 5

3.2 Strengths and weaknesses 6

3.2.1 Random Forest Algorithm 6

3.2.2 Fully connected neural network 6

3.2.3 Convolutional neural network 7

3.3 Architecture and hyperparameters 8

3.3.1 Random Forest Algorithm 8

3.3.2 Fully connected neural network 9

3.3.3 Convolutional neural network 11

4. Results and discussion 13

4.1 Random Forest Algorithm 13

4.2 Fully connected neural network 15

4.3 Convolutional neural network 16

5. Conclusion 18

6. Reflection 19

7. Reference 20

1. Introduction

In the information age, images are of great value because they often carry a large amount of information. When faced with a large number of images, it becomes an important research direction to classify them effectively and obtain more information in less time by IT algorithms. The performance of different image classification algorithm and their hyperparameter that effects on the results are the aim that this report needs to focus on. Currently, there are many different approaches to image classification, and the most widely used image classification algorithms are mainly focused on machine learning and deep learning. This report hopes to show the use of different algorithms based on machine learning and deep learning in order to explore their pros and cons. The comparison and analysis of different algorithms in this report will provide important suggestions for better selection of image classification algorithms.

2. Data

2.1 Data description and exploration

The dataset used for the experiments in this report is the Fashion_MNIST dataset, which uses a three-dimensional array to store 60,000 training images and 10,000 test images, where each image has a size of 28*28. The original source of the Fashion-MNIST dataset are from the classification on the Zalando website (Xiao,

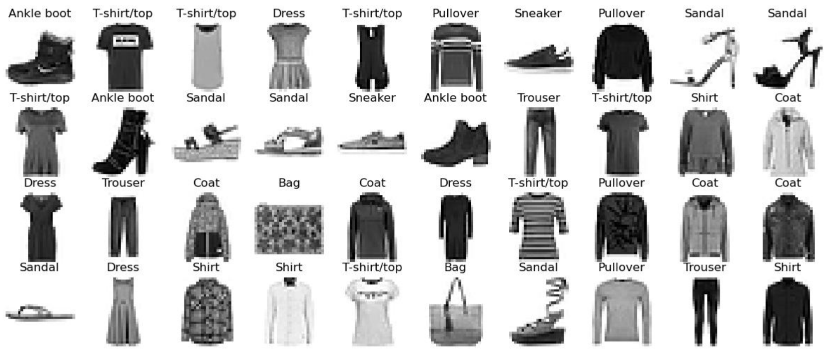

Figure 1: Partial samples from the Fashion_MNIST dataset

Rasul, & Vollgraf, 2017). It can be seen in Figure 1 that the dataset is associated with ten classes of labels, including T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. In all classes, clothing types, such as T-shirt/top, pullover, coat and shirt, and shoes type, such as sandal, sneaker and ankle boot, have similar features while other classes differ more. This dataset can be seen as an upgraded version of the MNIST dataset, so it has similar features to the MNIST dataset. Like MNIST, the Fashion_MNIST dataset also uses grayscale images as its data

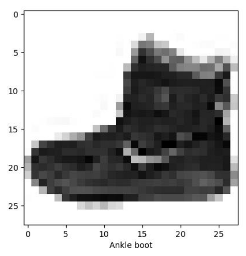

Figure 2: Single sample from the Fashion_MNIST dataset

source, but as can be seen by the single displayed images (Figure 2), the complexity is much greater than that of the MNIST dataset, so the training process is more challenging and difficult, which results in a final accuracy rate that is not as good as the MNIST dataset shows.

2.2 Pre-processing

First of all, the data needs to be normalized. Since the image data stored in the dataset is unit type from 0-255, the normalization process can convert the data range into 0-1 to facilitate displaying the images. Also, this process helps simplifies the subsequent algorithm and speeds up the training time. In order to adapt the machine learning algorithm afterwards, the original three-dimensional array needs to be stretched and transformed into two-dimensional array by using the *reshape* function. Finally, the training set is sliced and divided into a new training set and a validation set to facilitate the training and validation of the final deep learning model. Also, in order to make the dataset adaptable to the classifier of the convolutional neural network, the experiment needs to add a dimension to the classified training set, validation set and test set. The dimension means the number of values per image pixel of the data. Since the images in the Fashion_MNIST dataset are black and white, the dimension here is filled with 1. For color image classification, the image channel is usually filled with 3.

3. Methods

3.1 Summary

3.1.1 Random Forest Algorithm

The random forest algorithm is an ensemble learning method based on decision trees and is widely used in problems such as classification and regression (Great Learning Team, 2020). As a classical supervised learning method, the decision tree algorithm utilizes a tree structure in which leaves represent labels and branches represent the combination of features that lead to these classes of labels for classification and prediction. However, single decision trees often suffer from overfitting problems when faced with complex and noisy data, significantly reducing their accuracy. For this reason, ensemble methods are often used to solve the decision tree overfitting problem. Random forest is a new machine learning algorithm after ensemble learning of decision trees, which corrects the problem that decision trees are prone to overfitting and outperforms decision trees in terms of performance and accuracy.

When solving the image classification problems, the first thing that needs to be ruled out is algorithms like logistic regression that deal with binary classification problems, which are clearly not suitable for the Fashion_MNIST dataset with ten class labels. Although the algorithms such as knn algorithm are simple and easy to understand, and support vector machine algorithm can get higher accuracy, its running time and hyperparameters tuning process are too long and complicated due to the large training set size, so neither of these is a good choice (Sheykhmousa et al., 2020). Meanwhile, Naïve Bayes can handle high-dimensional data more efficiently, but its accuracy is not ideal. The final choice falls on decision trees, ensemble learning, and random forest algorithms. As an optimization algorithm for decision trees, it is obvious that random forests with better parameter settings and classifier algorithms are a better choice. In fact, when faced with an image classification problem, especially when the training set is large, random forest can handle the problem more efficiently and accurately. However, if the training set is small, support vector machines can also be another good choice.

3.1.2 Fully connected neural network

The multilayer perceptron plays an important role as a fully connected feedforward neural network in problems such as image classification. Artificial neural network, as a machine learning algorithm that simulates the function of the human brain, is composed of the interconnection between a large number of neurons (Hardesty, 2017). Each neuron has a specific output function, also known as the activation function. The connections between neurons represent the weighted values between signals, also known as weights. In this way, it is possible to make the neural network achieve an effect of self-classification and self-learning. A neural network structure in which the input layer gets the output through an activation function is called a perceptron. Single perceptron is often unable to solve nonlinear or heterogeneous problems, so there is a multilayer perceptron. By adding hidden layers between the input and output layers, it can classify and learn complex problems. In a multilayer perceptron, the neurons in each layer only receive the output of the neurons in the previous layer and output to the neurons in the next layer (Taud & Mas, 2017). The information in the whole network is propagated in one direction, which is a typical feedforward neural network (Lavine & Blank, 2009).

3.1.3 Convolutional neural network

CNN (Convolutional Neural Networks) is a feed-forward neural network with convolutional structure, which can reduce the amount of memory occupied by the deep network by learning directly from data, eliminating the need for manual feature extraction (Mathworks 2022). Three key operations - local perceptual field, weight sharing and pooling layer - effectively reduce the number of parameters of the network and alleviate the overfitting problem of the model.

CNN mainly consists of the following structures:

- Input layer: Input data.
- Convolution layer (CONV): Feature extraction and feature mapping using convolution kernels.
- Activation layer: Non-linear mapping (relu).
- Pooling layer (POOL): Down sampling and dimensionality reduction.
- Rasterization (Rasterization): Unfolding of pixels, fully connected to the fully connected layer, which can be omitted in some cases.
- Fully Connected layer (FC): Fitting at the tail to reduce the loss of feature information.
- Activation layer: non-linear mapping (relu).
- Output layer (Output layer): Output results.

3.2 Strengths and weaknesses

3.2.1 Random Forest Algorithm

The strength of random forest algorithm is that it solves the problem of overfitting of decision tree algorithm, so that it can have high accuracy even for complex training data. The highly concurrent training method speeds up the training time and makes it more flexible for solving most classification and regression problems. In addition, another feature of random forest is that it does not require much clean data, as it can automate missing values and handle irregular data well, which makes it more widely applicable to different types of training data (Great Learning Team, 2020). However, the weakness of using random forest algorithm is that it requires a large amount of data, which requires a lot of computing power and resources as support to help in the construction of trees (Great Learning Team, 2020). At the same time, although random forest can process data with high concurrency, in order to ensure accuracy, more data needs to be trained, so it still requires a long training time. Also, since the final result is a collection of a large number of decision trees, this leads to a certain impact on the interpretability of the algorithm, which cannot determine the importance of each variable well.

3.2.2 Fully connected neural network

As a classical algorithm of neural networks, the multilayer perceptron has high parallel processing capability and scalability, which makes it capable of handling large amounts of complex data and thus is often applied in different fields such as image recognition and speech recognition. Also, the multilayer perceptron has high accuracy in handling nonlinear problems and fitting smooth nonlinear functions, which indicates that the algorithm has good fault tolerance and strong adaptability and self-learning ability. However, good fault tolerance for correct data usually means that the algorithm also adapts easily to noisy data, while fitting to noise often means that the final model generalizes poorly to the test set and tends to fall into local extremes or leads to inadequate learning (Anguita, Ghelardoni, Ghio, Oneto, & Ridella, 2012). In addition, the slow learning speed of the multilayer perceptron and the tuning of its parameters is also a major weakness. The inability to identify the nodes of the hidden layer and the choice of different activation functions increase the challenges of the algorithm during hyperparameters tuning.

3.2.3 Convolutional neural network

The strength of convolutional neural networks lies in the ability to reduce the dimensionality of large data volumes to small data volumes. An image is made up of many pixels, each of which in turn is made up of colour. A colour image has three channels of RGB (i.e., optical triplet) for each pixel, which is used to represent colour information. So, for a common 10000*10000 definition picture, it would contain 10000*10000*3 values to be calculated, which is obviously a huge amount of data. To solve this problem, CNNs reduce the large number of parameters into a small number of parameters. It is important to note that the dimensionality reduction does not affect the results. For example, if an image is reduced in size within a certain range, the features of the image remain unchanged. What’s more, it can also effectively retain the image features. As the layers get deeper, the extracted information becomes more abstract, and the neurons change from simple shapes to 'higher' information. This means that the original signal (pixel dots in various colours) is first taken in, then the boundaries between different objects are extracted, then abstractions of the image are recognised, and finally objects are judged.

The weaknesses of convolutional neural networks are also evident. Due to the use of backpropagation, a large amount of data is required in order to be able to accurately extract features from the same class of images. In addition, the weakness of convolutional neural networks is that they have the property of translation invariance. Translational invariance, which means a slight change in the orientation or position of the same object, may not activate the neurons that recognise that object. If a neuron is used to recognise a car, its parameters will change as the position and rotation of the car changes. Training data augmentation as the most important factor in obtaining translation-invariant representations of images using convolutional neural networks (Kauderer-Abrams 2017). Finally, the presence of a pooling layer in a convolutional neural network can lead to the loss of a lot of very valuable information, as well as ignoring the correlation between the whole and the parts. The CNN discriminates in such a way that if the probability of several features of an item being presented at the same time is high, when the features are present at the same time, then this is the item being identified. For example, in the process of face picture recognition by CNN, after the CNN has used the eyes, mouth, nose on the face picture as the features for recognizing faces, if there is a picture on which the positions of the five senses are swapped, for example, swapping the positions of the nose and the mouth, then the CNN will still determine it as a face picture because this picture has the features of the mouth and nose in a human face.

3.3 Architecture and hyperparameters

3.3.1 Random Forest Algorithm

The experiment uses the random forest classifier from the *sklearn* library for training and classification of Fashion_MNIST dataset. The processed training set was trained under default parameters in the experiments and it can be seen that for the random forest algorithm, a high accuracy can be easily obtained with the default hyperparameters. However, we still chose to tune following three parameters in the hope of geting a higher accuracy. The process of tuning the hyperparameters was done by K-fold cross validation and the model selection function *GridSearchCV*, also from the sklearn library, was used to search for the best combination of parameters. The K-fold cross validation is one of the most commonly used methods for classifier model selection and error estimation, which refers to the method of dividing the training set into K subsets and then iterating to obtain the accuracy to evaluate the performance of each model. To improve the training and tuning time, it is only set 3-fold cross validation, and higher K values can be used if better model accuracy is desired.

First is the criterion of the algorithm, including "gini", "entropy" and "log_loss". It mainly refers to the measure of random forest in selecting the nodes of different decision trees. "gini" refers to the gini impurity of the nodes and is the default criterion for this classifier. The Gini coefficient criterion allows each child node to achieve a higher purity. The smaller the gini coefficient, the higher the purity and the smaller the uncertainty. "entropy" refers to the information gain. The information entropy criterion can determine the information brought by different nodes, and the more information, the more important the feature is. "log_loss" refers to the logarithmic loss function, whose mathematical method of selecting nodes is similar to the calculation of information entropy. Using the entropy as tree node splitting criterion is equivalent to minimizing the log loss between the true labels y_i and the probalistic predictions $T_k(x_i)$ of the tree model *T* for class *k* (Breiman, 1984).

The second hyperparameter that needs to be tuned is "n_estimators", which refers to the number of decision trees generated by the random forest and is one of the most important random forest hyperparameters. The number of decision trees is too small to reflect the difference between random forest and single decision tree, and it is easy to generate overfitting problem. However, too many decision trees will lead to longer training time, and the accuracy of the algorithm will stabilize when the number of decision trees reaches a certain level, thus wasting time and computational power, so it is important to choose an appropriate value. The default value in the algorithm is 100, and the range we choose here is specified to be around 100, which are 50, 100, 150 and 200.

The final parameter chosen is "max_leaf_nodes", which refers to the maximum number of leaf nodes. The tuning of the maximum number of leaf nodes can effectively prevent the decision tree from overfitting problems, and the algorithm builds the optimal decision tree within the maximum number of leaf nodes. The default parameter is "None", which means no restriction on it, but when the training set has more features, limiting the maximum number of leaf nodes can get better accuracy.

3.3.2 Fully connected neural network

Since the Fashion_MNIST dataset used for the experiments is similar to the MNIST dataset, the model structure of the MNIST dataset can be used as a reference for the initial construction of the multilayer perceptron model. The algorithmic model of the multilayer perceptron, keras, is derived from the tensorflow library, and the scikeras library is also used to make it compatible with other functions in sklearn. First of all, the experiments were performed by constructing the function "build_mlp" to help us quickly generate a model we need. The inputs to the function are the hyperparameters to be tuned, which are the activation function of the hidden layer and the learning rate of the gradient descent optimization algorithm. The multilayer perceptron model is built in the function, including an input layer, two hidden layers and an output layer. The purpose of the flatten layer is to flatten the 2D input into a 1D vector. For the hidden layer, it is often a difficult task because the nodes of the hidden layer are usually set for no particular reason and a larger number of nodes often means better performance, but longer training time. Since the Fashion_MNIST dataset has 784 input features, we can choose this value as the number of nodes for the first hidden layer, but based on the experience with the MNIST dataset, the experiments finally set the number of nodes for the hidden layer to 300 and 100. The final layer is usually determined by the nature of the classification problem. The Fashion_MNIST dataset has ten class labels, so experiment sets 10 as the number of the output layer nodes and use softmax as the activation function of the layer to convert the original output of the layer into a probability distribution of classes. At the same time, the experiments encapsulate the process of model compilation in the function. Since the learning rate is an important hyperparameter that needs to be tuned, the experiments choose gradient descent as the learning algorithm, while the choice of the loss function is usually determined by the nature of the problem. For a classification task, the probability distribution of the output layer can be scored using cross-entropy loss. Since the labels are in the form of indexes rather than one-hot vectors, the experiments utilize the "sparse_categorical_crossentropy" loss.

After defining the functions, the experiments first used the tuning results from the MNIST dataset, but since the data in the Fashion_MNIST dataset is more complex, the experiments increased the epoch to 30. The learning rate was set to "5e-2", which is the scientific notation form of 0.05. In the output of the training process, "loss" represents the loss rate of the training set, "accuracy" represents the accuracy of the training set, "val_loss" represents the validation set loss represents the loss rate of the training set, and "val_accuracy" represents the accuracy of the validation set. After visualizing the callback results (*Figure 3*), it is obvious that both "accuracy" and "val_accuracy" show an increasing trend, which indicates that 30 epochs are desirable.

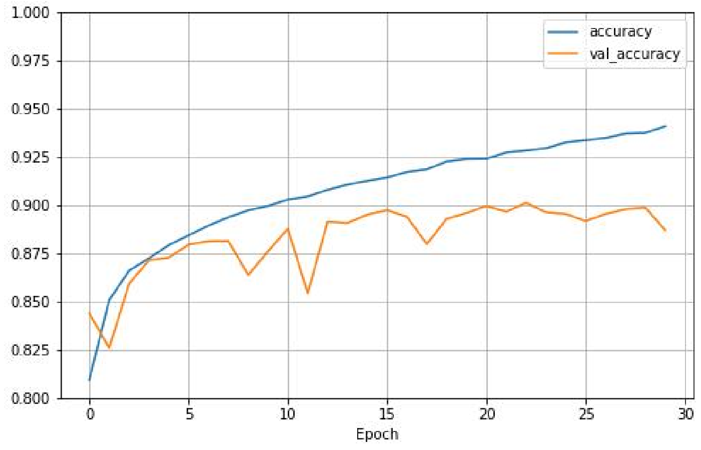

Figure 3: Visualization of training set accuracy and validation set accuracy over time

For multilayer perceptrons, the activation function of each layer is the most important hyperparameter for the model to achieve self-learning and training. It can be seen from the experiments without activation function that the model cannot perform prediction and classification without the activation function. Therefore, choosing the right activation function will be a key factor in whether the model can have higher accuracy. The range of activation functions chosen for tuning the multilayer perceptron is "relu", "sigmoid" and "tanh". The "relu" refers to the linear rectifier function, which is one of the more commonly used activation functions in artificial neural networks, defined as $f(x) = max(0, x)$, and its function image is shown in Figure 4 (Liu, 2017). In the neural network, the nonlinear output of the neuron after the linear transformation $\omega^T x + b$ is defined. This activation function can make the gradient propagation and backpropagation more efficient, and effectively avoid the gradient explosion and gradient disappearance problems. At the same time, the simple function calculation also makes the neural network more computationally efficient. The "sigmoid" function is also a common activation function defined by the formula $f(x) = \frac{1}{1+e^{-x}}$, which takes values in the range between 0 and 1 (Han & Moraga, 1995). Its function image is shown in Figure 5. The "sigmoid" function can effectively map a real number to the interval from 0 to 1 and efficiently perform binary classification problems. The function is smooth and easy to derive, so it works better when the difference in the data set is complex or not particularly large. However, as can be seen from the formula, the activation function is computationally intensive when back propagation is performed because it involves the derivation of the division method, and it is also prone to gradient disappearance. The "tanh" refers to the hyperbolic tangent of the hyperbolic function, which is defined by the formula $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$and takes values in the range between -1 and 1. Its function image is shown in Figure 6. Its biggest advantage is that it is a zero-centered function, and it is usually regarded as the optimization of the sigmoid function, but the activation function in the use of gradient disappearance problem (Kalman & Kwasny, 1992).

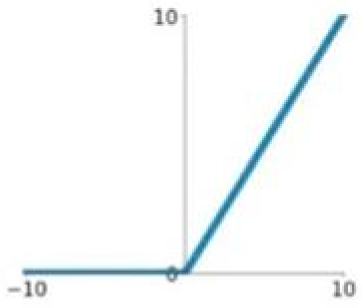

Figure 4: The image of "relu" activation function

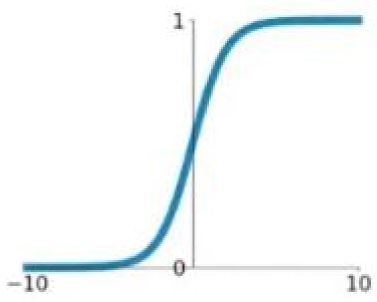

Figure 5: The image of "sigmoid" activation function

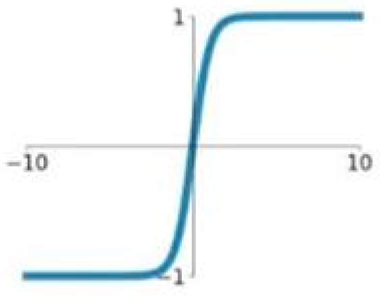

Figure 6: The image of "tanh" activation function

Since the experiments use a gradient descent optimization algorithm, choosing the appropriate learning rate is also a hyperparameter to focus on during the tuning of the multilayer perceptron model. The gradient descent algorithm is an iterative optimization algorithm that is usually used to find minima of differentiable functions. In gradient descent algorithms, the learning rate is the hyperparameter used to control the update rate of the algorithm. In general, a too small learning rate will lead to slow convergence, while a too large learning rate will lead to divergence or eventual loss function failure to converge, so choosing an appropriate learning rate will directly affect the training speed and results of the multilayer perceptron algorithm. The learning rates in 0.1, 0.01 and 0.001 have been chosen for tuning in the experiments.

3.3.3 Convolutional neural network

This experiment mainly uses the tensorflow.keras library to train and classify the Fashion_MNIST dataset using convolutional neural networks.
The model was set up with two convolutional layers with filter values of 32 and 64. to determine the optimal number of iterations, the model was trained 100 times on the processed training set with default parameters (kernel_size= (3,3), strides=1, learning_rate=0.01), and by plotting the accuracy-epoch, we found that when accuracy reaches a high value when epoch is 30, meanwhile, the learning efficiency of the model gradually decreases after thirty epochs. The accuracy of the model is about 90% for the test set when epoch is 30. It can be seen that for the convolutional

neural network, a high accuracy can be obtained with the default hyperparameters.

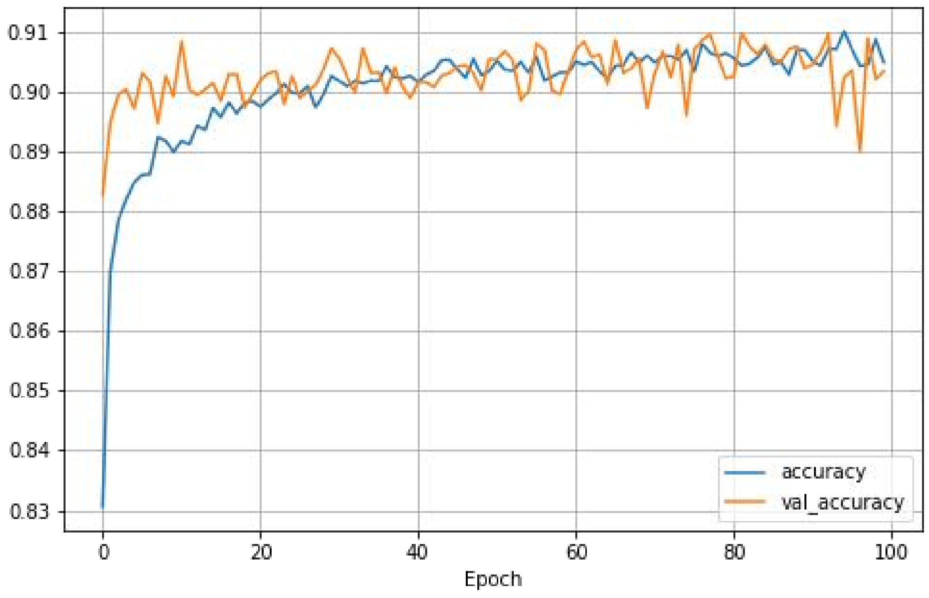

Figure 7: The graph of accuracy-epoch

However, we still choose to tune the following three parameters in the hope of obtaining higher accuracy. The process of adjusting hyperparameters is done by K-fold cross-validation, and the model selection function *GridSearchCV*, also from the sklearn library, is used to search for the best combination of parameters. K-fold cross-validation is one of the most commonly used methods for classifier model selection and error estimation, which refers to dividing the training set into K subsets and then evaluating the accuracy by iterating to obtain the accuracy of each model's performance by iterating to obtain accuracy. To improve the training and tuning time, only 3-fold cross-validation is set, and higher K values can be used if better model accuracy is desired.

The first hyperparameter is kernel size. A two-dimensional convolution filter like 3x3 always has a third dimension. The third dimension is equal to the number of channels in the input image. 3x3x1 convolution filters (with one black and white channel) are suitable for application on grayscale images, while 3x3x3 convolution filters (with three channels, rgb) are applied on colour images. Also, most useful features in an image are usually local, so it makes sense to take only a few local pixels at a time to apply the convolution. Sliding kernels over the whole image allows features that are distributed in multiple places in the image to be found and extracted by the same kernel. An additional benefit of using a small kernel rather than a fully connected network is that it can benefit from weight sharing and reduced computational costs. Since we use the same kernel for different sets of pixels in the image, when we convolve these sets of pixels, the same weights are shared between these sets of pixels. Since there are fewer weights than in the fully connected layer, we have fewer weights to back-propagate (Sabyasachi 2018).

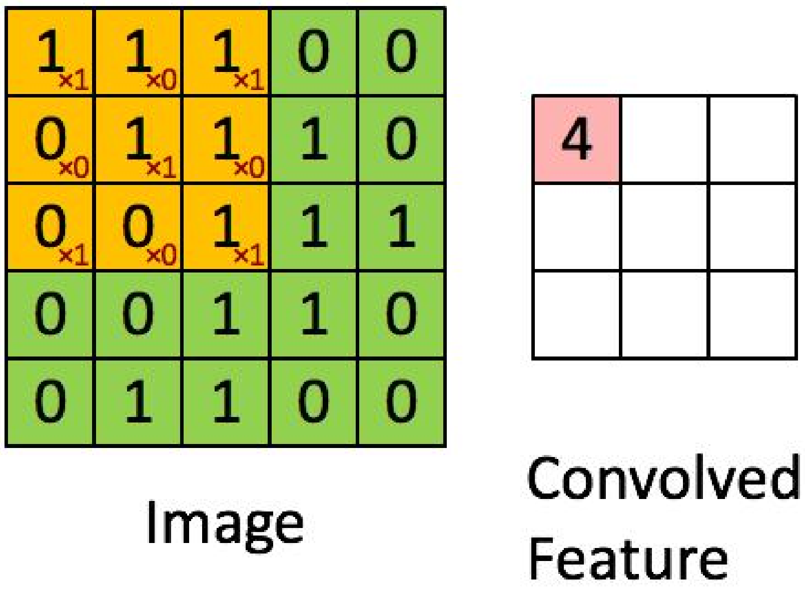

*Figure 8: A 3*3 2D convolution filter (Sabyasachi 2018)*

The second hyperparameter is stride. Stride is a component of a convolutional neural network, or a neural network tuned for compressing image and video data. stride is a parameter of a neural network filter that modifies the amount of movement on an image or video (Adit, 2016). For example, if the stride of a neural network is set to 1, the filter will move one pixel at a time. The size of the filter affects the amount of output that is encoded, so stride is usually set to an integer, rather than a fraction or decimal.

The third hyperparameter is learning rate. The learning rate controls how quickly the model adapts to the problem. Given the small change to the weights with each update, a smaller learning rate requires more training epochs, while a larger learning rate leads to rapid changes that require fewer training epochs. Too large a learning rate can cause the model to converge too quickly to a sub-optimal solution, while too small a learning rate can cause the process to bog down.

4. Results and discussion

Since all three algorithms use the 3-fold cross-validation, 'total time' in the tables for the combinations of hyperparameters shown below are the largest values of the combinations in 3-fold cross-validation for the sake of brevity of the graphs.

4.1 Random Forest Algorithm

For the random forest algorithm, based on a number of thirty iterations and the following three hyperparameter tuning:

- criterion = ['gini', 'entropy', 'log_loss']
- n_estimators = [50,100,150,200]
- max_leaf_nodes = [3000,4000,5000, None]

GridSearchCV came up with {'criterion': 'entropy', 'max_leaf_nodes': None, 'n_estimators': 200} as the optimal parameters, with an accuracy of 87.66%. However, in the case of image recognition using the default parameter values of the Random Forest algorithm, an accuracy of 87.71% was also achieved, which is even a little bit higher than 87.66% and not significantly different from the accuracy after tuning the parameters.

Random Forest Algorithm			
criterion	max_leaf_nodes	n_estimators	total time
gini	3000	50	23.5s
gini	3000	100	50.4s
gini	3000	150	1.2min
gini	3000	200	1.5min
gini	4000	50	25.4s
gini	4000	100	50.4s
gini	4000	150	1.3min
gini	4000	200	1.7min
gini	5000	50	25.7s
gini	5000	100	51.5s
gini	5000	150	1.2min
gini	5000	200	1.7min
gini	None	50	23.1s
gini	None	100	46.5s
gini	None	150	1.2min
gini	None	200	1.6min
entropy	3000	50	35.1s
entropy	3000	100	1.2min
entropy	3000	150	1.7min
entropy	3000	200	2.3min
entropy	4000	50	35.4s
entropy	4000	100	1.2min
entropy	4000	150	1.8min
entropy	4000	200	2.3min
entropy	5000	50	35.2s
entropy	5000	100	1.2min
entropy	5000	150	1.7min
entropy	5000	200	2.3min
entropy	None	50	33.5s
entropy	None	100	1.1min
entropy	None	150	1.7min
entropy	None	200	2.2min
log_loss	3000	50	0.2s
log_loss	3000	100	0.2s

log_loss	3000	150	0.2s
log_loss	3000	200	0.2s
log_loss	4000	50	0.2s
log_loss	4000	100	0.2s
log_loss	4000	150	0.2s
log_loss	4000	200	0.2s
log_loss	5000	50	0.2s
log_loss	5000	100	0.2s
log_loss	5000	150	0.2s
log_loss	5000	200	0.2s
log_loss	None	50	0.2s
log_loss	None	100	0.2s
log_loss	None	150	0.2s
log_loss	None	200	0.2s

Table 1: Random Forest Algorithm's hyperparameters combinations

According to the same conjecture, the best random forest node selection criteria for image recognition is judged by information entropy. Also, a higher number of nodes is thought to have a better prediction performance, so the final best number of nodes is given as 200, and it is believed that a higher number of nodes can get a higher accuracy rate if the training time is not considered. Surprisingly the result for the maximum number of leaf nodes is none, which indicates that there is no serious overfitting problem for the image recognition problem, so there is no particular need for the limitation of leaf nodes.

4.2 Fully connected neural network

For the fully connected neural network, based on thirty iterations and the following three hyperparameter tuning.

- "activation_function1": ["relu", "sigmoid", "tanh"]
- "activation_function2": ["relu", "sigmoid", "tanh"]
- "optimizer_lr": [1e-1, 1e-2, 1e-3]

GridSearchCV produced results with {'activation_function1': 'relu', 'activation_function2': 'sigmoid', 'optimizer_lr': 0.1} as the best parameters with an accuracy of 87.29%. When both activation_function1 and activation_function2 are set to None and optimizer_lr is set to 1e-1, the accuracy rate is only 10%. This shows that for fully connected neural networks, the setting of the activation function is far more important than the learning rate.

Fullyconnectedneuralnetwork			
activation_function1	activation_function2	optimizer_lr	total time
relu	relu	0.1	1.1min
relu	relu	0.01	1.1min
relu	relu	0.001	1.1min

relu	sigmoid	0.1	1.1min
relu	sigmoid	0.01	1.4min
relu	sigmoid	0.001	1.1min
relu	tanh	0.1	1.1min
relu	tanh	0.01	1.4min
relu	tanh	0.001	1.1min
sigmoid	relu	0.1	1.1min
sigmoid	relu	0.01	1.1min
sigmoid	relu	0.001	1.1min
sigmoid	sigmoid	0.1	1.4min
sigmoid	sigmoid	0.01	1.1min
sigmoid	sigmoid	0.001	1.4min
sigmoid	tanh	0.1	1.4min
sigmoid	tanh	0.01	1.1min
sigmoid	tanh	0.001	1.1min
tanh	relu	0.1	1.2min
tanh	relu	0.01	1.4min
tanh	relu	0.001	1.4min
tanh	sigmoid	0.1	1.2min
tanh	sigmoid	0.01	1.4min
tanh	sigmoid	0.001	1.4min
tanh	tanh	0.1	1.4min
tanh	tanh	0.01	1.2min
tanh	tanh	0.001	1.2min

Table 2: Fully connected neural network's hyperparameters combinations

The best activation function was expected to be "relu" and "tanh", but unexpectedly the second hidden layer "sigmoid" performed better than the first layer "relu". However, the performance of the second hidden layer "sigmoid" and the first layer "relu" are surprisingly matched to get a better training result. At the same time, it was thought that too large a learning rate would cause the loss function to fail to converge, but apparently the learning rate range chosen was reasonable, so even the maximum learning rate of 0.1 did not affect the final accuracy, but obtained a better performance.

4.3 Convolutional neural network

For the convolutional neural network, based on thirty iterations and the following three hyperparameter tuning:

- "ks": [2, 3, 4]
- "strides": [1, 2, 3]
- "lrs": [1e-2, 1e-3, 5e-3, 5e-4]

GridSearchCV produced results with {'ks': 4, 'lrs': 0.0005, 'strides': 1} as the best parameters with an accuracy of 89.58%. When both activation_function1 and activation_function2 are set to None and optimizer_lr is set to 1e-1, the accuracy rate is only 10%. This shows that for fully connected neural networks, the setting of the activation function is far more important than the learning rate.

Convolutionalneuralnetwork			
ks	lrs	strides	total time
2	0.0005	1	1.4min
2	0.0005	2	2.4min
2	0.0005	3	1.4min
3	0.01	1	1.5min
3	0.01	2	2.4min
3	0.01	3	1.4min
3	0.001	1	2.4min
3	0.001	2	1.4min
3	0.001	3	2.4min
3	0.005	1	1.5min
3	0.005	2	1.5min
3	0.005	3	1.4min
3	0.0005	1	1.5min
3	0.0005	2	1.5min
3	0.0005	3	1.5min
4	0.01	1	1.6min
4	0.01	2	2.4min
4	0.01	3	1.5min
4	0.001	1	2.4min
4	0.001	2	1.5min
4	0.001	3	1.5min
4	0.005	1	1.6min
4	0.005	2	1.5min
4	0.005	3	1.5min
4	0.0005	1	1.6min
4	0.0005	2	1.5min
4	0.0005	3	1.5min

Table 3: Convolutional neural network's hyperparameters combinations

The kernel size of 4 was the optimal parameter, which I was not expecting, because we thought that the smaller the kernel size, the more detailed the model would be, but instead 4 was the maximum value of kernel size in the tuning process. This led us to consider whether a kernel size of 5, 6, 7 or even more would result in higher accuracy. Unfortunately, however, we did not have enough time to try this because of the running time of the model and the deadline of the assignment.

The learning rate of only 0.0005 was also unexpected. Because, the general rule is "bigger batch size bigger learning rate" (Miguel 2017). Especially when the batch size of this model has reached 128.

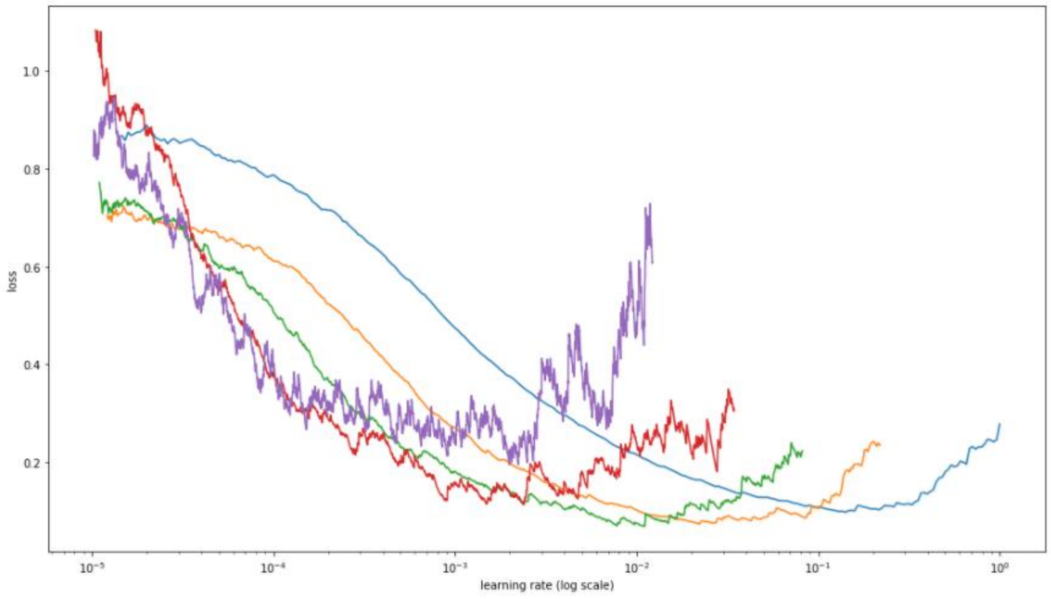

Figure 9: The relationship between learning rate error plotted using batches from 64 to 4 for the "cats vs. dogs" dataset (Miguel 2017)

The optimum value of 1 for stride is what I expected, as a stride of 1 means that the filter will only move one pixel at a time, allowing for more detailed extraction of features from the image.

5. Conclusion

As a result of the above work, we have the following findings:

- For the convolutional neural network, which is supposed to be the most commonly used neural network for image recognition, it does not seem to be as accurate as it should be in this assignment. I think this has to do with the batch size and the number of layers in *Conv2D*. Also, many parameters, such as activation function, pool size, have not been tuned yet. I believe that if these parameters can be optimised in the future, the accuracy rate will definitely be improved a lot.
- For machine learning, the physical environment in which it runs is also important. If you are running a machine learning model on your own computer, the lack of hardware such as a GPU can cause it to take too long to run, so you should run the program on a site such as Colab to ensure that you can get results as quickly as possible.
- The dataset for this assignment is not very large, so we split the dataset directly into a training set and a test set using *train_test_split*. However, in the future we may encounter a huge dataset, say 5 million, which would make this approach clumsy and costly in terms of trial and error. Therefore, we believe we can:
  1. Prepare small-scale samples, disregard data augmentation for the time being, and train the network directly after normal packaging, using small batches of samples to test for possible bugs in the network build until the

network can converge, to ensure the accuracy of the network build and to facilitate the localisation of problems when they arise subsequently.

2. After convergence of the small sample size, the model framework can be determined to be problem-free. Start to increase the sample size and train with large samples.

6. Reflection

As a result of the above work, we have the following reflection:

➢ In this assignment, we used *GridSearchCV* for the purpose of tuning the parameters. It does this by iterating through the group of each parameter.
  Advantages:
  - It is very simple to implement.
  - It can iterate through every combination and the results are more reliable.

  Disadvantage:
  - It is too time consuming and the run time increases geometrically if multiple parameters need to be adjusted or values added to each parameter, so it is generally not possible to try too many combinations of parameters with it.

  Since the computer used for training was not very powerful, the overly long tuning process resulted in inaccurate results, making the final test set results less than satisfactory. However, the purpose of this experiment and report is to demonstrate the understanding and tuning process of different algorithms and their choices when faced with image recognition problems, so the final test results do not affect the presentation of the report too much. For future work on tuning parameters, if multiple parameters need to be tuned or multiple values of a parameter need to be tried, then we will use Bayesian optimization as it takes into account the experimental result values corresponding to different parameters and is therefore more time efficient.

➢ The number of iterations also has a non-negligible effect on the time and accuracy of the run. Determining an optimal epoch, which can be achieved by looking at the inflection point in the accuracy-epoch curve (i.e., the point at which satisfactory accuracy has been achieved and the model starts to learn less efficiently in subsequent epochs), allows the highest possible accuracy to be achieved while reducing run time (as epoch participate throughout the process of learning, tuning parameters and validating the model). For example, if an algorithm is 90% accurate at 30 epochs, but 93% accurate at 100 epochs, it would be wise to sacrifice 3% accuracy to save the time of 70 epochs, and this is something we should maintain in the future.

7. Reference

Anguita, D., Ghelardoni, L., Ghio, A., Oneto, L., & Ridella, S. (2012). *ESANN 2012. The 20th European symposium on artificial neural networks, computational intelligence and machine learning. Proceedings - Bruges, Belgium from 25 to 27 April 2012.* Louvain-La-Neuve: I6doc.com.

Breiman, L. (1984). *Classification and regression trees*. Belmont, Calif. Wadsworth International Group.

Deshpande, A. (2016, July 29). A Beginner's Guide To Understanding Convolutional Neural Networks Part 2 – Adit Deshpande – Engineering at Forward | UCLA CS '19. Retrieved October 20, 2022, from https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Great Learning Team. (2020, February 19). Random Forest Algorithm- An Overview | Understanding Random Forest. Retrieved from GreatLearning website: https://www.mygreatlearning.com/blog/random-forest-algorithm/

Han, J., & Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. *Lecture Notes in Computer Science*, 195–201. https://doi.org/10.1007/3-540-59497-3_175

Hardesty, L. (2017, April 14). Explained: Neural networks. Retrieved from MIT News website: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

Kalman, B. L., & Kwasny, S. C. (1992, June 1). Why tanh: choosing a sigmoidal function. https://doi.org/10.1109/IJCNN.1992.227257

Kauderer-Abrams, E. (2017). Quantifying translation-invariance in convolutional neural networks. arXiv preprint arXiv:1801.01450.

Lavine, B. K., & Blank, T. R. (2009). Feed-Forward Neural Networks. *Comprehensive Chemometrics*, 571–586. https://doi.org/10.1016/b978-044452701-1.00026-0

Liu, D. (2017, November 30). A Practical Guide to ReLU. *Medium*. Retrieved from https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

Mathworks. (2022, October 19). What is a Convolutional Neural Network? - MATLAB & Simulink. Retrieved October 20, 2022, from https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html

Michaus, M. (2017, November 5). Visualizing Learning rate vs Batch size. Retrieved October 21, 2022, from https://miguel-data-sc.github.io/2017-11-05-first/

Sahoo, S. (2022, August 20). Deciding optimal kernel size for CNN | by Sabyasachi Sahoo | Towards Data Science. Retrieved October 21, 2022, from https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

Sheykhmousa, M., Mahdianpari, M., Ghanbari, H., Mohammadimanesh, F., Ghamisi, P., & Homayouni, S. (2020). Support Vector Machine Versus Random Forest for Remote Sensing Image Classification: A Meta-Analysis and Systematic Review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *13*, 6308–6325. https://doi.org/10.1109/jstars.2020.3026724

Taud, H., & Mas, J. F. (2017). Multilayer Perceptron (MLP). *Geomatic Approaches for Modeling Land Change Scenarios*, 451–455. https://doi.org/10.1007/978-3-319-60801-3_27

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *ArXiv:1708.07747 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1708.07747