

# Assignment 2 – Group 54

**Tutor:**  
Henry Weld

**Group members:**  
Julio Correa (jcor7731)  
William Ryan (wrya6230)  
Maria Yu Qian Lim (mlim0345)

## Table of Contents

Abstract .....	1
1 Introduction .....	1
2 Previous work .....	2
3 Methods .....	4
3.1 Design approach .....	4
3.2 Pre-processing .....	5
3.2.1 Standardising .....	5
3.2.2 Wavelet Transform .....	6
3.2.3 Zero Component Analysis (ZCA) Whitening .....	7
3.3 Classifiers .....	8
3.3.1 Gaussian Naive Bayes .....	8
3.3.2 Random Forest .....	9
3.3.3 Convolutional Neural Network (CNN) .....	10
4 Results .....	13
4.1 10-fold cross-validation analysis .....	13
4.1.1 GNB .....	13
4.1.2 Random Forest .....	14
4.1.3 CNN .....	14
4.1.4 Summary .....	15
4.2 Execution time improvements .....	16
4.2.1 GNB .....	16
4.2.2 Random Forest .....	16

4.2.3	CNN .....	16
4.3	Results of the classification task on the actual test set.....	17
5	Discussion .....	19
6	Conclusion .....	20
	References .....	21
	Appendix.....	22
	Appendix i. Instructions to run the other algorithms code.....	22
	Appendix ii. Instructions to run the best algorithm code.....	24
	Appendix iii. Confusion matrices in large scale .....	25

## Abstract

*In the present work we follow a research approach driven by a simple rule: Our hypothesis is that a simpler classifier applied to image data, performs worse than a more sophisticated one. We chose to work with the Cifar-100 dataset. In this context, we defined a set of three of them, starting from the first and simple one which is Gaussian Naïve Bayes, moving on to a medium complex Random Forest Classifier, and finally CNN the complex classifier. Our main finding is that our hypothesis, at least for the given experimental setting followed, holds, namely, we cannot reject it. The previous statement indicates that with Gaussian Naive Bayes we achieved low accuracy 19%, with the Random Forest Classifier we got 34% average accuracy, and with CNN we achieved 48% accuracy, for the coarse labels.*

## 1 Introduction

The present report aims to implement three classifiers for an image classification task. The importance of this report lays down in a question that is absolutely timely: How can someone compare different types of classifiers? And in fact, there is no-one-size-fits-all solution. In fact, quite a few research work during the last 20 years has been developed in the field to address this question. In<sup>1</sup>, the authors develop an elegant work for comparing algorithms from the following families: 1. Tree, 2. Bayesian, 3. Lazy, and 4. Function. They compare them using the Weka framework and often, KNN is the best classifier for that setting. In the same article, they state that the problem of comparing classifiers has its most profound impact in practical applications, and they give an example of people not being from the field of computer science trying to classify some data and not being able to compare different methods in a meaningful way.

Therefore, following in part the method that they followed and drive by our own working experience, we decided to set the following pathway:

1. Define three pre-processing methods to apply to the dataset.
2. Define three classifiers to implement, each with different levels of complexity.
3. Establish a common experimental setting which allows to compare the accuracy and the execution time of each classifier.
4. Predict the categories on the test set
5. Compare the predictions using the confusion matrix and accuracy, recall, and F1-score.

The database is built upon twenty different categories of objects, and each image is 32x32 pixels distributed in three different colour channels. Some features of the dataset could be summarized as follows:

Table 1-1 Summary of datasets

Feature	Value	
Name	Cifar-100	
Number of images	training set	50,000
	test set	10,000
Dimension of each image	32 x 32x 3	
Number of categories	coarse label	20
	fine label	100
Average of all features	training set	121.94
	test set	122.24
Standard deviation of features	training set	68.39
	test set	68.63

**Summary of contents.** In section 2, we briefly describe each classifier and the preprocessing task as well, and I add some references to articles that I used to get the implementation done. In section 3, there is a description of the results for both the training and classification tasks. In section 4, there is some discussion on the main results of the present work, and finally, in section 5, there are conclusions and further research.

## 2 Previous work

Cifar-100 is a widely used data set for the study and investigation of various classification methods. Without going any further, there is a web page called 'Papers with code' <sup>2</sup>, which includes and lists a series of algorithms that have been used to classify images in this dataset. The interesting point about this resource is that it lists the algorithms and the articles where the different authors developed each of them. Therefore, we will build on this resource and try to explain what each one is about.

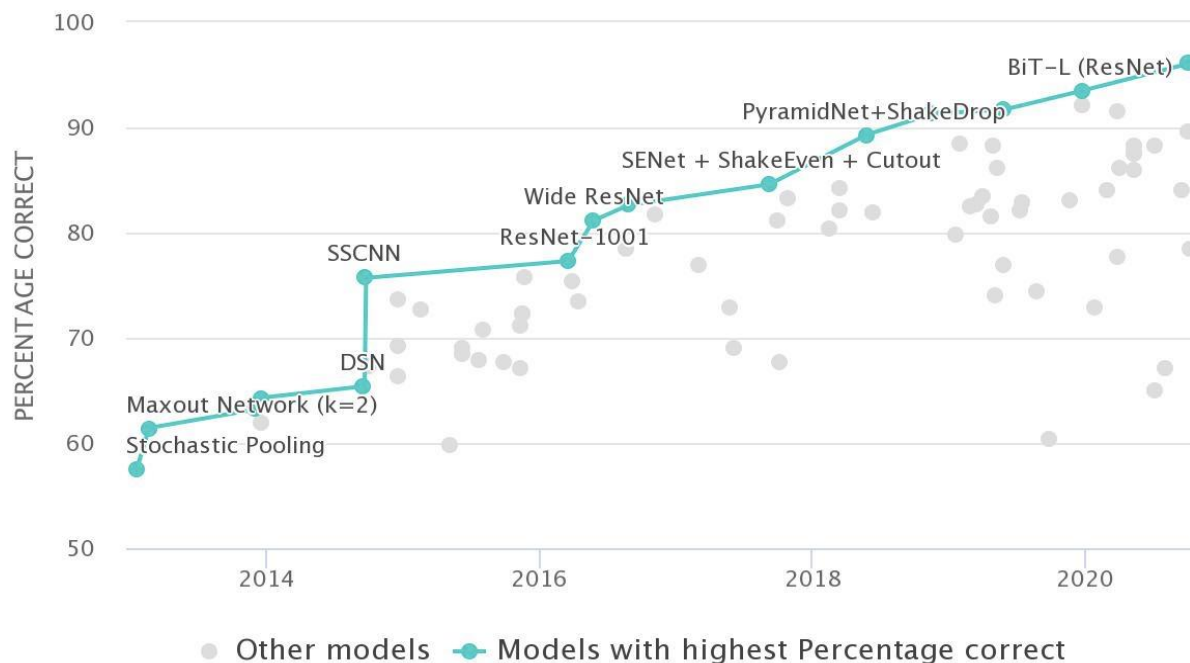


Figure 2-1 Plot of algorithms developed to classify Cifar-100 images

On the webpage, many more of them appear, though. The image illustrates what has happened in recent years and shows how the algorithms' accuracy has increased over the years. In general, this curve shows smooth growth, except in the period between 2014 and 2016, where a qualitative leap is seen. After that sharp transition, the smooth growth trend continues later.

**Source:** <https://paperswithcode.com/sota/image-classification-on-cifar-100>

We could say that the algorithms listed here belong to the following families of models:

- Networks (Deep learning)
- Trees
- Ensembles

However, those that predominate in terms of prevalence are those related to networks and deep learning. We can say that most of these articles mix the design and execution of this type of algorithm with processing techniques in GPU and TPU units. In particular, the first four owe their achievement in high accuracy to the fact that they can increase the number of experiments they carry out thanks precisely to the help of maximum power hardware. This is on top of the fact that the design process is very sophisticated as well.

A noteworthy algorithm for image classification was the convolutional neural network architecture called AlexNet which exceeded previous performances in the ImageNet Large Scale Visual Recognition Challenge 2012 by significant margins with a top-5 test error rate of 15.3

percent, beating the second place entry by 10.9 percentage points <sup>3</sup>. The success of AlexNet was further motivation for our choice of CNN as our third and most complex classifier.

Regarding the algorithms that we are proposing in this report, it is essential to note that ours are simplified versions and, in some cases, naive to approach the problem, particularly considering the application of the Gaussian Naive Bayes algorithm. From <sup>4</sup>, we know that it is not so appropriate for image classification because it does not consider the correlations between the features. In the case of images, this is critical because the pixels are correlated to each other. Nevertheless, the accuracy is not too low, considering that it is around 20% on the pre-processed data and, therefore, it is better than doing the classification randomly.

Concerning the Random Forest algorithm, in this set of previous works, there are more similarities. Some of them extend the ideas developed by Xu, B. et al.<sup>5</sup> in terms of introducing weighted mechanisms for the generated trees and, in this way, narrowing the search space of the algorithm. This idea is related to the concept behind MCMC weighted processes to sample efficiently in high dimension sampling spaces, such as random networks, when the analysis setting is Bayesian. More information on Exponential Random Graphs can be found in <sup>6</sup>.

However, our algorithm is different because it is a simple sklearn version of the classifier. However, our efforts to improve it are related to including parallelism, which means having threads running parallelly. With this algorithm, we can achieve a classification accuracy of around 35%, which is much better than the GNB case.

## 3 Methods

### 3.1 Design approach

The design process that we followed to carry out consisted of an approach based on implementing 3 algorithms, based on their level of technical complexity and the intuition behind them. Considering the previous statement, we selected the following three algorithms, ordered in terms of their complexity/easiness to understand:

1. Gaussian Naive Bayes
2. Random Forest
3. Convolutional Neural Network

Our intuition and, therefore, our central hypothesis are that the more straightforward to understand and implement an algorithm, the worse its performance will be. This approach is based

on the degree of complexity and size of the data. In the case of the Cifar-100 dataset, it is high due to the dataset's size and the configuration by channels.

Although this is our hypothesis, we are also aware that not all classification algorithms work in the same way for different types of problems, so this hypothesis and the results we achieved are not extensible to other classification problems. Although the logic to set the hypothesis could be extensible.

## 3.2 Pre-processing

### 3.2.1 Standardising

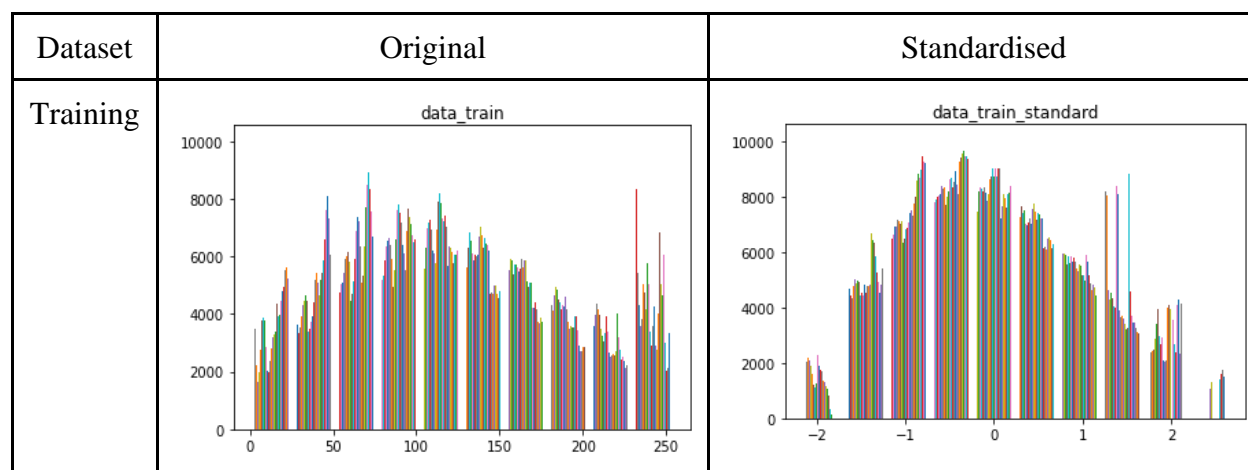
As one of the data scaling techniques, standardising involves scaling pixel values such that they are centered around the mean with a unit standard deviation. There are two levels of standardisation, either sample-wise where statistics are calculated and pixels are scaled at image level, or feature-wise where they are done across the entire dataset.

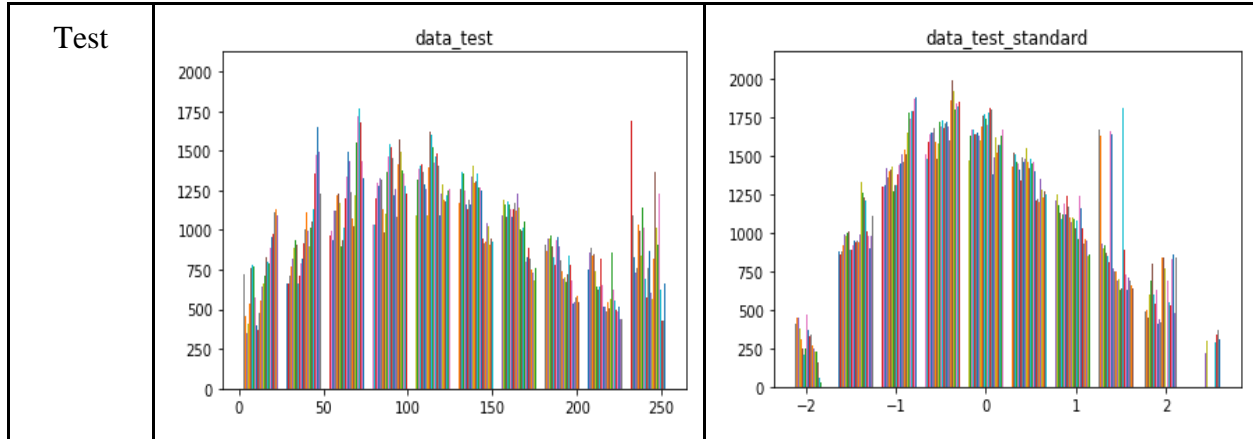
In our project, the latter method was used, where mean and standard deviation along the row of training data were first calculated, then the mean was subtracted from each pixel in both training and test datasets, followed by dividing them with the standard deviation:

$$x_{standardised} = \frac{x - \mu_{train}}{\sigma_{train}} \quad (1)$$

The test data was scaled using the same scaler in training data to avoid data leakage. Top left of Figure 3-1 shows an example of a standardized training image.

*Table 3-1 A comparison of distribution of original and standardised dataset. It is observed that the datas shifted around zero after standardising.*





### 3.2.2 Wavelet Transform

Image compression using discrete wavelet transform (DWT) was implemented in this project. The equation of DWT is given by <sup>7</sup> :

$$W_{\psi}(f)(j, k) = \langle f, \psi_{j,k} \rangle = \int_{-\infty}^{\infty} f(t) \bar{\psi}_{j,k}(t) dt \quad (2)$$

where  $\psi_{j,k}(t)$  is a discrete family of wavelets:

$$\psi_{j,k}(t) = \frac{1}{a^j} \psi\left(\frac{t - kb}{a^j}\right) \quad (3)$$

The computation details are not discussed here. As mentioned by<sup>7</sup>, the main idea of utilising wavelet transform is that efficient extraction of multi-scale structure by scaling and translating (respectively by parameter a and b) a given shape across a signal can be achieved, hence supporting an optimal tradeoff between time and frequency resolution.

Implementation in python: First, the Daubechies wavelet decomposition was applied to the first channel of an image. This would produce an array of coefficients with the length of the pixels in the channel. The coefficients were then sorted following the filtering process where those values above the chosen threshold were kept. Reconstruction of the channel using the new coefficient array was carried out and the process repeated for all channels of images in our dataset.



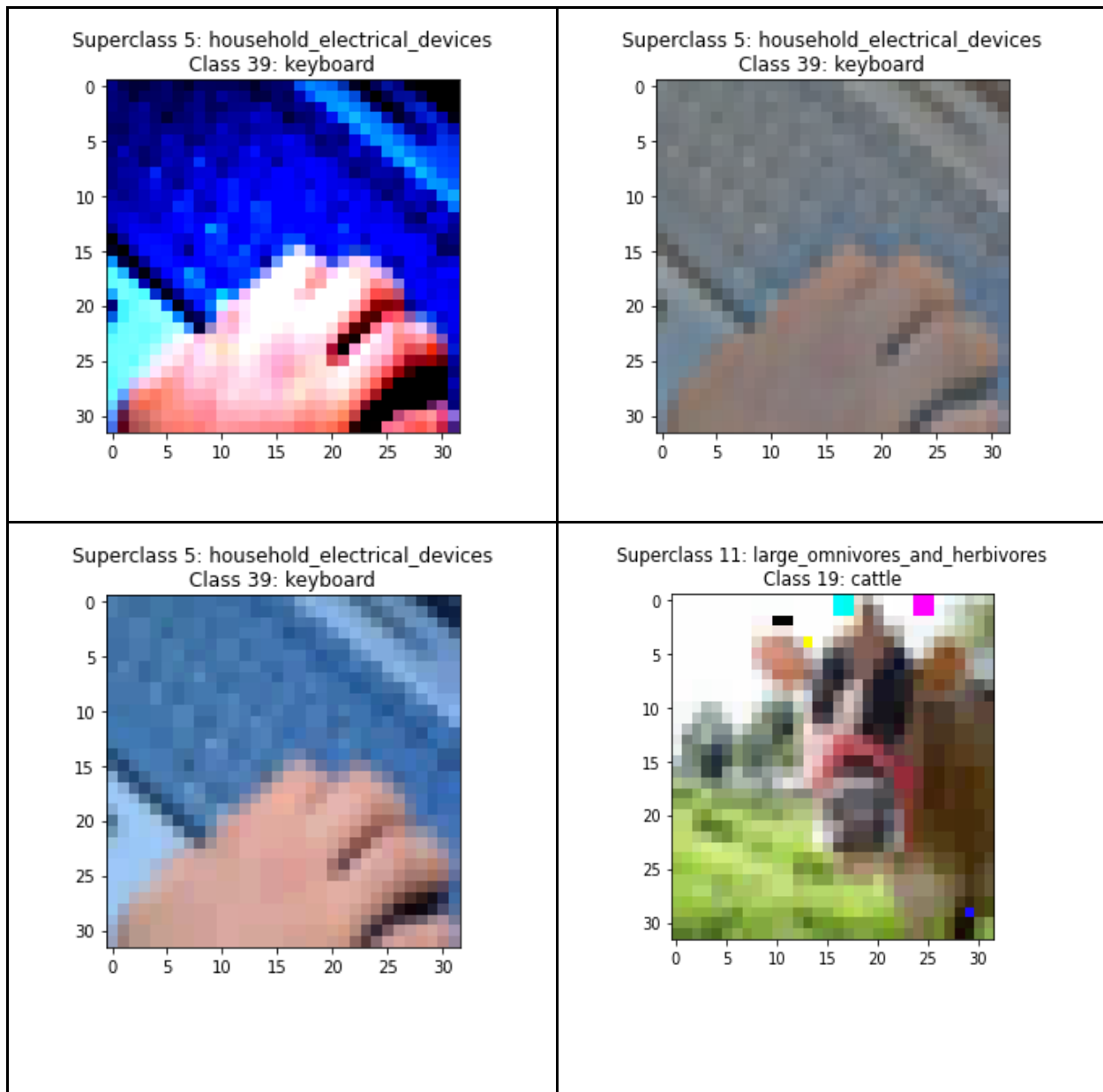


Figure 3-1 Images after different types of pre-processing techniques: (Top left) Standardising, (Top right) ZCA whitening, and (Bottom left and right) Wavelet Compression

### 3.2.3 Zero Component Analysis (ZCA) Whitening

Zero Component Analysis (ZCA) whitening transformation of the data was performed with the goal of decorrelating neighbouring pixels from each other. Following the rationale of <sup>8</sup> and <sup>9</sup>, we perform this particular whitening procedure because it is intended to help statistical learning algorithms pay attention to more interesting, higher-order interactions, rather than just observe correlations between neighbours.

This transformation involves multiplying each channel of the data matrix with a whitening matrix for that channel. We can produce the whitening matrix with the following transformations of our data.

$$W = \sqrt{n-1}(XX^T)^{-\frac{1}{2}} \quad (4)$$

where  $XX^T$  is the covariance matrix of the data.

### 3.3 Classifiers

In the following section, we will describe each one of the classifiers implemented. You will find a brief explanation and our own approach to implement them. Some references are added to include further evidence or research.

#### 3.3.1 Gaussian Naive Bayes

Naive Bayes (NB) is one of the simple probabilistic classifiers that holds the assumption of conditionally independence between attributes. It is called “naive” as the assumption is made to avoid complexity in computation (and hence save costs), but we know that it is not true.

Gaussian Naive Bayes (GNB) is no other but one of the NB classifiers that is applied to a dataset with real-valued features, which is also our case as the features are numerical pixel values. The equation is given by <sup>10</sup>:

$$P(x|C_k) = \prod_{j=1}^d \frac{1}{\sqrt{2\pi\sigma_{j,k}^2}} \exp\left(-\frac{1}{2\sigma_{j,k}^2}(x_j - \mu_{j,k})^2\right) \quad (5)$$

where  $x$  is the test data given that  $x_j|C_k$  follows the distribution of  $\mathcal{N}(\mu_{j,k}, \sigma_{j,k})$ , with mean  $\mu_{j,k}$  and standard deviation  $\sigma_{j,k}$  of the distribution from the training data.

#### How does it work?

The mechanism is fairly straightforward and easy to implement:

For each category

1. train the model by computing the mean and standard deviation of training data of the same category and
2. fit the model by computing the likelihood probability of test data,

The prediction of the test label is made by choosing the class which gives the highest value of likelihood probability.

#### Advantages and disadvantages

NB classifiers are well-known for their low computation cost, in fact their complexity is just  $O(CD)$  for  $C$  classes and  $D$  features. Due to their simplicity, they are “relatively immune to overfitting”, as discussed in <sup>10</sup>. The main disadvantage of NB is that it is a linear classifier, hence it works better when the decision boundary is linear, which is not true in image data.

### 3.3.2 Random Forest

This classifier is built upon extending the application of the traditional Decision Tree classifier. Put in simple terms, this means that the algorithm works by building a set (to be exact, an ensemble) of decision trees and running a random search through them. According to different sources, it is a simple classifier and most of the time, produces good results.

#### How does it work?

The following is a brief explanation of the algorithm.

1. Define the hyperparameters. We need to define, at least, the following hyperparameters to have a good result from the classification task:
  - a. Number of estimators: Number of trees in the forest
  - b. Maximum Depth: The maximum depth for each tree.
2. Training. This process will occur by splitting the training data in two parts, so as to fine tune the hyperparameters of the model. In our case, we are using 10-folds cross validation through a random grid to get the results from the classifier. In the particular case of the random forest, the definition of the grid is as follows:

```
# 1. Definition of the set of possible hyperparameters of the model
```

```
n_estimators = [10, 50, 100, 200, 400, 600]
max_features = ['auto', 'sqrt', 'log2']
max_depth = [10, 20, 30, 40, 50, 100, 200, None]
min_samples_leaf = [1, 2, 5, 10]
```

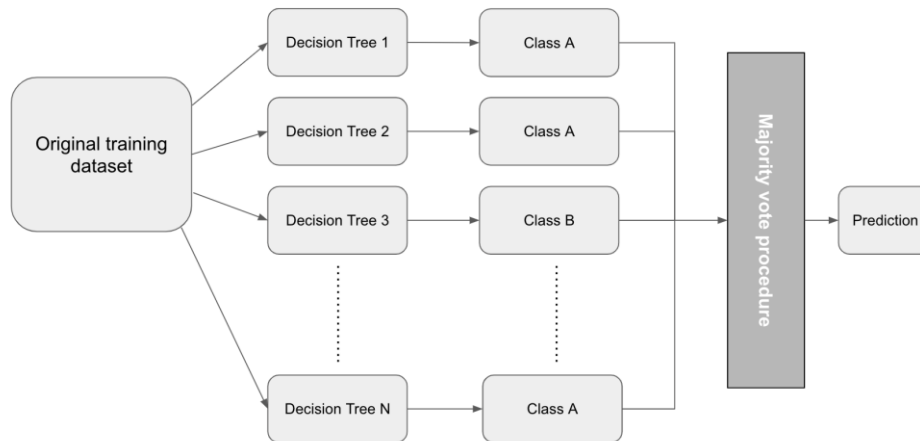
```
# 2. Defining the random grid
```

```
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_leaf': min_samples_leaf}
```

3. After defining the grid, it is necessary to run the fitting process, which is the function fit in our classifiers' classes. Basically, in this step we are running the fine-tuning process to get the best parameters to use to feed the function to predict the true labels using the test set provided. It is important to mention that the decision criteria underlying this classifier is

the majority vote procedure, which is close to the one generated by the KNN classifier. This procedure consists of ranking the number of classes that are most repeated after each tree classification and decides according to the top one.

The architecture of the classifier can be seen in the Figure 3-2.



*Figure 3-2 Random Forest architecture*

Source: Own elaboration based on different diagrams found after research

### Advantages and disadvantages

Commonly, this classifier prevents overfitting, because of the effect provided by the large number of trees that the algorithm creates to classify. In addition to that, it can be used for both classification and regression tasks and its parameters are simple to understand. The disadvantage is mainly its speed. For real time applications it is hard to use, especially after training.

### 3.3.3 Convolutional Neural Network (CNN)

The CNN was chosen as our third and most complex classifier based on our choice of dataset, Cifar-100. CNNs have been shown to perform extremely well for image classification, achieving state of the art results.

We used the Keras Sequential class in order to build our CNN architecture, with 4 convolutional layers, and 2 dense layers. We chose categorical cross entropy as our loss function. Our model ended up with 208,860 trainable parameters. The specific architecture is described in Figure 3-3.

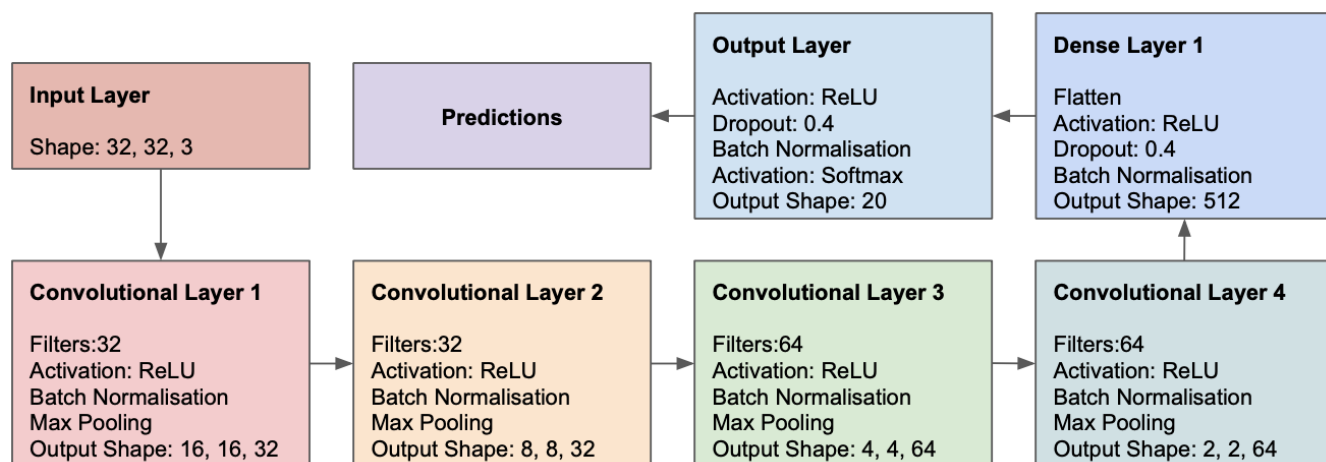


Figure 3-3 CNN architecture

Optimisation of this architecture and the hyper-parameters of the model was performed both by reading academic literature and and class material on the topic, as well as by performing systematic experiments ourselves in order to tune certain hyper-parameters such as the *learning rate* and the choice of *optimizer* for the gradient descent.

We chose the following hyper-parameter settings and architecture decisions for our model based on research, with justification and references given:

Batch size = 32	
Kernel size = (3,3)	for all filters
Strides = (1,1)	for all filters
Activation function = ReLU	for all layers
Pool size = (2,2)	for all layers
Dropout = 0.4	for dense layers

Batch size, Kernel size, Strides, and Pool size were chosen as good ‘heuristic’ choices given an absence of ability to finely tune all these parameters on a large-scale model such as ours. These decisions were largely informed by our lecture material and Henry Wald’s Week 11 tutorial.

ReLU (rectified linear unit) was used as an activation function as it allows us to avoid some of the pitfalls of other non-linear activation functions like sigmoid or tanh such as saturation of values near the extremities of the functions, as well as faster computation time.

Dropout was used to prevent overfitting as suggested by<sup>11</sup>, and the value of 0.4 was chosen as we observed reducing accuracy for different parameter settings.

*Learning rate* and *optimizer* were tuned with a brief experimental process. The results are reported here below in Figure 3-4.

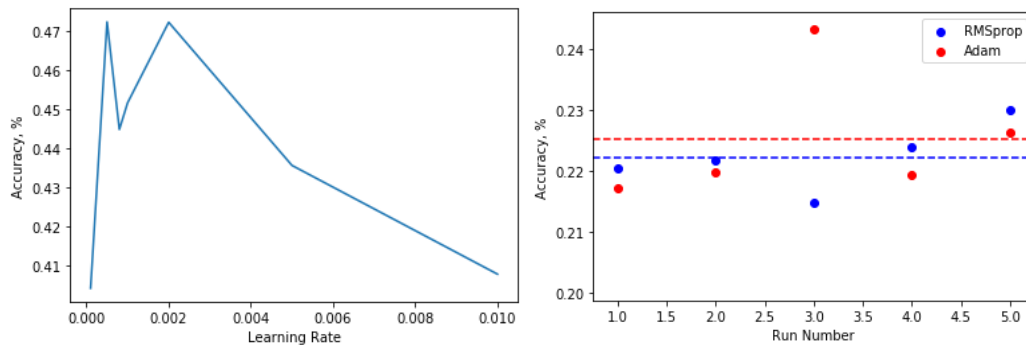


Figure 3-4 Learning rate(left) and optimizer choice(right)

Following from this process, we could see that a small *learning rate* between 0.002 and 0.0005 was optimal and so our experiments continued with the choice of 0.002. Comparing the RMSprop and Adam optimizers we see that they perform similarly. Due to an outlier, the Adam optimiser has a slightly higher average accuracy, but we believe that this is due to small sample size. If we remove this outlier, RMSprop performs better and so our choice of *optimizer* was RMSprop.

### Advantages and disadvantages

The largest disadvantage of a CNN architecture is that the combinatorial space of possible hyper-parameters is very large and as a result, it is a difficult search problem to find the optimal hyper-parameter settings for a given problem. On top of this, training of the model can be computationally costly, and both time taken to run the algorithm and accuracy can be largely dependent on the computational costs that can be devoted.

Importantly, the CNN classifier is able to recognise multi-scale features of data, as is demonstrated by its state of the art results on image and sound data. This is the primary motivation and advantage of using this model.

### Reflection on CNN architecture

Reflecting on our CNN architecture, there are some important decisions that I would highlight. Max pooling, while useful, was perhaps used without careful consideration in our particular case. Four uses of max pooling means that our features become very small, I speculate that we should have used only two or three instances of max pooling and not used any in the first layer, so as to preserve important information for longer in the process that max pooling may have been removing too early. A larger grid search of the space of hyper-parameters is necessary for optimising the architecture.

## 4 Results

This section will be presented using the following order:

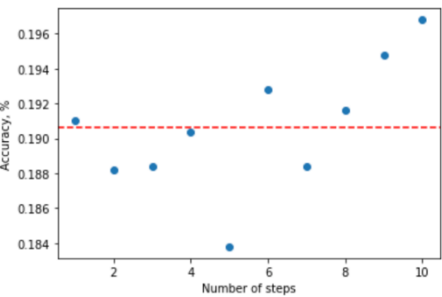
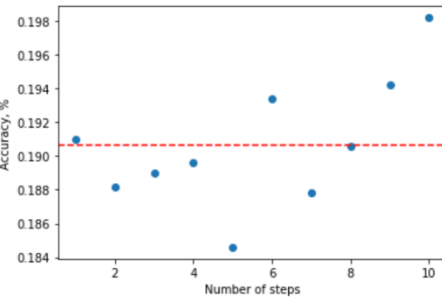
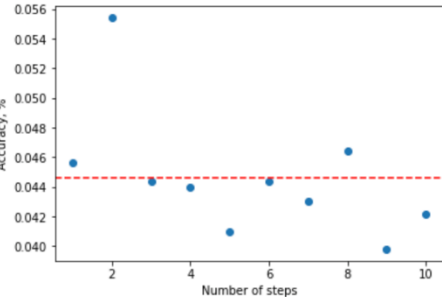
1. In Section 4.1, 10-fold cross-validation analysis: We include a table for each classifier, and in each one of them, we have its performance in each of the pre-processed datasets. These correspond to the columns of each table. In the first row, the graphs show the different results of each fold's accuracy; in the second, we have the value of the average accuracy in the complete process. The third row shows the total execution time of the cross-validation process. We wrapped up by offering a summary comparison of the information in two formats, one plot and one table.
2. In Section 4.2, we discuss our efforts to improve the execution time and optimize our classifiers.
3. Section 4.3 presents the classification results on the original test set for the three classifiers; the training set corresponds to the one where we got the best performance in the 10-fold cross-validation. We use the hyperparameters resulting from the fine-tuning process for Random Forest and CNN. There was no fine-tuning for the GNB classifier.

### 4.1 10-fold cross-validation analysis

The next set of tables present the main results of our 10-fold validation processes executed for each classifier in the different three pre-processed datasets.

#### 4.1.1 GNB

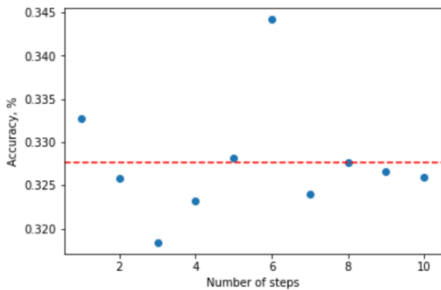
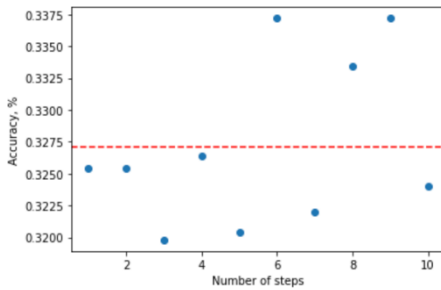
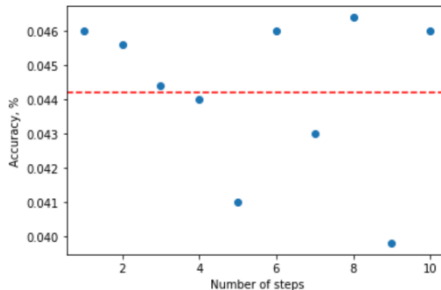
*Table 4-1 GNB 10-fold cross-validation result for three differently pre-processed dataset*

Standardised dataset	Wavelet transformed dataset	ZCA transformed dataset
		
19.06 % average accuracy	19.06% average accuracy	4.462% average accuracy
36 seconds	51 seconds	53 seconds

**Remark:** Accuracy for the standardised and wavelet datasets is very similar. In terms of execution time, the former is better, with 17.25 minutes versus 17.40 minutes of the latter. Regarding the classification using the ZCA transformed dataset, it is worse than the previous two.

## 4.1.2 Random Forest

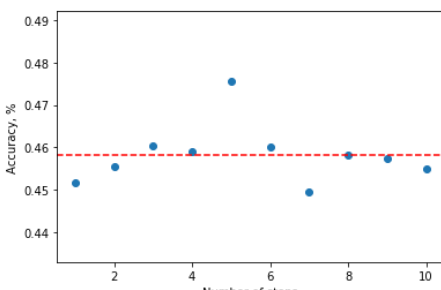
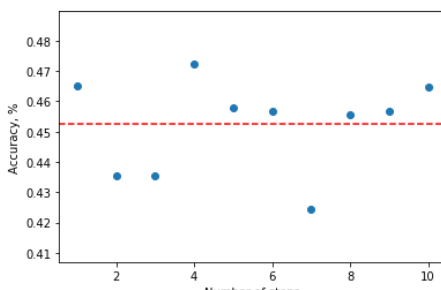
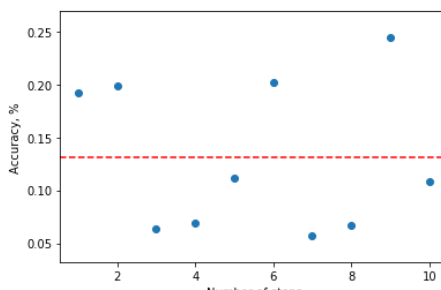
Table 4-2 RF 10-fold cross-validation result for three differently pre-processed dataset

Standardised dataset	Wavelet transformed dataset	ZCA transformed dataset
		
32.77% average accuracy	32.71% average accuracy	4.42% average accuracy
17,4 minutos	17.25 minutes	11.16 minutes

**Remark:** Accuracy for the standardised and wavelet datasets is very similar. In terms of execution time, the former is better, with 17.25 minutes versus 17.40 minutes of the latter. Regarding the classification using the ZCA transformed dataset, it is worse than the previous two.

## 4.1.3 CNN

Table 4-3 CNN10-fold cross-validation result for three differently pre-processed dataset

Standardised dataset	Wavelet transformed dataset	ZCA transformed dataset
		
45.82% average accuracy	45.25 average accuracy	0.1315
77 minutes	75 minutes	83 minutes

**Remark:** Accuracy for the standardised and wavelet datasets is almost the same. In terms of execution time, the latter is better, with 75 minutes versus 77 minutes of the former. Regarding the classification using the ZCA transformed dataset, it is worse than the previous two.



#### 4.1.4 Summary

A histogram was plotted based on the result of 10-fold cross validation for different experiments (see Figure 4-1). “GNB” denotes the experiment using Gaussian Naïve Bayes classifier, “RF” for Random Forest and “CNN” for the one using Convolution Neural Network. The abbreviation behind the classifiers indicates the pre-processing methods: “STD” is the experiment using standardised data, “WV” wavelet-transformed data, and “ZCA” the data transformed using zero-component analysis whitening.

From the plot, we can visualise the overall performance of each classifier on three differently pre-processed data. It is noted that the results we observe in 10-fold cross validation are congruent with the results we have on actual test data which will be presented in the Section 4.3, where 1) CNN worked the best in terms of accuracy on all types of data, RF the second, and GNB the worst, 2) the running time of GNB is the shortest, followed by RF, and CNN has the longest execution time (in fact from Figure X the running time of CNN is around 125 times longer than that of GNB), and 3) performance of each type of pre-processed data is consistent from classifier to classifier, where standardised data stands out among all.

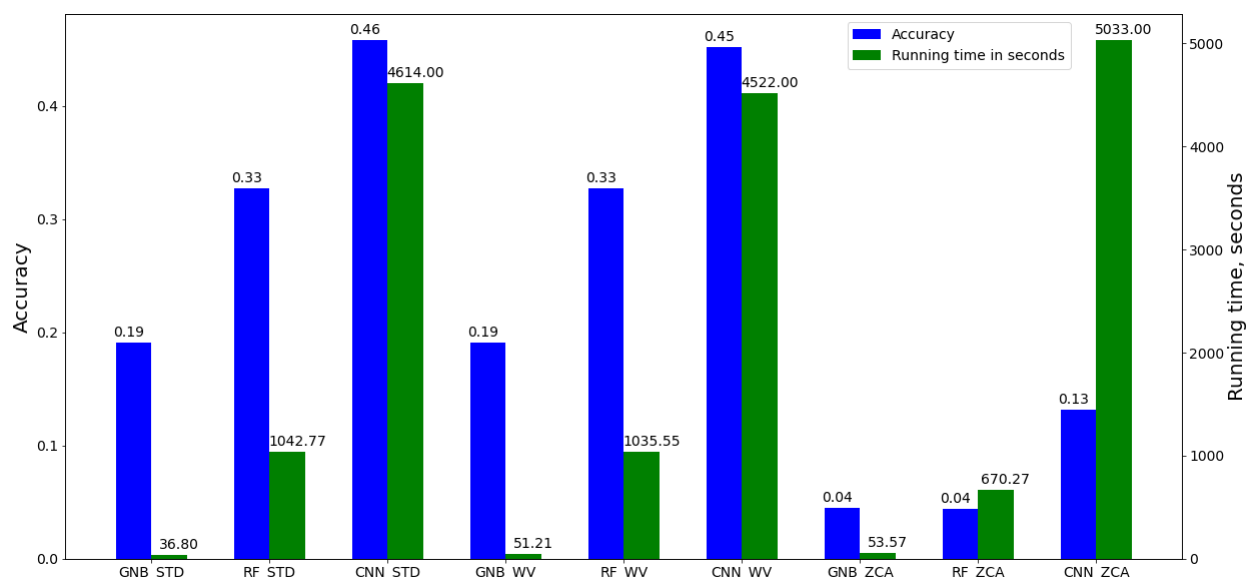


Figure 4-1 Overview of 10-fold cross validation results of different experiment using three classifiers

Table 4-4 summarizes the information shown in the previous plot in a table form. It is important to notice that in Random Forest and CNN the classification task using the wavelet transformed dataset performs better than the one using the standardised dataset. This is not true for GNB.

Table 4-4 Summary for K-fold cross validation in table.

		Classifiers		
Processing		GNB	Random Forest	CNN
Standardised data	Accuracy	19%	33%	46%
	Execution time	36 seconds	17 minutes	76 minutes
Wavelet transformation	Accuracy	19%	33%	45%
	Execution time	51 seconds	17 minutes	75 minutes
ZCA transformation	Accuracy	4%	4%	13%
	Execution time	53 seconds	11 minutes	83 minutes

## 4.2 Execution time improvements

### 4.2.1 GNB

Due to the nature of this classifier, it was not necessary to apply any improvement in execution time.

### 4.2.2 Random Forest

We applied parallelization. We define the parameter `n_jobs = -1`, which implies that the algorithm opens parallel threads to improve execution time. It was vital for the fine-tuning process because as we executed it using a random grid, it would take a lot of time and computing resources. We went from more than 1.5 hours to something like 40 minutes of execution.

### 4.2.3 CNN

For this classifier we found time improvements with no loss in accuracy by reducing the number of features that each convolutional layer was creating. For example, reducing feature numbers in our first convolutional layer from 96 to 64 to 32 did not reduce the accuracy performance of our classifier. Similarly, but with some cost in accuracy, we reduced the number of epochs to 5 for most of our hyper-parameter tuning process in order to improve the time taken to achieve results.

### 4.3 Results of the classification task on the actual test set

Table 4-5 shows summary of classification metrics using the given test set and Table 4-6 shows the confusion matrices of three classifiers. For our final accuracy, recall, and precision reporting of CNN we ran our classifier for 10 epochs and achieved a small improvement in our result. It was not possible to do this during the 10-fold cross validation process due to the computational and time cost of doing so, but it seemed prudent to include this promising result that more viewings of the dataset improved our performance on the test set.

Table 4-5 Performance of classifiers in predicting test data coarse labels using standardised data.

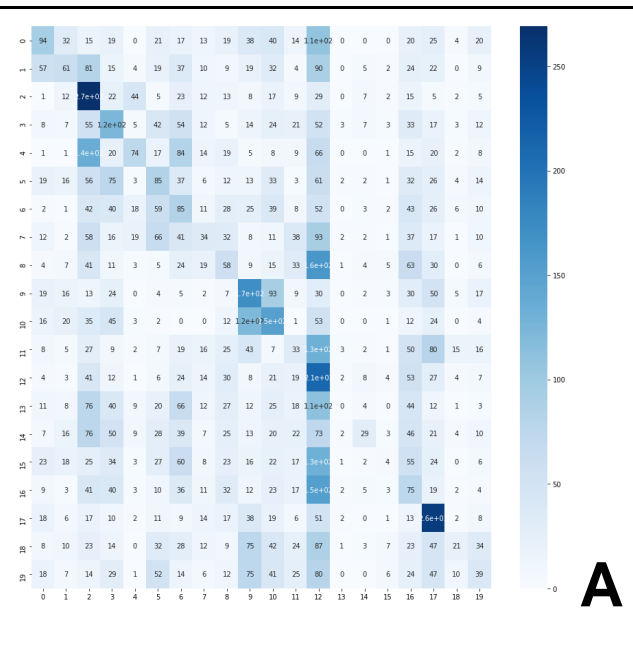
		Classifiers		
Processing		GNB	Random Forest	CNN
Running over the standardised dataset	Accuracy	18.74	34.5	48.43
	Recall	18.74	34.5	48.43
	Precision	19.05	34.41	48.32
	Execution time	57 seconds	7 minutes	14 minutes

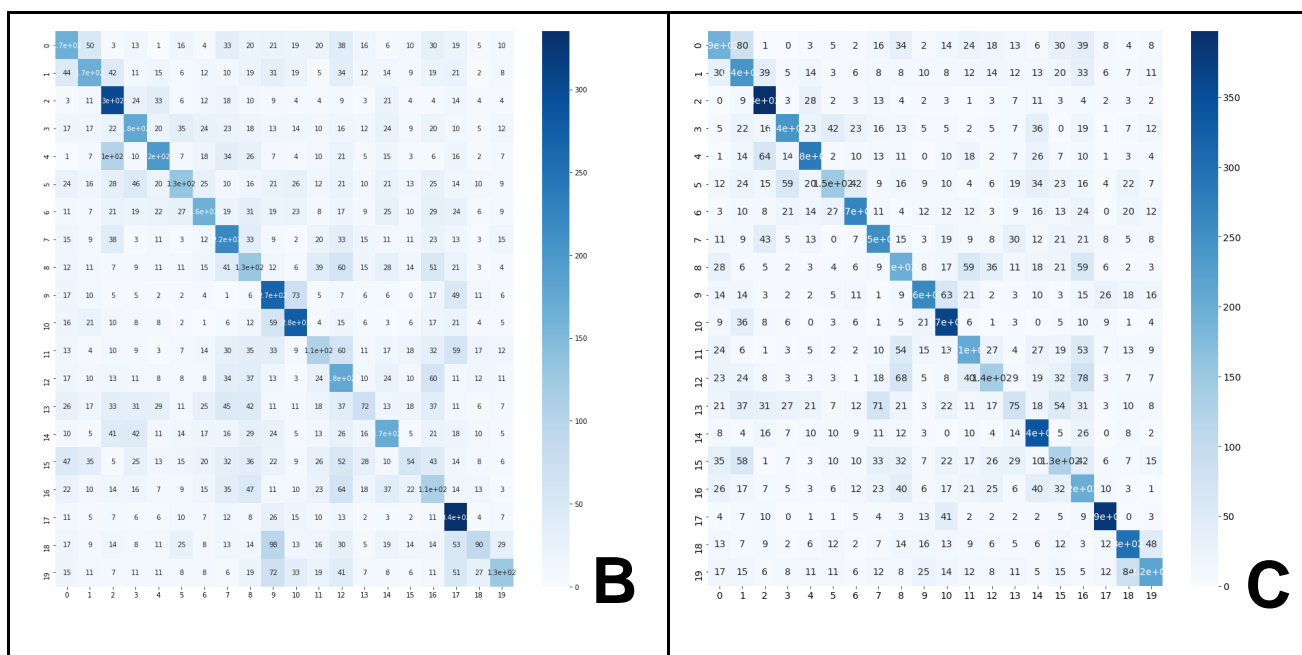
Table 4-6 Confusion matrices of for three classifiers.

A: Confusion matrix for GNB classifier  
B: Confusion matrix for Random Forest classifier  
C: Confusion matrix for CNN

**Remark:** Notice that the effect in the diagonal line becomes stronger from A to C. This means that the accuracy of the classification task of A is the lowest and in C is the largest one. B is conveniently in the middle.

In appendix ii we added these plots in larger size.





## 5 Discussion

The very first issue to highlight here is that all the three classifiers work better on the standardised dataset. To be honest, we actually do not know why this happens. Specially, because we expected that the classification using the standardised dataset performs worse than Wavelet and ZCA.

Also, the classification using the wavelet transformed dataset is very similar in the three cases, in terms of accuracy. In terms of execution time, for GNB it was three times greater than the one using the standardised dataset. In it comes to compare the execution time for Random Forest and CNN classifiers, the execution time using the Wavelet transformation is slightly better than the one obtained for GNB.

About the Wavelet transformation applied, we believe that as long as a uniform statistical threshold was applied for all images to keep 50% of the largest wavelet coefficient, the results of the classification tasks were not remarkably better than the ones from using just the standardised dataset. Bottom row of Figure 3-1 shows two examples of wavelet-compressed images. Observing by eyeball, we could see that when a threshold of 50% was applied, the features of the figure on the left were basically preserved well, but not in the case of the one on the right, where some information is lost. This observation suggests that the threshold should be varied across the images. This is one of the issues that we should explore if any further research is applied.

On the other hand, it is expected that ZCA works well at least for CNN, but it does not seem true in our study. We agreed that the main reason would be an incorrect implementation, because in the literature, there are results showing that ZCA might lead to better results<sup>9</sup>. Given our results, there is evidence that our implementation should be revised in future work.

The no free lunch theorem (NFLT)<sup>12</sup> states that no classifier is optimised for all problems. Research in the area of image classification has determined however that CNN architectures perform much better than many other machine learning algorithms at this task. NFLT suggests that there will be inherent differences in the performance of different classifiers on the same task, even though they may perform the same when averaged across all tasks.

## 6 Conclusion

In this project, three classifiers with different levels of complexity were implemented, starting with the simplest GNB, followed by Random Forest and finally CNN, the most complex algorithm. We experimented these classifiers with three types of pre-processed data: the standardised data, wavelet-transformed data and ZCA-transformed data. Based on the results, it is concluded that the performance of the classifiers increases with the level of complexity of algorithms as well as their computation cost. In fact, this is in accord with our predictions, where from the web page 'Papers with code' mentioned in Section 2 had depicted that deep learning methods are the best classifiers for Cifar-100 dataset.

Comparing the performance of pre-processing methods, it is interesting to see how standardised data turned out to be the one achieving the highest accuracy for all experiments, despite the expectation of high performance in ZCA-transformed - CNN combination.

Regarding the future work, there are a few suggestions:

1. Pre-processing methods
  - Revisit wavelet transform, where threshold to retain number of features should be varied by each image instead of a uniform value
2. Classifiers
  - Revisit Random Forest, using MCMC approach
  - Adopt other deep learning algorithm such as ResNet
  - Experiment the combination of Random Forest and CNN

## References

1. Amancio D, Comin C, Casanova D, Travieso G, Bruno O, Rodrigues F, et al. A systematic comparison of supervised classifiers. *PloS one*. 2014;9:e94137.
2. Wilson J, Denny MJ, Bhamidi S, Cranmer S, Desmarais B. Stochastic weighted graphs: Flexible model specification and simulation. *Soc Networks*. 2017;49:37-47.
3. Krizhevsky A, Sutskever I, Hinton G. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*. 2012;25.
4. Park D-C. Image classification using naïve bayes classifier. *International Journal of Computer Science and Electronics Engineering (IJCSEE)*. 2016;4(3).
5. Xu B, Ye Y, Nie L, editors. An improved random forest classifier for image classification. 2012 IEEE International Conference on Information and Automation; 2012 6-8 June 2012.
6. Hu L-Y, Huang M-W, Ke S-W, Tsai C-F. The distance function effect on k-nearest neighbor classification for medical datasets. *SpringerPlus*. 2016;5.
7. Brunton SL, Kutz JN. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge: Cambridge University Press; 2019.
8. Krizhevsky A. *Learning multiple layers of features from tiny images*. University of Toronto. 2012.
9. Pal KK, Sudeep K. Preprocessing for image classification by convolutional neural networks. 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT). 2016:1778-81.
10. Murphy KP. *Machine learning: A probabilistic perspective*: The MIT Press; 2012.
11. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*. 2014;15:1929-58.
12. Wolpert DH, Macready WG. No free lunch theorems for optimization. *Trans Evol Comp*. 1997;1(1):67–82.

## Appendix

### Appendix i. Instructions to run the other algorithms code

Download the Cifar-100 dataset from the link provided in the Assignment 2 specification document. The data can be loaded easily if the notebook files exist in the same folder as the folder that houses the data.

Basically, run all the cells from the first to the last one in order.

Please, notice that the cells of the fine-tuning processes are commented, because that part of the code takes a bit longer in getting to the results. So, it will not be possible to run them and we assume that the value of our hyperparameters for Random Forest are the following:

```
{ 'max_depth': 50,  
  'max_features': 'sqrt',  
  'min_samples_leaf': 2,  
  'n_estimators': 600 }
```

It is signaled where to stop. It will be signaled as follows.

### Experimental Setting

**STOP HERE!!!!!!!**

### Experiments for Gaussian Naive Bayes

```
"""start_time_GNB_std = time.time()  
kf = KFold(n_splits=10)  
  
cv_accuracy_list_std = []  
cv_recall_list_std = []  
cv_precision_list_std = []
```

Also, it is marked where to continue running the code. It is as follows.



```
[ ] """print("The accuracy is {:.08f}, recall is {:.08f}, and precision is {:.08f").\n      format(accuracy_std_rf, recall_std_rf, precision_std_rf))"""
```

The accuracy is 0.33963636, recall is 0.33963636, and precision is 0.33963636

## ▸ Fitting to the train set

***START HERE AGAIN!!!!***

## ▸ GNB

```
[75] ## Gaussian Naive Bayes\n\nstart_gnb = time.time()\n# 1. Create a Gaussian Naive Bayes classifier\ngnb_test = GNB()\n\n# 2. Training the model using the splits from the Kfold step\ngnb_test.fit(X_train_standard, Y_train_standard)
```

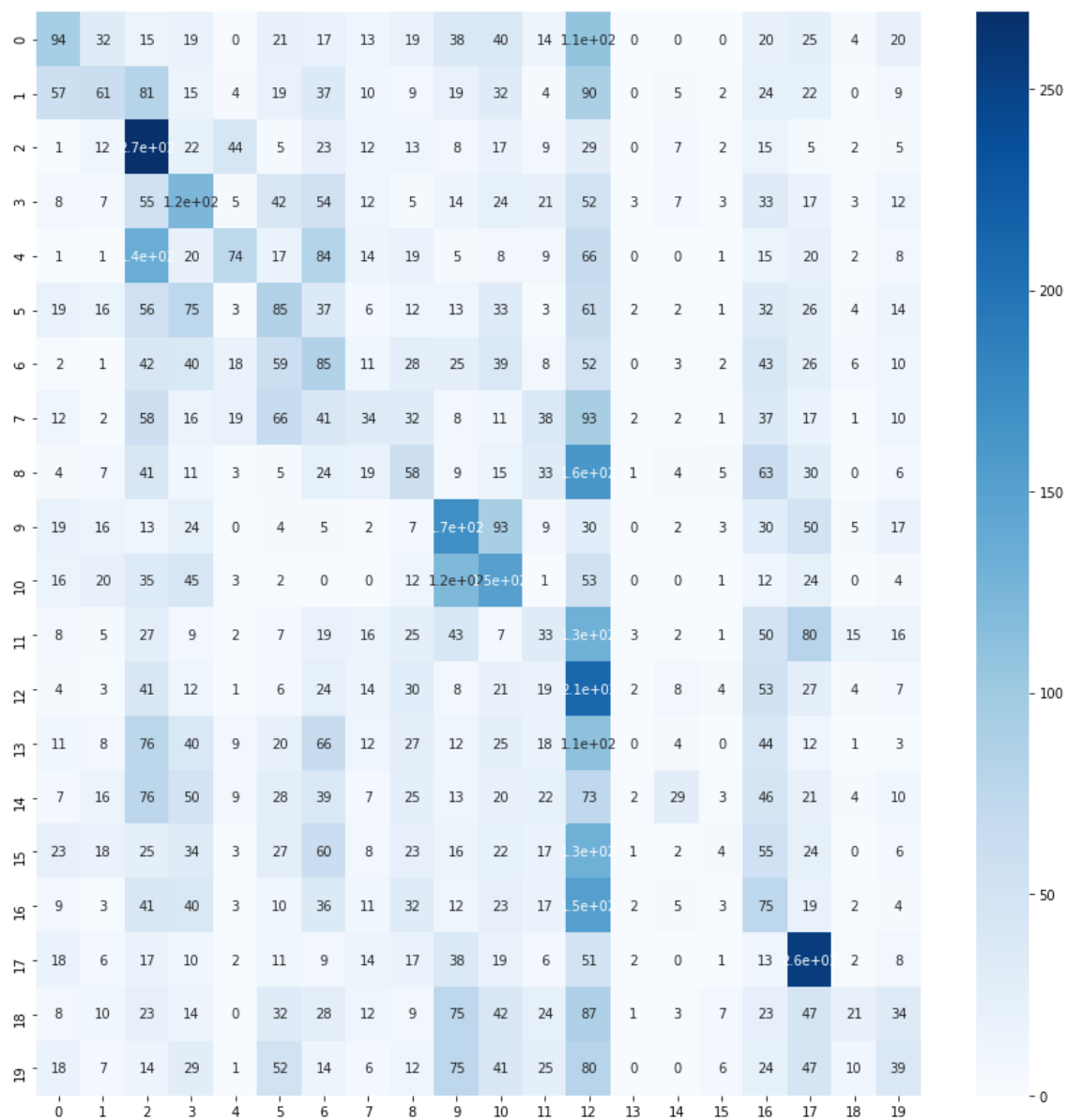
## Appendix ii. Instructions to run the best algorithm code

Download the Cifar-100 dataset from the link provided in the Assignment 2 specification document. The data can be loaded easily if the notebook files exist in the same folder as the folder that houses the data.

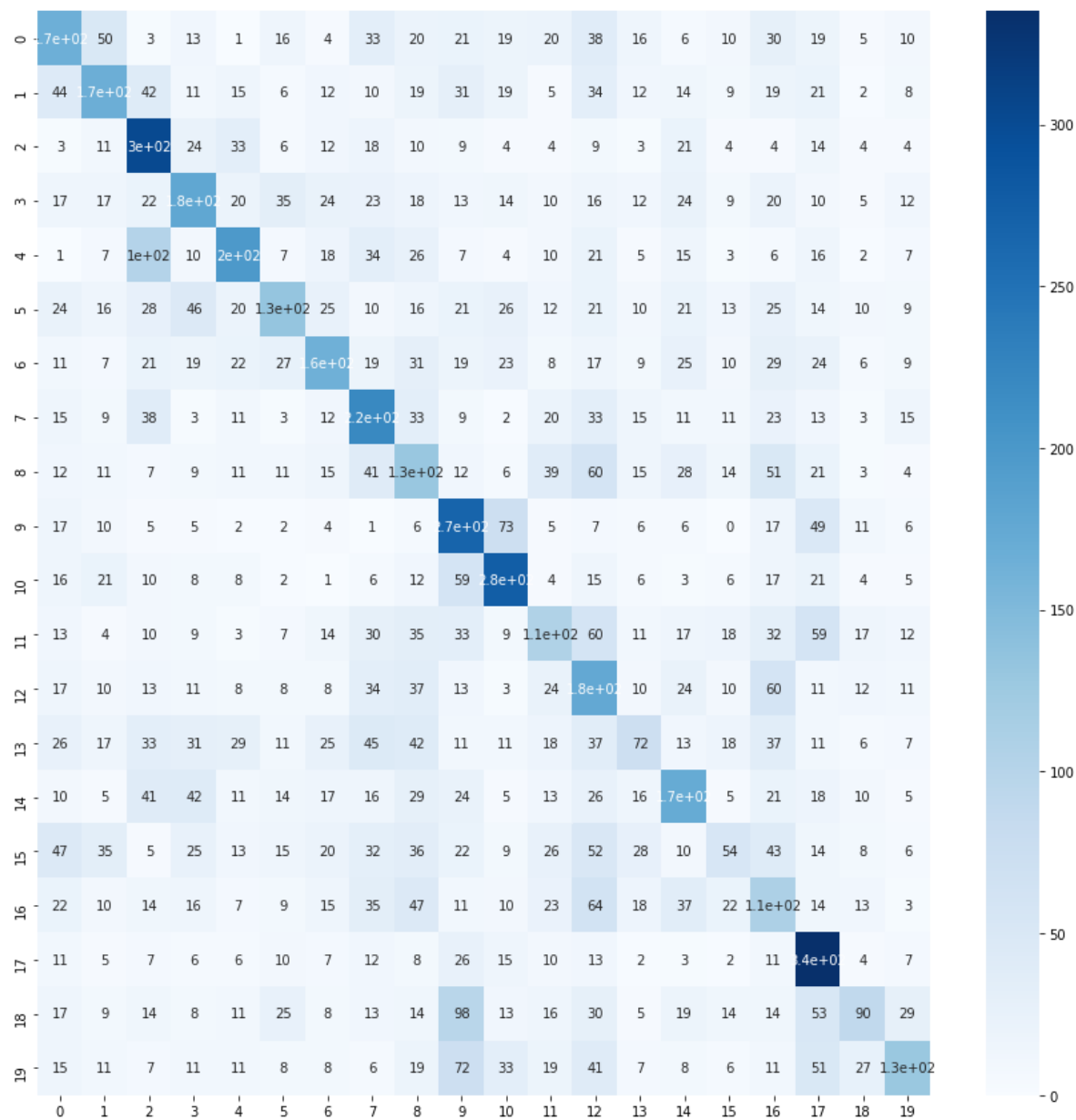
The notebook can be run from top to bottom as all lengthy calculations and plots that rely on that data are commented out. You can expect longer computation time during the processing of the data sets (3 minutes) and during the running of the final classifier (14 minutes).

## Appendix iii. Confusion matrices in large scale

## Gaussian Naive Bayes



## Random Forest Confusion matrix



## CNN Confusion matrix

