COMP5318: Assignment 2

Image Classification Report

Group 102
SID1:490576560
SID2:520653377

Contents

1 ABSTRACT

This report outlines the methods of building a multi-class image classification on the Fashion-MNIST image dataset. Three machine learning algorithms are designed, implemented and evaluated for this image classification task. The three algorithms are K-nearest Neighbour (KNN), Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN). This paper discusses the importance of each algorithm and compares the differences between the algorithms/classifiers. With further exploring the impact of hyper-parameters of each algorithm and tuning each algorithm to its best performance, the final choice is based on factors including run time, interpretability, concreteness and overfitting.

2 INTRODUCTION

This report is to compare, evaluate and explore multiple machine learning classification algorithms by practising on a publicly released image dataset called Fashion-MNIST. This well-researched image dataset is considered to be an essential learning path for applying practical methods and exploring the understanding of machine learning algorithms. So, this study is aimed to provide the following two aspects on the purposes of study and significance of practising this project.

Specifically speaking, this task is given to experiment with the whole processes of a machine learning task that contains the data loading, data pre-processing, algorithm design, set-up, hyperparameter tuning and final modelling decision. Both theoretical knowledge and practical tools and skill application are conducted to solve real-world classification problems. The gap between theories and practical experiments needs to be addressed and researched. Each sub-task is achieved to evaluate the different classifiers and grasp the summaries of features and analysis. Real-world tasks may be heavy and not all of the algorithms can be tested due to the limited resources. Various analytical skills and tools are implemented through this study and in-depth learning of how various algorithms are realised in a real case is well-developed and integrated. Real-world tasks may be heavy and not all of the algorithms can be tested due to the limited resources. So, this task would be essential to the research on picking reasonable algorithms/models.

Broadly speaking, this project is aimed at learning how technical implementation meets the fast-growing demands of classification on images. In our study, online shopping with various clothing pictures would require more understanding of solving the issue of information explosion. Therefore, recognising the clothing types would become the benchmark for exploring and validating machine learning algorithms. In modern digital platforms, the need for image-based classification is still growing and the application of image classification is now widely seen, such as auto-driving, biological diagnosis and remote sensing. Further, prototyping of other image-based classification tasks would be explored and experimented with to expand the application of such areas.

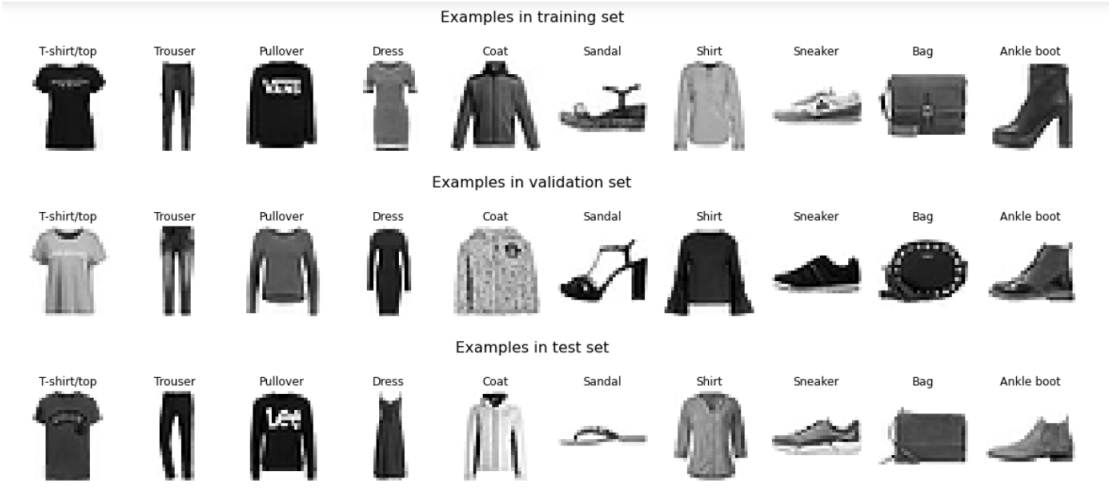
3 DATA

3.1 Data Description and Exploration

The original dataset is from an online fashion platform called "Zalando" (Han, Rasul, & Vollgraf, 2017). With research on how the fashion area is connected with technology, the original MNIST dataset with handwritten digits is provided. To further explore the algorithms and avoid overused simple examples, the Fashion-MNIST dataset comes out to replace the MNIST dataset.

The Fashion-MNIST data we use in this study is already split into a train and test set. The training set originally has 60000 images and the test set originally has 10000 images, with each image represented as a 28 x 28 pixel in grayscale. Thus, the training set has 3 dimensions that represent the number of examples, height and width respectively. Unlike other images that have an additional dimension representing the depth or channel Only one value is in each pixel and the image is black and white. The fourth dimension is omitted and will be added in CNN by looking at the data type of all examples, each pixel is a byte with the type shown as "uint8". A value from 0 to 255 would represent each pixel. The label set is a vector. Ten different classes are created for each image and the ten labels by the order are T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot. The ten labels are represented as indices in this study.

The below picture shows some examples from the training set, validation set and test set with 10 different classes on each set. It is clear that all images are in the normalised and centred size. The potential difficulties for this classification task could be classes with similar features. For example, the features of class pullovers and class shirts are visually similar according to the following examples. Sometimes, a sneaker could be similar to an ankle boot. Some dresses have long sleeves while others are not, which are similar to a T-shirt/top. A coat looks similar to some shirts.



(Figure1: Examples of a training set, a validation set and a test set)

3.2 Pre-processing

Data pre-processing is usually the first step to ensure data quality and data efficiency. Biassed data would finally result in poor performance and data is an essential factor related to the accuracy of models. Data pre-processing may solve issues, such as inconsistency, and turn the raw data into a more useful format. Various techniques may be applied according to the data domains.

In order to better decide the proper point to stop training when training, tuning and optimising neural networks, a validation set is created to compare distinct models in an unbiased approach and evaluate the hyperparameters of various models to avoid overfitting. Overfitting usually happens when the training data is too little or over-training models may fit the training data but have poor generalisation ability. So, a training set cannot deal with this issue and a validation set serves the purpose of generating a fair measure of a neural network. In this study, 10% of the original training data is used as the validation set. Therefore, the training set has 54000 image examples, the validation set has 6000 image examples and the test set has still 10000 image examples. The ten labels are equally distributed in the training set, the validation set and the test set, meaning each class has the same number of corresponding images.

As normalisation or scaling is typically recommended before designing and building models, it is crucial for the dataset in our study to be normalised. When unnormalised inputs are processed into the activation functions, large inputs would make gradients very close to 0 and flat regions in the domain may influence the training and learning process. Most weight initialization would require normalisation and it would avoid issues with gradients in neural networks.

The input features are scaled to the range of 0 to 1. This would ensure the pixels of all images are within a uniform range. The methodology of normalisation is as followed:

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

Where x' is the new value, x is the old value, x are all values of the attribute and max/min are the max/min values of the attribute

$$new\ pixel\ value = \frac{original\ pixel\ value\ -\ 0}{255 - 0} = \frac{original\ pixel\ value}{255}$$

In this study, because the features have the same scale, the new values can be simply divided by 255 according to the above formula. However, to avoid hard coded and ambiguity, sklearn.preprocessing.MinMaxScaler is formally used.

Dimension reduction such as Principal Component Analysis(PCA) and Non-negative Matrix Factorization (NMF) was considered, but not implemented. Some algorithms require a vector input, while others require a matrix input. This technique was well performed in certain algorithms. For example, after flattening an image matrix to a vector, in this study, the size of features is 784 which is 28*28. When the PCA was introduced, 14 * 14 = 196 features preserved around 96% of the information. This can

also be observed by inspecting the dataset that many features in an example are zero (black pixels). One significant advantage is the reduction of computational cost due to the much smaller size of features. However, the drawback is that it lost its interpretability. Especially for CNN which relies on extracting features of images, it will definitely degrade the performance of CNN. On the other hand, only using PCA processed data in some algorithms could bring some bias when evaluating other models trained by the original data. Thus, the original shape of the dataset is retained.

4 METHODS

After preparing the data, this section would focus on what machine learning models/algorithms are selected to perform classification tasks. A classification task could be regarded as an algorithm to categorise data into predefined classes. In this study, specific image classification is given and has multiple classes. Three different classifiers would be finally used for training and tuning.

4.1 Summary

The chosen three algorithms are KNN, MLP and CNN. A brief explanation of how these three classifiers are examined is shown below in detail:

K-nearest Neighbour Classifier

With doing initial experimentation on selecting basic machine learning algorithms, default/simple parameters are implemented on the full training set to check the accuracy and training time of four simple algorithms, namely K-nearest neighbour (KNN), Naive Bayes (NB), Decision tree (DT), and Random forest (RF). The table below describes the accuracy and training time for these four classifiers.

Classifier Type	Accuracy	Training Time (s)
KNN	0.8519	0.05
NB	0.5838	0.54
DT	0.8001	28.93
RF	0.8760	78.00

(Table1: A table with four classifiers showing their accuracy and training time)

Even though Random Forest performs best in this case, it has a relatively long training time. K-nearest neighbour Classifier has relatively high accuracy and the least training time from the above table. Therefore, the K-nearest neighbour is chosen.

KNN is the supervised algorithm to consider the ‘k’ nearest neighbours of a given data point. Then a class label can be voted to that data point. The distance between that data

point and other data points would be recorded to decide which data points are closest to a given data point. The k value would determine how many neighbours are checked to see the classification of a given data point. Several distance matrices could be used: such as Euclidean distance, and Manhattan distance.

Multi-layer Perceptron

By imitating the human brain operation process, neurons connected with each connection that has a weight are called neural networks. A multilayer artificial neural network is a part of deep learning, which mainly contains three layers: an input level, the hidden level and the output level. The simplest neural network is named perceptron. A Multi-layer Perceptron is a fully connected multi-layer neural network where a fully connected network means each neuron in its layer is directly connected with all neurons from the previous layer. Also, a Multi-layer Perceptron is the feedforward neural network, meaning each neuron would only receive input from the previous layer. A transfer function would be used for passing the weighted sum of the outputs from the previous layer. With defining the error function, weights are adjusted backwards to minimise the errors. A stop condition will be checked at the end of each epoch to avoid overfitting.

Convolutional Neural Network

Convolutional Neural Network is an extension of fully connected Neural Networks. It typically has the following structure: input layer for image values, convolutional layer, activation function to map features, pooling layer, and fully connected layers. The convolutional layers would extract features by the filter and the pooling layers would gather information from the convolution layers. The fully connected layer would collect features and finally output the probabilities for each class. As CNN is similar to the MLP, the backpropagation algorithm is also applied.

4.2 Strengths and weaknesses

This section describes the theoretical knowledge about the selected three algorithms: KNN, MLP and CNN based on their strengths and weaknesses. Multiple factors, such as training/running time, interpretability, number of parameters and overfitting, are considered and would be used for further discussion to compare with experimentation results.

K-nearest Neighbour Classifier

Strength:
Given the simplicity of KNN, it can be easily implemented, applied and interpreted. The training time would be very short since the training process would store all training

data in memory. KNN only has a relatively few hyperparameters for tuning.

Weakness:
The predicting time would be generally much longer than the training time, which may be on the opposite side of actual needs. It means that KNN may need more memory and storage space. More data may take a longer time for computation and test time would require a longer time since each test image would need to compare with all stored training images. The KNN may not have expected performances when dealing with a high dimensional dataset(here, the image contains many pixels). The performance is very sensitive to the selection of the k value.

Multi-layer Perceptron

Strength:
MLP works well with large input data. The prediction time is quick after training the models. It works well with non-linear models. MLP is a universal approximator, meaning two hidden layers can represent any hypothesis with a small error. It works well with new techniques/solutions coming out such as activation functions, dropout and optimizers. Dropout is the solution for solving issues of overfitting.

Weakness:
Computation is generally costly and time-consuming. The tuning process requires many hyperparameters, such as the number of neutrons in the hidden layer, the number of hidden layers, learning rates and the number of epochs. It requires careful tuning as they can get stuck in a bad local minimum.

The learning rate is sensitive to model performance and sometimes only local optimums could be found. However, the global minimum may cause the problem of overfitting. The vanishing gradient could be a potential problem in Neural Networks using the Sigmoid function. Now, an alternative activation function ReLU could solve such an issue. Finding the optimal architecture is still under exploration due to a lack of theory.

Convolutional Neural Network

Strength:
CNN is designed for image data but can be applied to other data types. The filter would extract the features of images. CNN is designed to recognize visual patterns directly from pixel images with minimal pre-processing. Then, CNN can recognize patterns with high variability, e.g.handwritten characters. CNN are more robust to distortions and geometric transformations such as shifting. The pooling layer can reduce the number of parameters and help with overfitting. It can also improve the computation speed.

Weakness:
It is hard to determine the CNN architectures with various factors, like complexity, runtime, and the number of layers. The computational cost may be relatively high for CNNs due to the need for a great GPU to speed up the computation in parallel.

The gradient descent algorithm used for training would sometimes result in a local minimum. The boundary information of images may get lost but this may be improved by adding padding. It is hard to tune the hyperparameters of CNN.

4.3 Architecture and hyperparameters

This section would talk about the architectures built for each algorithm and how hyperparameters are selected for tuning purposes in order to have better performances for each classifier. Search methods are discussed in the following details:

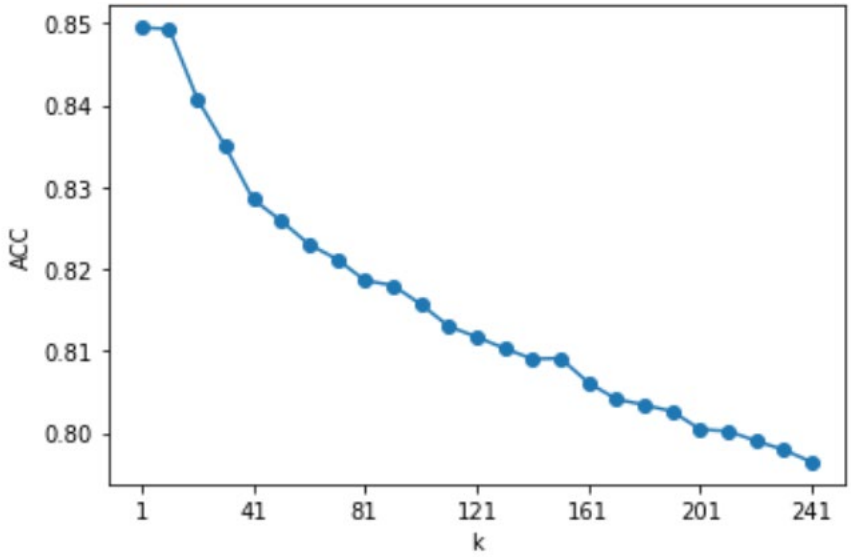
Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. If the folder is set to 10, each model is trained 10 times on 90% of the training set with each different parameter combination. Considering the running time due to a lot of parameter combinations as well as an innate long training time of a particular algorithm such as neural networks, in this study, instead of using sklearn.model_selection.GridSearchCV, we rewrite the search method using sklearn.model_selection.ParameterGrid to pass each parameter combination. This method trains a model on the training set and records the score on the validation set once with each parameter combination. Although setting CV = 1 using StratifiedKFold might work, given that we have already split the full training set, the full training set passed to StratifiedKFold would generate another validation set which is different to the one we have, which means we may not be able to control the randomness and the identity.

K-nearest Neighbour Classifier

As discussed in section 4.2, we compared different algorithms from the first 6 weeks with default parameters. Then we selected KNN. KNN is the simplest model in this study, with fewer parameters. From sklearn, a set of parameters can be tuned, such as algorithms with leaf size, metric and weight. The most important one is K. Besides, we also choose p representing the power parameter for the Minkowski metric and weights that considers weight in a vote.

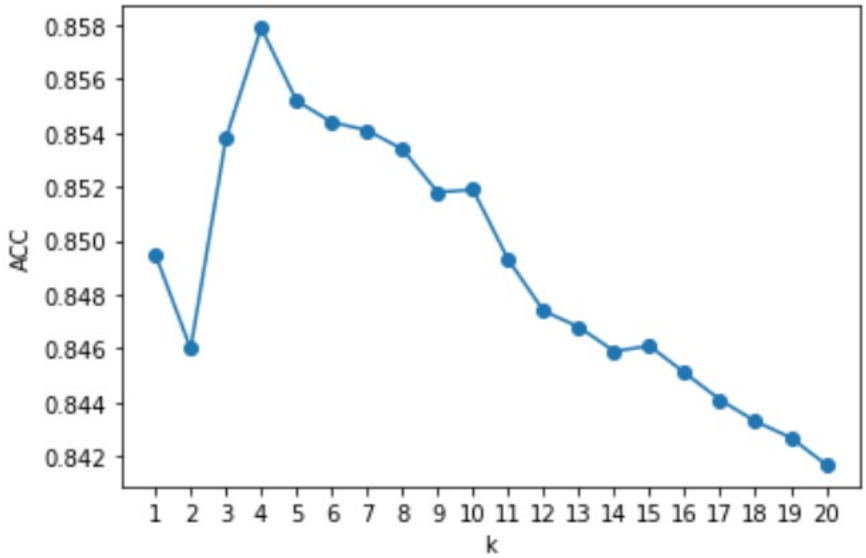
Smaller K and bigger K may bring bias and make the model less accurate. Empirically K should be less than the square root of the number of examples, usually use 10 in business. P = 2 represents the l2 norm while p = 1 using the l1 norm. Basically, the l1 norm is harder to calculate than l2, which means a longer evaluation time. The uniform weights mean all points are weighted equally, while the distance weights mean a closer point has more influence.

Widely used p and weights are $p \in [1, 2]$ and weights $\in$ ["uniform", "distance"]. For K, to determine a rough trend of the accuracy with different k, we first calculate the accuracy every 10 with different k, until k = 245 (sqrt(examples)), e.g., $k \in [1, 11, 21, ..., 241]$.



(Figure 2: Accuracy Versus k value in range 1 to 241)

The graph depicts the trend of accuracy with the changing value of k in the range from 1 to 241. The consistent decline trend of accuracy induces that greater k values generate worse results. Then we narrow the K range to 20. As shown in this figure, we choose k $\in$ [3, 5, 9] as it is better to choose an odd value to avoid a tie and randomness.



(Figure 3: Accuracy Versus k value in range 1 to 20)

As discussed before, the condition is limited for grid search with cross validation. Especially in KNN, if the l1 norm is applied, the evaluation time would be extremely long. Thus, standard grid search is utilised for tuning. l2 different combinations of parameters are defined. Each combination is trained on the training set and tested on the validation set once. Then the best parameter set is found based on the records.

Multi-layer Perceptron

One setback of the neural network is that there is no theory or guide to support how to build the architecture such as the number of hidden layers and the number of neurons in each layer. Empirically, the architecture should be built on an existing model that performs well in similar or related tasks. In this study, because the dataset is quite similar (the same input shape and the same number of labels) as the one in the tutorials, we could have easily chosen the same architecture. However, to make it reasonable and convincing, we build the architecture from a theoretical point of view.

First, the numbers of layers need to be settled. Apart from the input layer and output layer, the number of the hidden layer can be a variable. As discussed in the lecture, any function can be approximated to arbitrary small errors by a network with two hidden layers. Considering the training and evaluation in a feasible time, two hidden layers are suitable for this task.

Since the hidden layer (dense) requires a vector input, the first layer would flatten the input. For numerical attributes, basically one neuron per attribute. Here in our study, we have 28 * 28 = 784 attributes for each example. Thus, the number of neurons in the input layer should be 784.

For the output layer, since we have ten classes, the number of neurons in the output layer is 10. Because of the classification task, the activation function for this layer is the Softmax function that converts the raw outputs of this layer into a probability distribution over the classes.

Next, for the hidden layers, Sigmoid is the most widely used as the transfer function. We simply set most parameters by default as well as the basic Stochastic Gradient Descent (SGD) learning algorithm. Since our labels are in index form rather than encoded as one-hot vectors, as we discussed earlier, we utilize the sparse_categorical_crossentropy loss.

The number of neurons in the hidden layer is a hyperparameter that can be tuned. However, it is impossible to explore all settings. To find the reasonable number of neurons for hidden layers, our strategy is to perform a simple search and observe the trend of the numbers of neurons with respect to Mean Square Loss (MSE). We set the maximum epochs for this simple one to 50, and the stopping condition is the errors of the last five epochs plateaus within a predefined criterion. Empirically, the number of neurons should be between the size of the input layer and the size of the output layer. For the first hidden layer, we set n1 $\in$ [100, 200, 300, 400, 500, 600], and n2 = 50.

Neurons	100	200	300	400	500	600
Loss	0.4257	0.4294	0.4352	0.4377	0.4376	0.4369
Stopping epoch	31	30	29	28	28	28

(Table2: neurons variation with results of loss and epochs when meeting stop conditions on the first layer)

It shows that when the number of neurons in the first layer is 100, the loss is minimal. Although the epoch is slightly larger, which may enhance the performance, the training time is much faster because of the lesser neurons. To keep the model small, we choose 100 as the first number of neurons of the hidden layers.

Then with 100 settled for the first hidden layer, we set n2 $\in$ [20, 40, 60, 80].

Neurons	20	40	60	80
Loss	0.4352	0.4277	0.422	0.4261
Stopping epoch	34	31	31	31

(Table3: neurons variation with results of loss and epochs when meeting stop conditions on the second layer)

We select 60 as the neurons of the second hidden layer, as it achieves the best result with fewer epochs, and the training time is similar.

After settling the architecture, the hyperparameters to be tuned would be discussed. For the activation function, the sigmoid and the tanh have some limitations such as Vanishing Gradient Problem that can be overcome by ReLU. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments and sometimes works better than SGD in some tasks. The learning rate of the optimizer is also important. A small learning rate would slowly converge and increase the computational cost, while a larger one might overshoot the local minima.

As discussed, we use basic grid search without cross validation. The parameters set is that activation function $\in$ ["relu", "sigmoid", "tanh", None], optimizer $\in$ ["sgd", "Adam"] and learning rate $\in$ [0.1, 0.01, 0.001]. There are 24 different combinations in total. Each model is trained on the training set and scored on the validation set, the maximum epoch is set to 30. The random seed of the kernel_initializer is also set to avoid bias from the initialised weights that have an impact on the local minimum.

After the best parameters are found with the predefined architecture, the number of epochs may bring overfitting or underfitting. We set the maximum epochs to 30 and observe the accuracy with different epochs. Based on the trend, a reasonable epoch can be set.

Convolutional Neural Network

The structure of the CNN is similar to the structure of MLP. Thus, the architecture of the network needs to be settled first.

The input shape needs to be specified. As standard CNNs expect a four-dimensional array for the input and the last dimension indicates the number of channels, the input shape should be (28, 28, 1) due to the dataset that consists of grey images, which means the channel is 1. The extra 1 dimension would be added and passed when training.

Although usually the CNN performs well, the theory of determining the architecture, such as the number of convolutional layers, and the number of filters, is unclear. Sometimes it is determined by experiment and experience. Given that the condition is limited, one feasible and useful way is to choose a similar architecture from the tutorial. Therefore, two conv and pool blocks with 32, and 64 (empirically it should be exponentiation of 2) filters respectively, followed by a simple fully-connected layer, are chosen.

The ReLU would be selected as the activation function since it is simpler than sigmoid and can be faster in computation. Some advantages such as simplification of gradient, no upper bond, no saturation and less vanishing gradient would make ReLU become the first choice of the activation function. Max Pooling is chosen due to the distribution of pixels in each image, i.e. the features of images are presented as black (255) pixels while the background is white (0) pixels. The "same" padding is introduced to solve the problem of incompatible size between the kernel and the feature map. The softmax function converts the raw outputs of this layer into a probability distribution over the classes. A dropout layer is added to avoid overfitting and the loss function utilises sparse_categorical_crossentropy since labels are in the index.

The filter size, strides size, and learning rate are chosen and would be tuned using the standard grid search method. For filters with even size, distortions across the layers could happen without symmetry brought by an odd-sized filter. Thus, we set filter size $\in$ [(3, 3), (5, 5)]. For strides, bigger ones produce smaller feature maps. We set strides $\in$ [(1, 1), (2, 2)]. The learning rate of the optimizer cannot be bigger due to the overshooting problem. The impact has been discussed and this time we set the learning rate $\in$ [0.01, 0.005, 0.001].

We have defined 12 different combinations of the parameters. Considering the much longer training time than algorithms before, we set the epoch to 10 in the grid search step. Each model is trained on the training set and scored using the validation set. The same strategy in MLP is used to find a reasonable epoch avoiding the overfitting or the underfitting for the final model after the best parameters are found with the predefined architecture.

5 RESULTS AND DISCUSSION

This section would mainly talk about the tuning results and discuss the results and trends based on some performance measures. Our prediction on these algorithms would be compared with the practical results and the methods we used would be analysed to see if they matched the expectations.

K-nearest Neighbour Classifier

With tuning the parameters, the results of each parameter set are as below:

Parameter Sets	Score	Training time	Validation time
n_neighbors= 3, p= 1, weights: uniform	0.8622	0.05	40.38
n_neighbors= 3, p= 1, weights: distance	0.8653	0.05	40.33
n_neighbors= 3, p= 2, weights: uniform	0.8583	0.05	2.81
n_neighbors= 3, p= 2, weights: distance	0.8602	0.04	2.68
n_neighbors= 5, p= 1, weights: uniform	0.8603	0.05	40.39
n_neighbors= 5, p= 1, weights: distance	0.8622	0.04	40.23
n_neighbors= 5, p= 2, weights: uniform	0.855	0.05	2.92
n_neighbors= 5, p= 2, weights: distance	0.8577	0.05	2.68
n_neighbors= 9, p= 1, weights: uniform	0.8628	0.05	40.42
n_neighbors= 9, p= 1, weights: distance	0.8645	0.04	40.57
n_neighbors= 9, p= 2, weights: uniform	0.8512	0.05	3.06
n_neighbors= 9, p= 2, weights: distance	0.8533	0.05	2.89

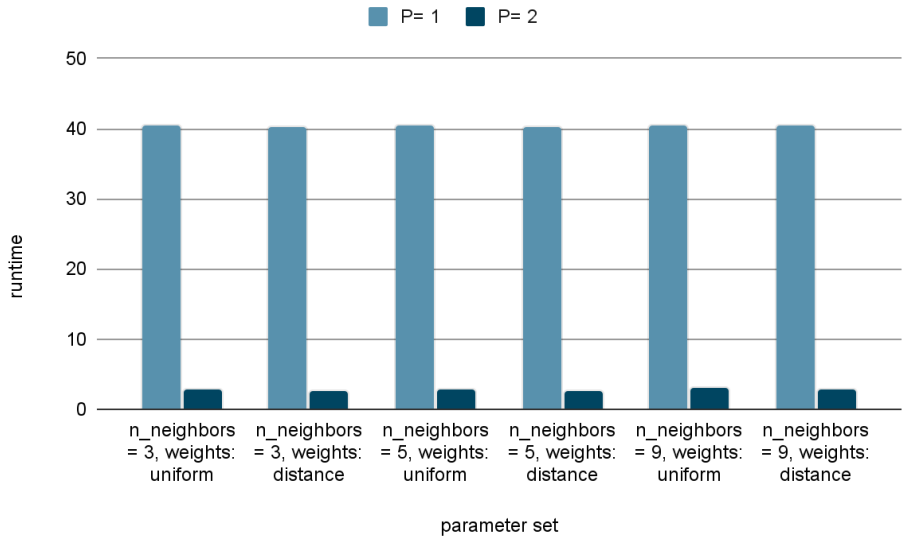
(Table4: A table of hyperparameter combination with their responding result scores)

From the table above, the best hyperparameter set achieves the score of 0.8653 with the parameter combination: n_neighbors= 3, p= 1(Manhattan distance) and weights: distance.

The p, which is the distance matrix, would influence the runtime by checking the results. To better compare and check the difference of runtime between choices of p, the figure below only chooses p as a variable and ensures other parameters in tuning are consistent. When p equals 1(Manhattan distance matrix), the runtime is larger than choosing p equals 2 (euclidean distance matrix). This is mainly because computing the l1 norm is harder than the l2 norm usually. Models with p = 1 perform better than p = 2 in this task.

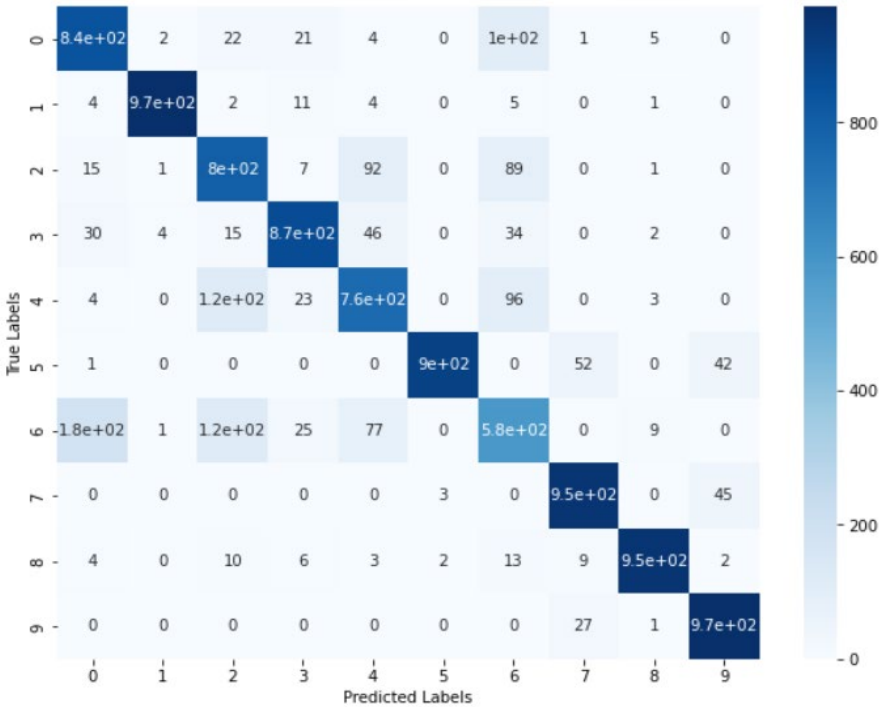
With other parameters fixed, models with distance weights perform better than ones with uniform weights, which means weighted votes can reduce the error and make the model more convincing.

For three different k, no specific trend can be observed. However, as discussed before in section 4.3, the higher the k, the lower accuracy. This is mainly because of the distribution of the examples.



(Figure4: p versus runtime on several parameter sets)

According to the figure below, the confusion matrix could provide information about which class has confusion on classification. By visually checking the lighter cell on the diagonal line, it means that classification has the largest errors in figuring out results between T-shirt/top and shirt classes. The coat label also has some confusion with the pullover and the shirt.



(Figure5: confusion matrix for a KNN model)

Multi-layer Perceptron

First, the results of the parameter combination are shown in the table below.

No.	Parameter Sets	Score	training time (s)	validation time (s)
1	activation_function: relu, optimizer: sgd, optimizer__lr: 0.1	0.872	49.03	0.31
2	activation_function: relu, optimizer: sgd, optimizer__lr: 0.01	0.8833	49.13	0.25
3	activation_function: relu, optimizer: sgd, optimizer__lr: 0.001	0.8488	48.98	0.25
4	activation_function: relu, optimizer: Adam, optimizer__lr: 0.1	0.198	52.16	0.25
5	activation_function: relu, optimizer: Adam, optimizer__lr: 0.01	0.8577	52.98	0.26
6	activation_function: relu, optimizer: Adam, optimizer__lr: 0.001	0.8927	52.72	0.25
7	activation_function: sigmoid, optimizer: sgd, optimizer__lr: 0.1	0.8867	51.64	0.25
8	activation_function: sigmoid, optimizer: sgd, optimizer__lr: 0.01	0.8455	51.55	0.26
9	activation_function: sigmoid, optimizer: sgd, optimizer__lr: 0.001	0.6782	51.52	0.26
10	activation_function: sigmoid, optimizer: Adam, optimizer__lr: 0.1	0.4063	54.35	0.26
11	activation_function: sigmoid, optimizer: Adam, optimizer__lr: 0.01	0.8545	53.68	0.25
12	activation_function: sigmoid, optimizer: Adam, optimizer__lr: 0.001	0.894	53.34	0.26
13	activation_function: tanh, optimizer: sgd, optimizer__lr: 0.1	0.8873	50.07	0.25
14	activation_function: tanh, optimizer: sgd, optimizer__lr: 0.01	0.8873	50.41	0.25
15	activation_function: tanh, optimizer: sgd, optimizer__lr: 0.001	0.8467	50.21	0.25

16	activation_function: tanh, optimizer: Adam, optimizer__lr: 0.1	0.1	57.98	0.27
17	activation_function: tanh, optimizer: Adam, optimizer__lr: 0.01	0.7078	57.59	0.26
18	activation_function: tanh, optimizer: Adam, optimizer__lr: 0.001	0.8857	56.22	0.27
19	activation_function: None, optimizer: sgd, optimizer__lr: 0.1	0.8398	52.35	0.26
20	activation_function: None, optimizer: sgd, optimizer__lr: 0.01	0.8512	57.59	0.26
21	activation_function: None, optimizer: sgd, optimizer__lr: 0.001	0.8485	52.61	0.26
22	activation_function: None, optimizer: Adam, optimizer__lr: 0.1	0.7832	55.43	0.26
23	activation_function: None, optimizer: Adam, optimizer__lr: 0.01	0.7968	55.06	0.26
24	activation_function: None, optimizer: Adam, optimizer__lr: 0.001	0.8475	54.47	0.27

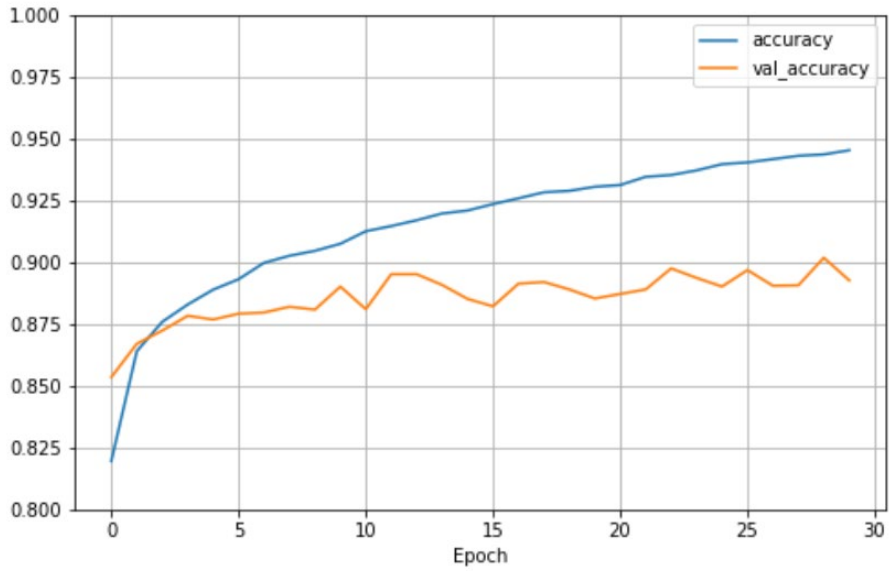
(Table5: parameter combination with relative scores, training and validation time on MLP)

The best score shown in the table above is 0.894, which contains the parameter set: activation function: sigmoid, optimizer: Adam and optimizer_lr: 0.001(here, shown in bold cells).

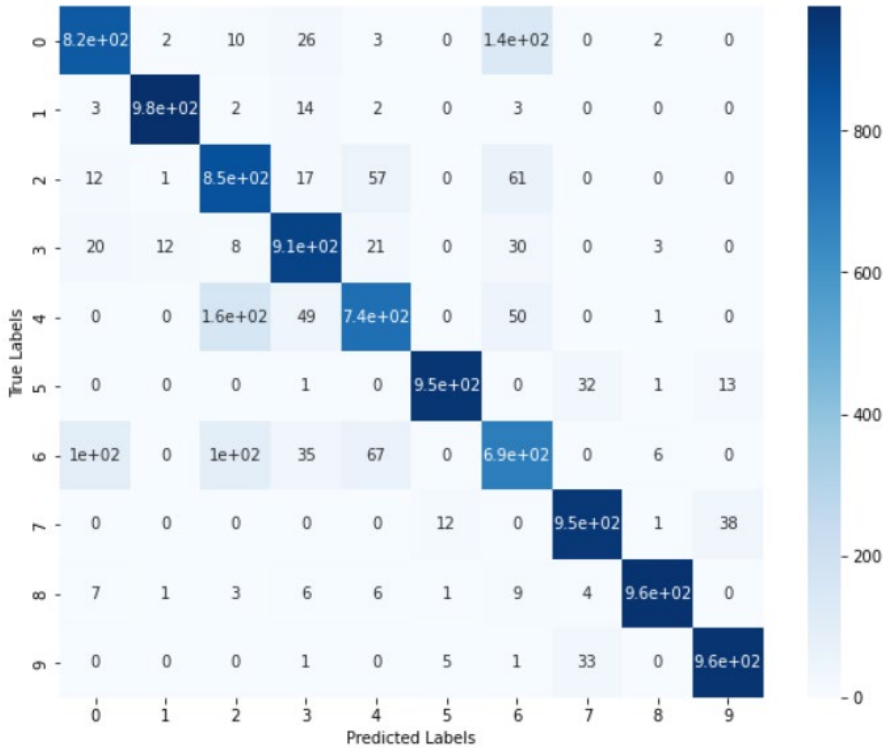
From the results, some interesting observations can be obtained. Models without an activation function usually perform worse. Consistently, activation is important to neural networks. Adam is more sensitive to the learning rate than SGD. Adam performs well with a proper learning rate, while SGD is more robust when the learning rate changes. However, SGD usually requires less training time than Adam. ReLU usually generates a good model with high accuracy in this task, and the training time is the fastest, which means higher efficiency.

More epochs might not result in higher performance on the test set, but the accuracy of the training set keeps increasing, which means the generalization error is high.

The figure below expresses the trend of accuracy as the number of epochs varies. The accuracy fluctuated after epoch = 10. That is the reason the epoch is set to 10 for the final model to keep it small with a good performance.



(Figure 6: Epoch Versus accuracy on MLP)



(Figure 7: confusion matrix for an MLP model)

According to this confusion matrix on MLP, it could provide the distribution of correctness based on a matrix. By visually checking the lighter cell on the diagonal blue line and the darker cell on other cells, the class pullover has some confusion with the class shirt. The coat label also has some confusion with the pullover.

Convolutional Neural Network

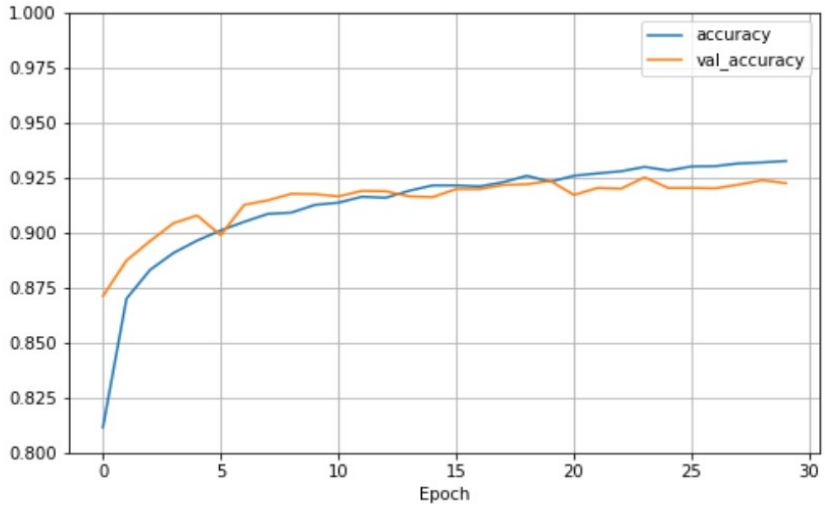
Parameter sets	Score	training time(s)	validation time(s)
kernel_size(3,3), optimizer__lr 0.01, strides(1,1)	0.8717	130.88	0.54
kernel_size(3,3), optimizer__lr 0.01, strides(2,2)	0.8602	43.34	0.31
kernel_size(3,3), optimizer__lr 0.005, strides(1,1)	0.8988	130.16	0.54
kernel_size(3,3), optimizer__lr 0.005, strides (2,2)	0.889	42.69	0.31
kernel_size(3,3), optimizer__lr 0.001, strides (1,1)	0.9175	138.12	0.6
kernel_size(3,3), optimizer__lr 0.001, strides (2,2)	0.889	45.52	0.31
kernel_size(5,5), optimizer__lr 0.01, strides(1,1)	0.8502	136.82	0.61
kernel_size(5,5), optimizer__lr 0.01, strides(2,2)	0.8483	44.92	0.32
kernel_size(5,5), optimizer__lr 0.005, strides(1,1)	0.8833	141.77	0.61
kernel_size(5,5), optimizer__lr 0.005, strides(2,2)	0.8688	43.64	0.32
kernel_size(5,5), optimizer__lr 0.001, strides(1,1)	0.9137	140.6	0.61
kernel_size(5,5), optimizer__lr 0.001, strides(2,2)	0.8857	43.86	0.31

(Table 6: parameter combination with score, training and validation time on CNN)

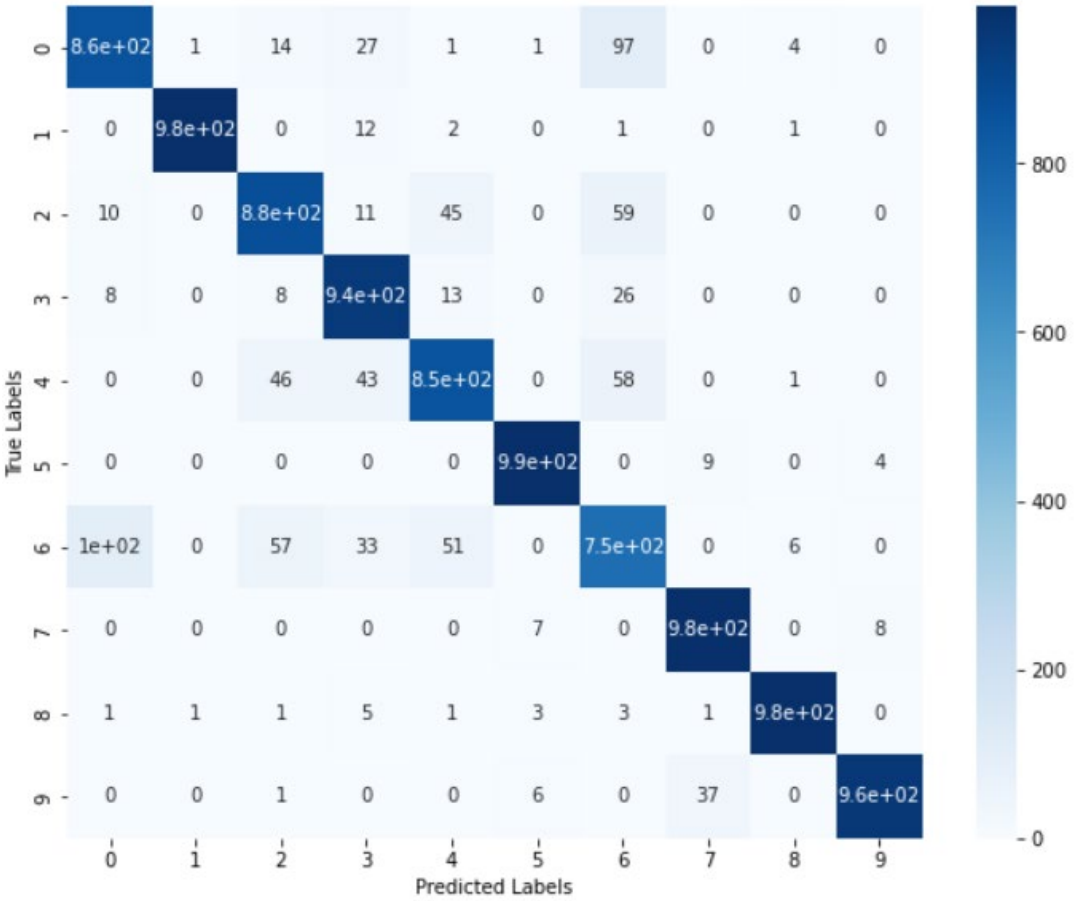
From the table above, the best parameter combination based on accuracy is kernel size of (3, 3), learning rate 0.001 and strides(1,1), showing the highest accuracy is 0.9175. All of our parameter combinations are in a good state of performance and there is no really low accuracy.

Unlike the MLP, the trends between these parameters are clear. Kernel size (3, 3) always performs well in terms of accuracy and training time with other parameters fixed. The reason might be that more details are caught by a smaller kernel. Strides size (1, 1) always performs well with other parameters fixed. A higher stride means a smaller and less detailed feature map. On the other hand, smaller strides mean longer training time. For the three learning rates, the accuracy increases as the learning rate go down.

The number of epochs has an impact on the model. The figure below exhibits the trend of accuracy when the epoch varies from 1 to 30. As the accuracy of a validation set fluctuates since epoch = 12, more epochs are not helpful to reduce generalization error.



(Figure 8: Epoch Versus accuracy on CNN)



(Figure 9: confusion matrix for a CNN model**)**

According to this confusion matrix on CNN, the shirt label is still facing the confusion condition. The coat label is also confused by this CNN model. All three algorithms have

mainly similar confusion on shirt class, coat class and pullover class. Even though better models could reduce the chances of errors, at least we know their limits on some specific classes from the confusion matrix.

Summary of final models

Overall, three final models with their best parameter sets are chosen by tuning the hyperparameters on each algorithm and the performances on the test data set are shown below:

Classifiers with their best parameter sets	accuracy	Training runtime (s)	Evaluation time (s)
KNN	0.8597	0.07	Around 120
MLP	0.8864	24.62	Less than 0.01
CNN	0.9126	191.29	Less than 0.01

(Table 7: three algorithms with their best parameter combinations and their performances)

Based on the table provided above, the results basically agree with our expectations on section 4.2. CNN has the highest accuracy as we discussed before CNN does well in capturing image features and works well in image classification tasks. Although only half Trainable parameters than MLP, the CNN has the longest training runtime among the three models, which is also mentioned before that it requires more computational time in each epoch. KNN has the least training time and still has a relatively good performance, but the evaluation time is longer if using Manhattan distance. MLP also achieves a relatively satisfactory accuracy with nearly 89% accuracy and acceptable training time due to a very simple and small architecture as well as fewer epochs being chosen.

6 CONCLUSION

In this study, three algorithms: K-nearest Neighbour, Multi-layer Perceptron and Convolutional Neural Network are experimented on the Fashion-MNIST dataset serving the purpose of image classification. After intensive discussion and analysis, we conclude that all three classifiers can provide relatively good results after tuning hyperparameters and CNNs can achieve a higher accuracy rate with taking a comparatively more computation time. MLP has around 89% accuracy and the moderate runtime is generally acceptable in this task. Theoretically, CNN should be the first choice for an image classification task among the three models. However, it is hard to tell which model is the best in practice considering aspects such as simplicity, interpretability, runtime and accuracy. The practical result basically follows the theoretical knowledge of the three algorithms, which brings us a more thorough

understanding of the logic behind the algorithms.

One limitation of this study is that restricted resources, such as computation time and cost, may limit us from further exploring more possible factors/parameters and their outcomes. Some parameters may have similar or concurrent effects on the results and we may not conclude all of them. Sometimes a single factor may be affected by some hidden conditions. Another limitation is the method. Because we are restricted by experimental conditions, the method is not optimal. For example, we just specify the same epoch in the grid search. Thus, we do not know whether a relatively bad model would perform better than the best combination we have chosen with different epochs. Another example is data preprocessing. As we discussed, to eliminate bias, feature reduction was not introduced. As a result, it is not fair to simply compare the performance with the same input, as an algorithm may perform better with the suitable input.

The limitations show that we cannot guarantee the result is optimal either with all possible parameters that can be tuned or with the parameters combinations we choose. However, this experiment is a good start. Considering the future work for such image classification tasks, KNN is limited to modelling efficiency when higher dimensions are conducted, especially for RGB images with three channels. When tweaking the parameter ‘k’, KNN is highly sensitive. KNN hits the barrier of 86% accuracy in this dataset. To further explore this project in the future, more adjustments on MLP could be done by experimenting with more hyperparameters, such as momentum, decay and batch size. More efficient search methods can be tested to tune parameters such as Bayesian Search.

7 REFLECTION

This study well reflects the experimentation of machine learning and deep learning algorithms on an image classification task. An important thing we learned from this project is that the architectures across different algorithms are based on extensive experimentation and theories may not be applied all the time. Sometimes, the theories come after the experiments. This, in turn, explains the difficulties to design the architectures and the importance to test models on various datasets. As we implement several algorithms and tune hyperparameters to search for the best performances in this project, experimentation is developed to better prepare us to further take a grasp of machine learning and deep learning.

8 BIBLIOGRAPHY

Han, X., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv.org*.

Jung, Y. (2018). Multiple predicting *K*-fold cross-validation for model election, *Journal of Nonparametric Statistics*, *30*(1), 197-215, DOI: 10.1080/10485252.2017.1404598

Xu, L., Xu, Q.-S., Yang, M., Zhang, H.-Z., Cai, C.-B., Jiang, J.-H., Wu, H.-L. and Yu, R.Q. (2011), On estimating model complexity and prediction errors in multivariateccalibration: generalized resampling by random sample weighting (RSW). *J. Chemometrics*, *25*, 51-58. https://doi-org.ezproxy.library.sydney.edu.au/10.1002/cem.1323