ANNEXURE B – Declaration of originality

DECLARATION OF ORIGINALITY

UNIVERSITY OF PRETORIA

The Department of INFORMATION SCIENCE places great emphasis upon integrity and ethical conduct in the preparation of all written work submitted for academic evaluation.

Academics teach you about referencing techniques and how to avoid plagiarism; it is your responsibility to act on this knowledge. If you are at any stage uncertain as to what is required, you should speak to your lecturer before any written work is submitted.

You are guilty of plagiarism if you copy something from another author's work (e.g. a book, an article or a website) without acknowledging the source and pass it off as your own. In effect you are stealing something that belongs to someone else. This is not only the case when you copy work word-for-word (verbatim) but also when you submit someone else's work in a slightly altered form (paraphrase) or use a line of argument without acknowledging it.

Students who commit plagiarism will not be given any credit for plagiarised work. The matter may also be referred to the Disciplinary Committee (Students) for a ruling. Plagiarism is regarded as a serious contravention of the University's rules and can lead to expulsion from the University.

The declaration which follows must accompany all written work submitted while you are a student of the Department of INFORMATION SCIENCE. No written work will be accepted unless the declaration has been completed and submitted.

Full names and surname of student: BASETSANA SEKHOTO

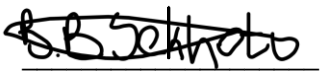
Student number: U21608611

Topic of work: Assignment 2 Report

Declaration

1. I understand what plagiarism is and am aware of the University's policy in this regard.

2. I declare that this ASSIGNMENT (e.g. essay, report, project, assignment, dissertation, thesis, etc) is my own original work. Where other people's work has been used (either from a printed source, Internet or any other source), this has been properly acknowledged and referenced in accordance with departmental requirements.



SIGNATURE

11/10/2024

DATE

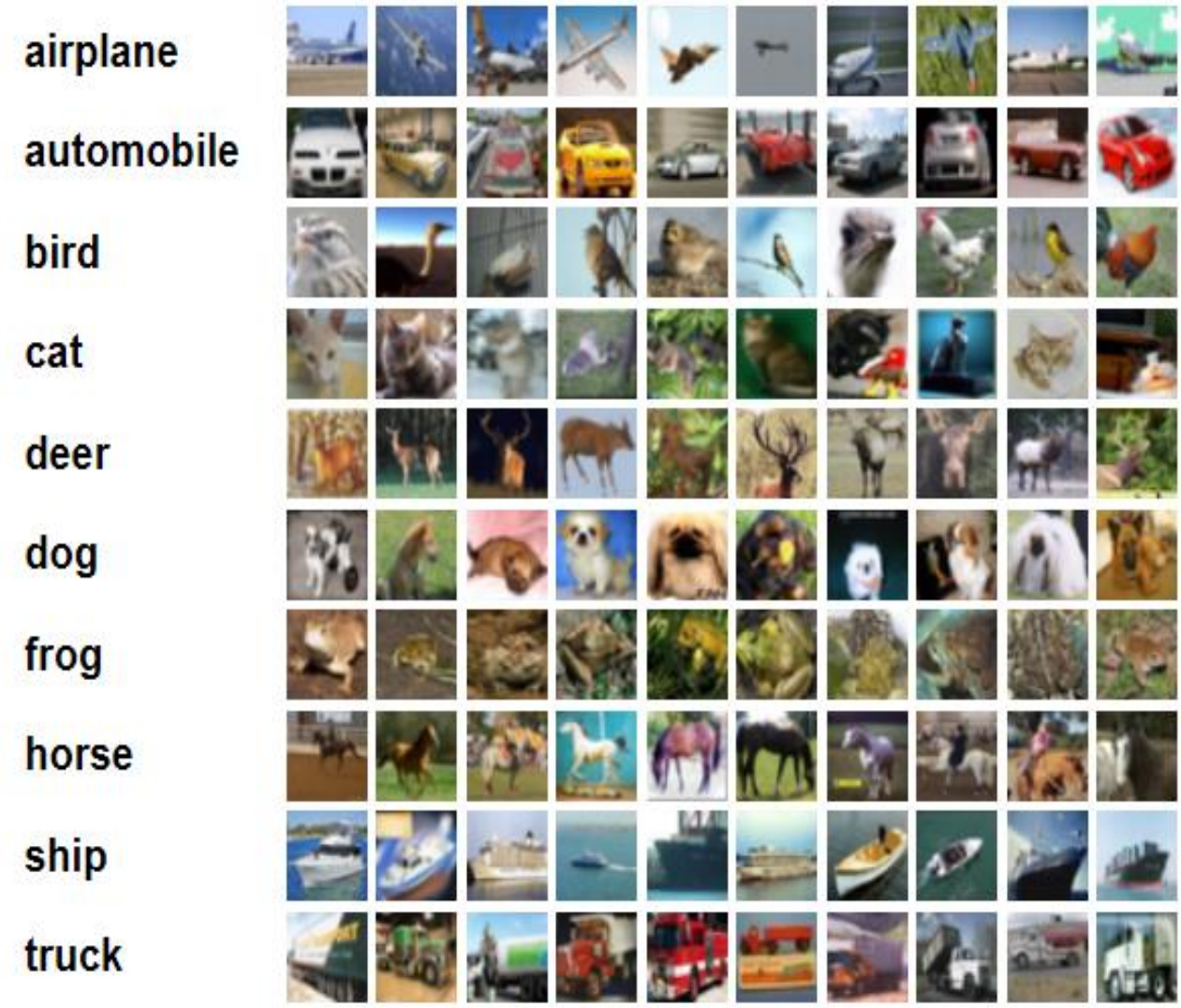
INF791 ASSIGNMENT 2

Introduction: Unsupervised Learning

Current NLP research in unsupervised learning methods is increasingly successful (Berg-Kirkpatrick et al., 2010). Unsupervised learning is a method that reflects the statistical structure of the overall input pattern collection through enabling systems to represent specific input patterns (Dayan, 1999). Instead of testing data structures through the use of statistical methods, constructing prediction or classification models based on specific conditions and responses, unsupervised learning explores data structures and then generates hypotheses (Valkenborg et al., 2023). Unsupervised learning is crucial due to its likeliness of being more prevalent in the brain than supervised learning (Dayan, 1999). In comparison to supervised learning, unsupervised learning is a subjective, non-predictive method typically used in exploratory data analysis, with no simple goal for prediction (James et al., 2023). This report is about unsupervised learning that is done on the Cifar-10 dataset. It consists of the introduction, methodology, code explanations, results and conclusions.

Data Collection: The CIFAR-10 Dataset

The data is collected from the CIFAR-10 dataset that is open access on Kaggle. CIFAR-10 is a computer-vision dataset used for object recognition, consisting of 60,000 32x32 colour images from the 80 million tiny images dataset and Kaggle is launching a CIFAR-10 leaderboard for machine learning enthusiasts, allowing them to compare their approaches to the latest research methods on Rodrigo Benenson's classification results page. The dataset was downloaded from the Kaggle website along with its own train data and test data and the data is loaded on the code (Soriano, 2021).



The image below shows all superclasses found in the Cifar-10 dataset and the classes. Moreover, it shows the names of the images found within the classes of the Cifar-10 dataset.

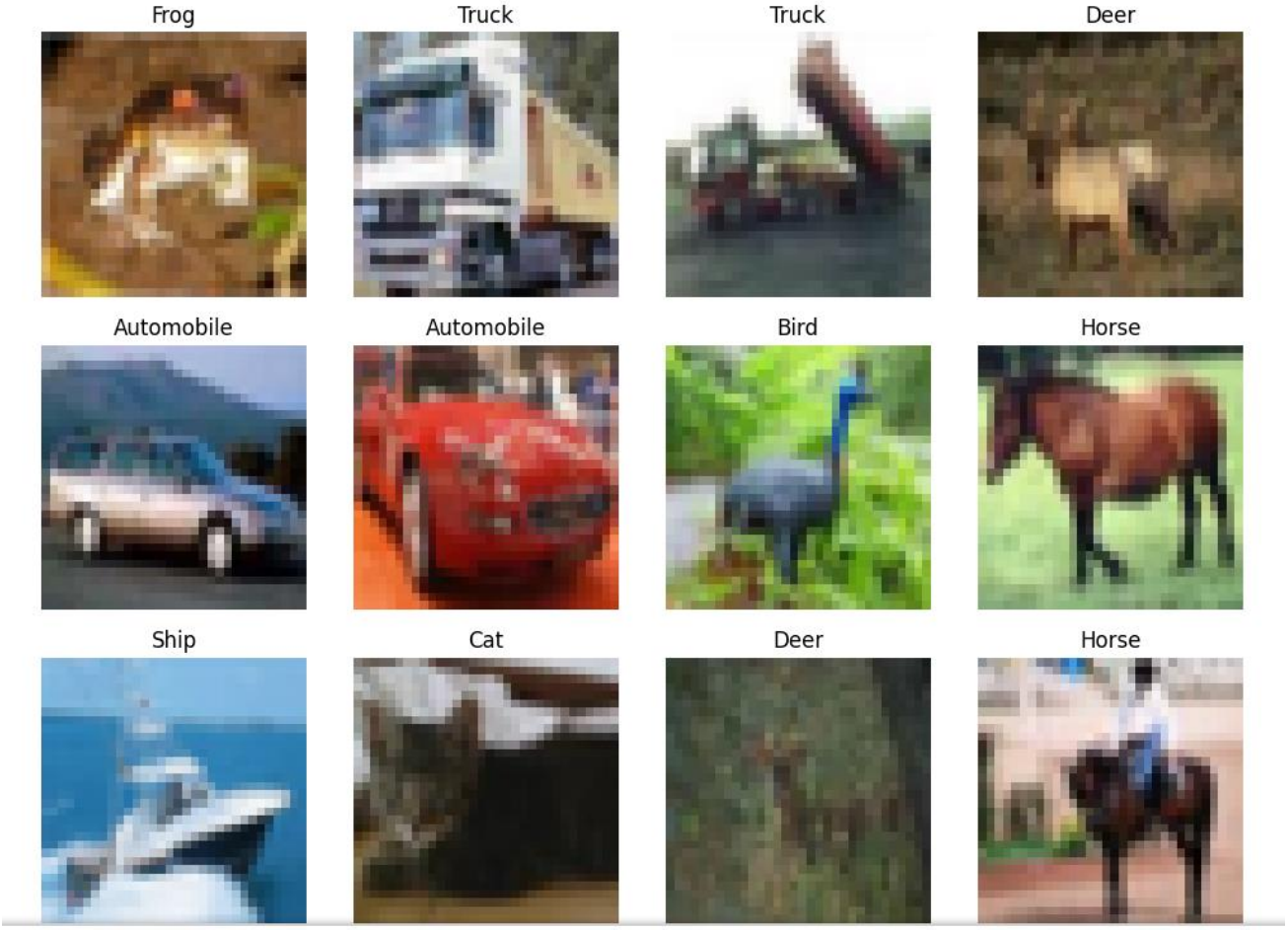
Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Methodology: Exploratory EDA

The methodology used is exploratory data analysis with visualisations. The data visualisation is important when you want to get to know your data and it helps with understanding your data and getting a clear understanding of what your data looks like. For this assignment the data was visualised using bar graphs, box plots, violin plots, pie charts, the ROC evaluation metrics and the correlation metrics for the models used. The models used are the random forest, the naives bayes, the decision tree and lastly the ensemble learning for all the models. Histograms were also used to visualise the data. The data was loaded and the class names to be visualised were selected which are ‘Airplane’. ‘Automobile’, ‘Bird’, ‘Cat’, ‘Deer’, ‘Dog’, ‘Frog’, ‘Horse’, ‘Ship’, and ‘Truck’. The images were then normalised and then a subset of the images was plotted to see how the images looked, and the first 16 images were visualised and the results are the screenshot below. The following libraries were used:

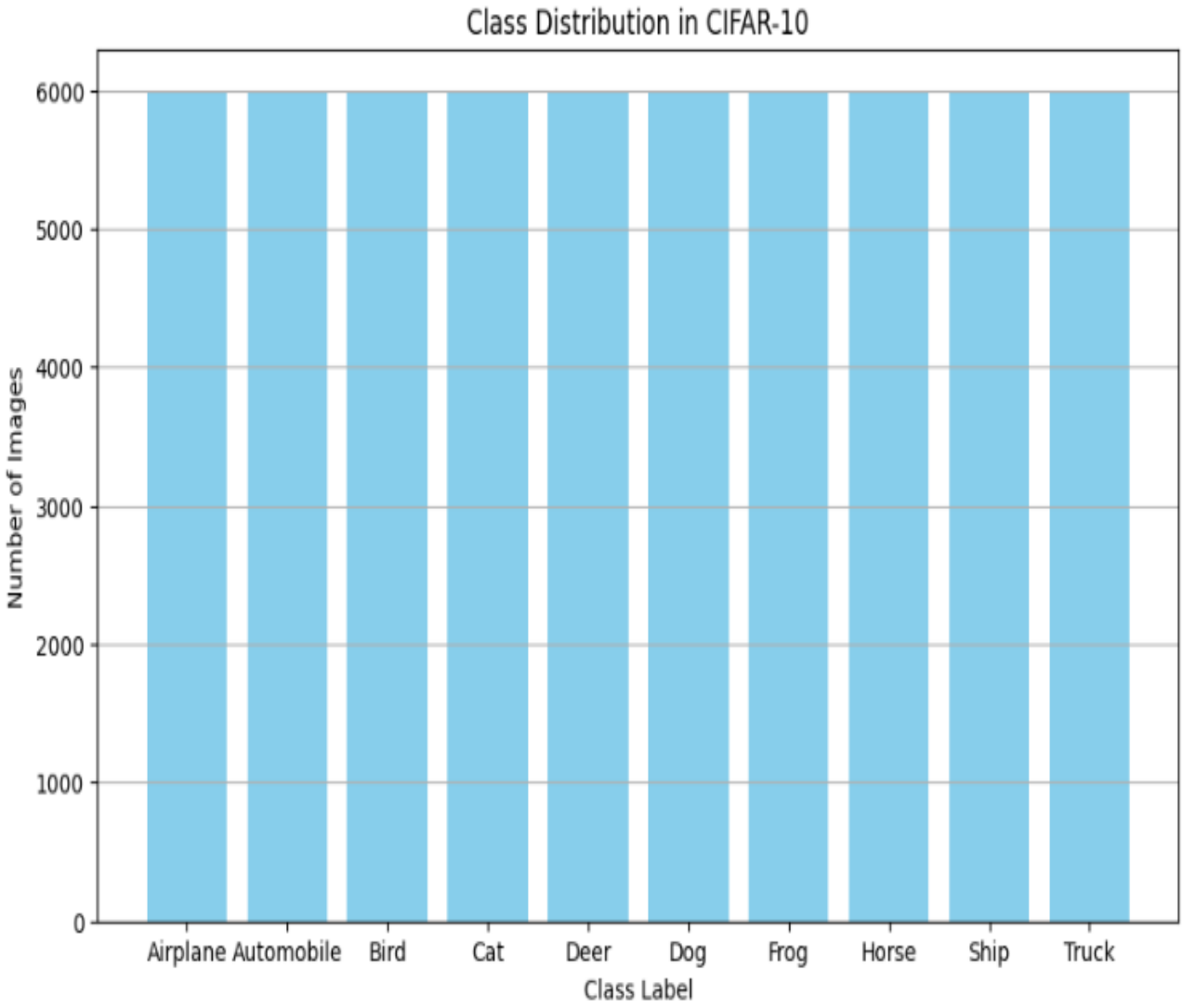
- Numpy
- Pandas
- Seaborn

- Matplotlib
- Tensorflow (keras)
- Scikit-learn



The graph below shows that the CIFAR-10 dataset has 10 classes, and each class has 6000 images. Therefore, 6000 x 10 is equal to 60000. The classes are:

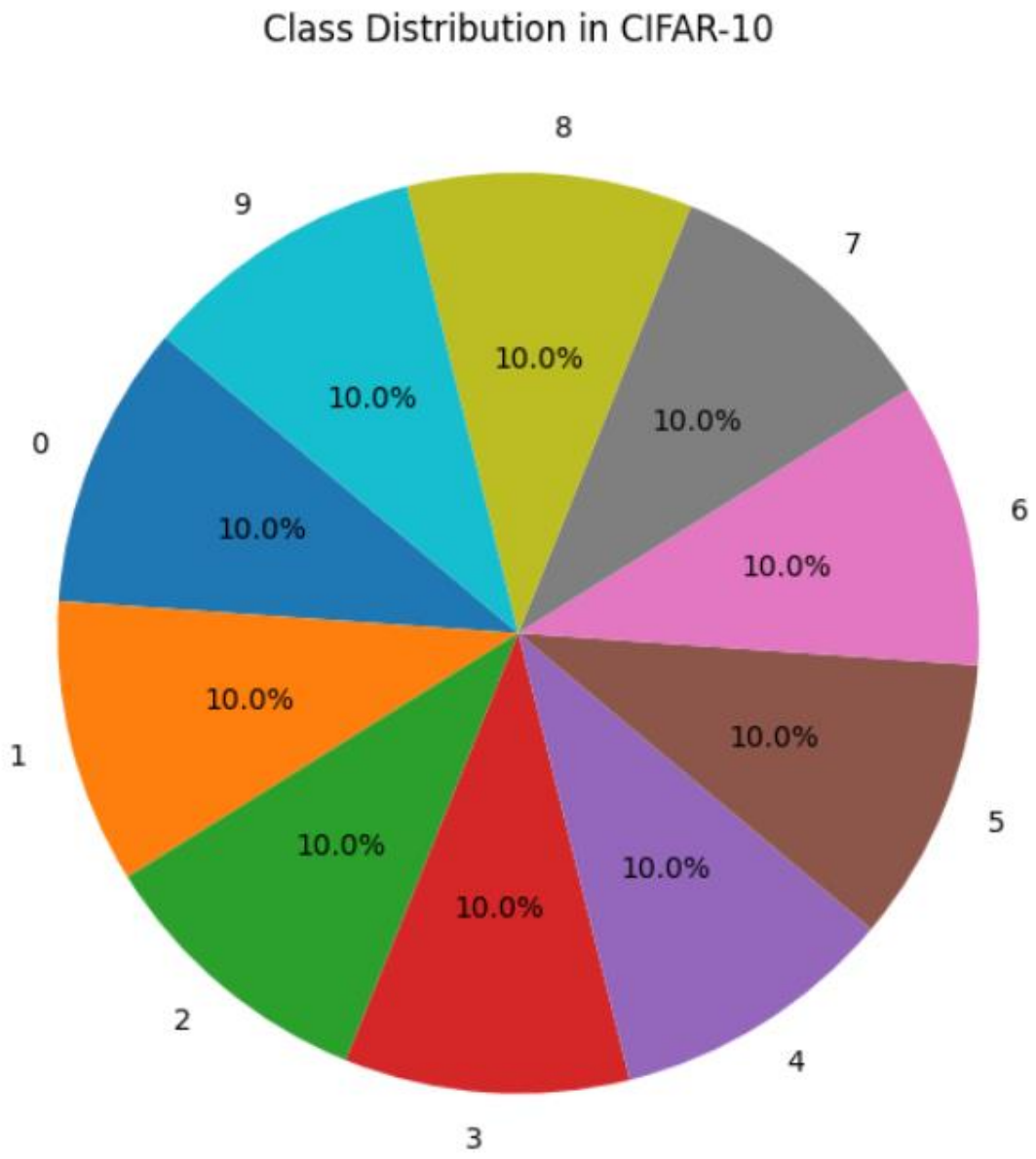
- Airplane
- Automobile
- Bird
- Cat
- Deer
- Dog
- Frog
- Horse
- Ship
- Truck

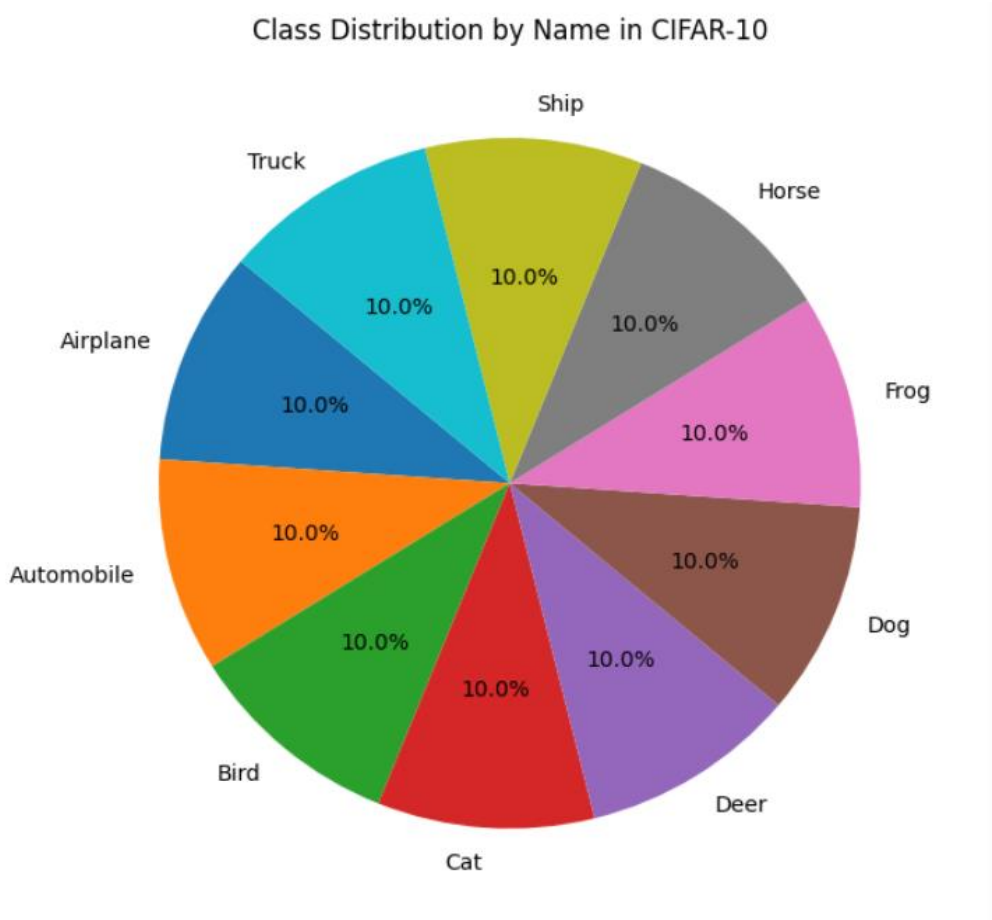


The class distribution pie chart shows the class as numbers and the class name have been encoded into numbers. Thus:

- Airplane = 1
- Automobile = 2
- Bird = 3
- Cat = 4
- Deer = 5
- Dog = 6
- Frog = 7
- Horse = 8
- Ship = 9
- Truck = 10

The pie chart below shows the distribution of the classes by their names. Each class accounts for 10% of the entire dataset and the addition of all the percentages that each class accounts.



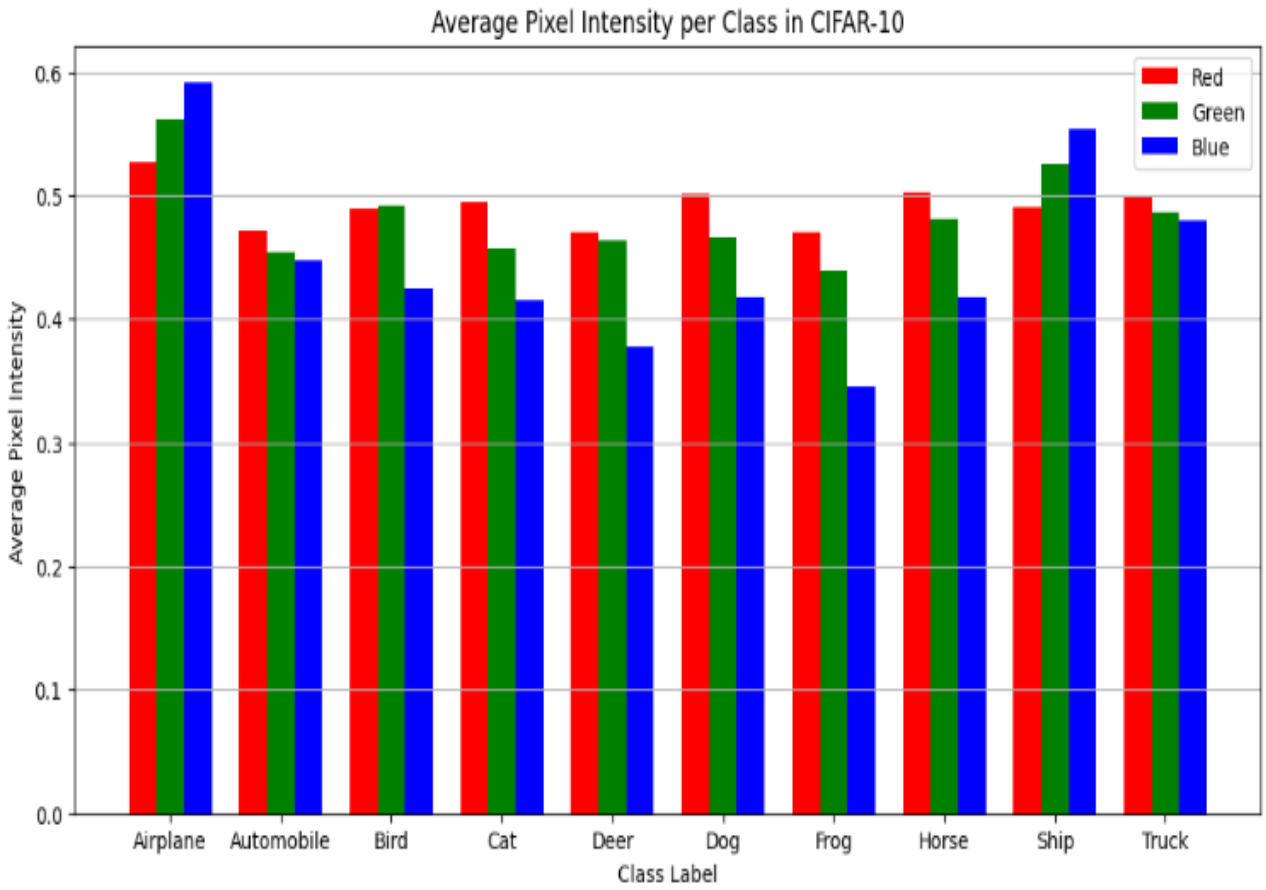


The CIFAR-10 dataset has images with the pixel intensities of the following colours:

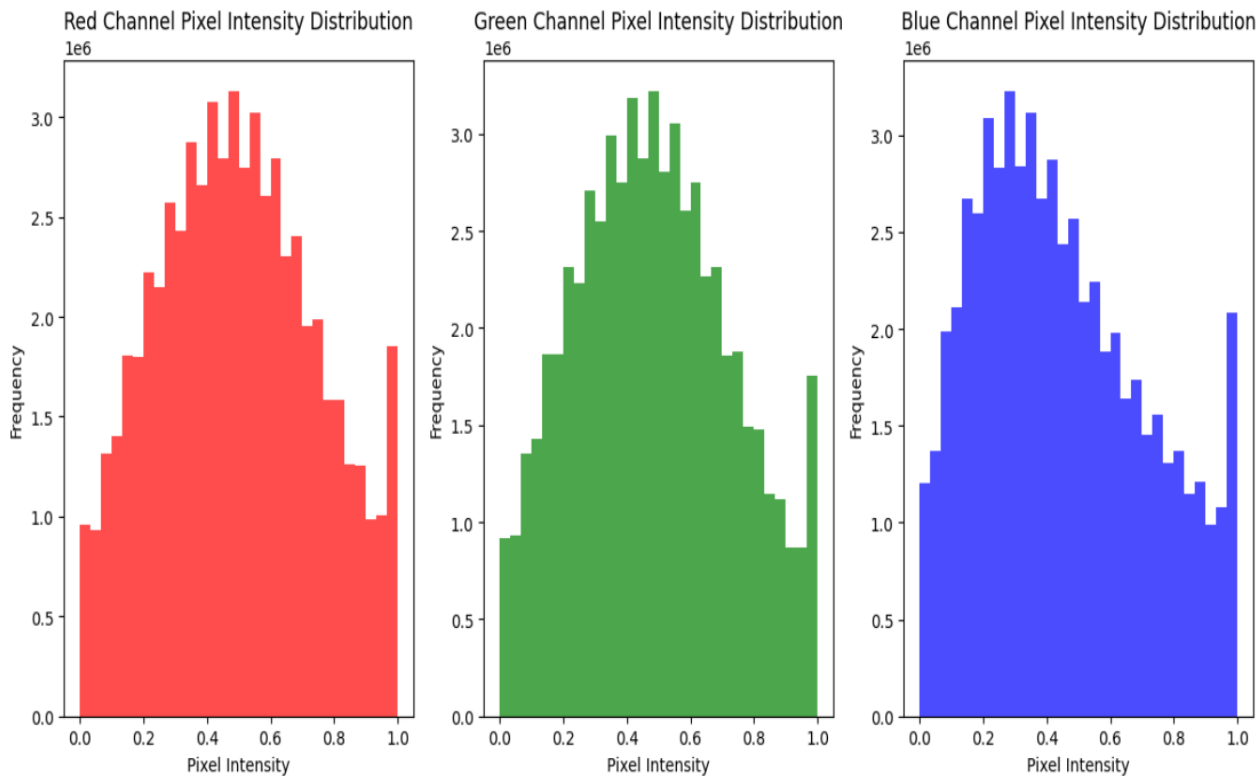
- Red
- Green
- Blue

Below is a bar graph that shows how the average of the colour Red, Green and Blue pixels images in a certain class have. In the CIFAR-10 dataset, the Airplane class has the highest pixel intensity across the Red, Green and Blue pixels and the Frog class has the lowest pixel intensity across the Red, Green and Blue pixels.

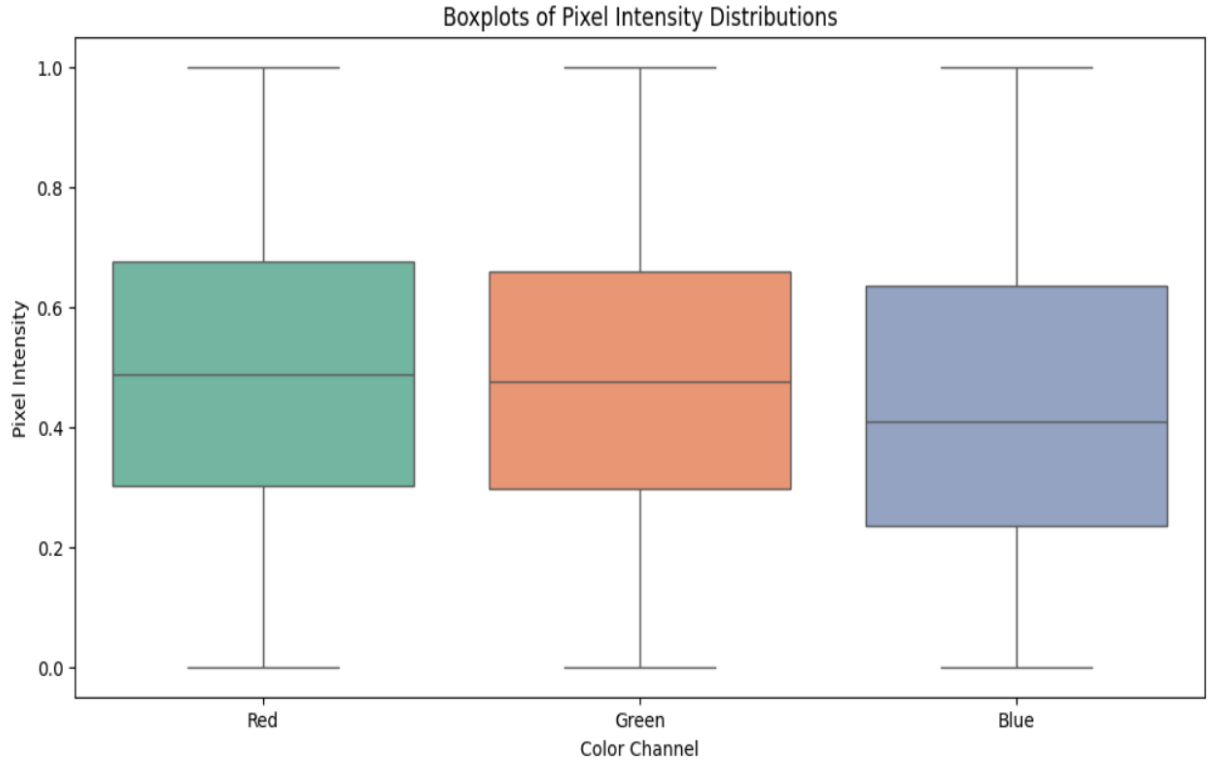
Statistical Analysis

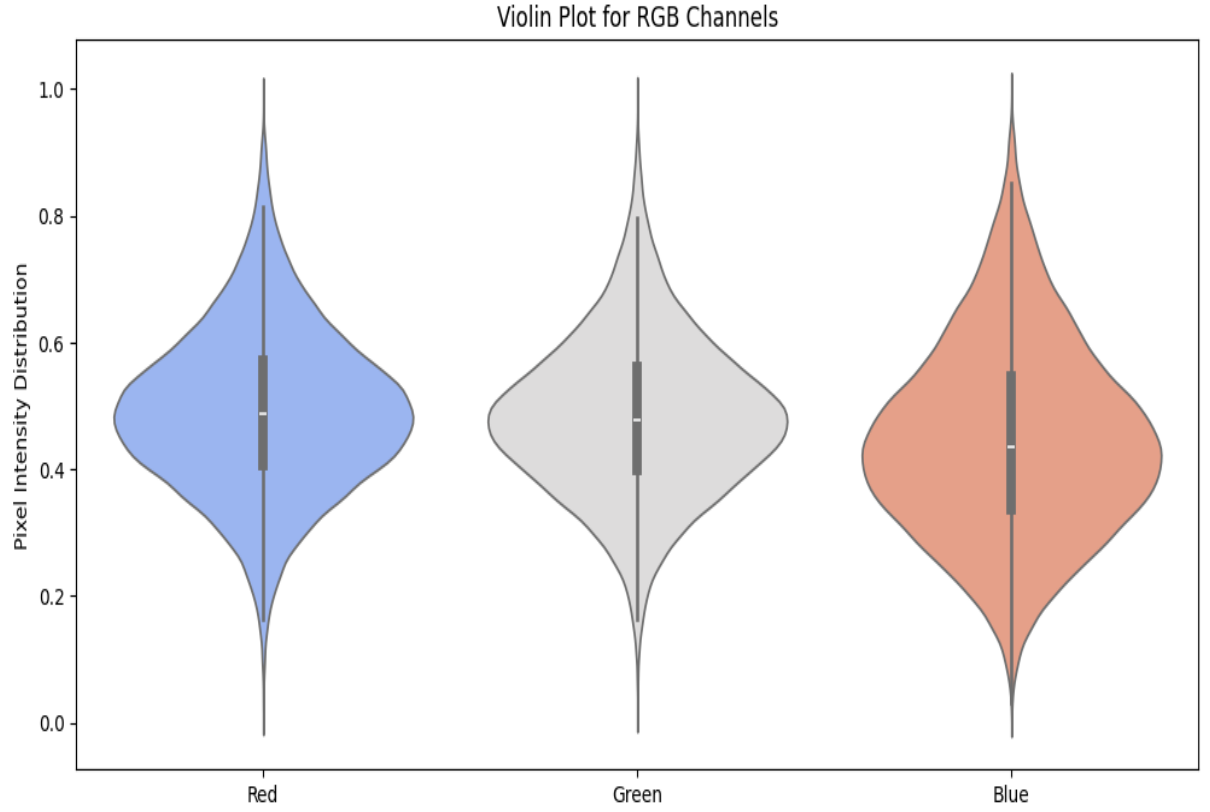


The dataset has images with pixels of different colours. These colours include Red, Green and Blue. As shown in the image below, the distribution of the intensity of images with Red and Green pixel intensities are more evenly distributed. However, the distribution of the intensity of images with the Blue pixel intensity is distributed to the left. In other words, it is negatively skewed. The distribution of the Blue channel intensity is higher in terms of frequency compared to the Red and Green channel distributions of the images in the CIFAR-10 dataset. The frequency represents how frequent images with a certain colour pixel intensity appear throughout the dataset.

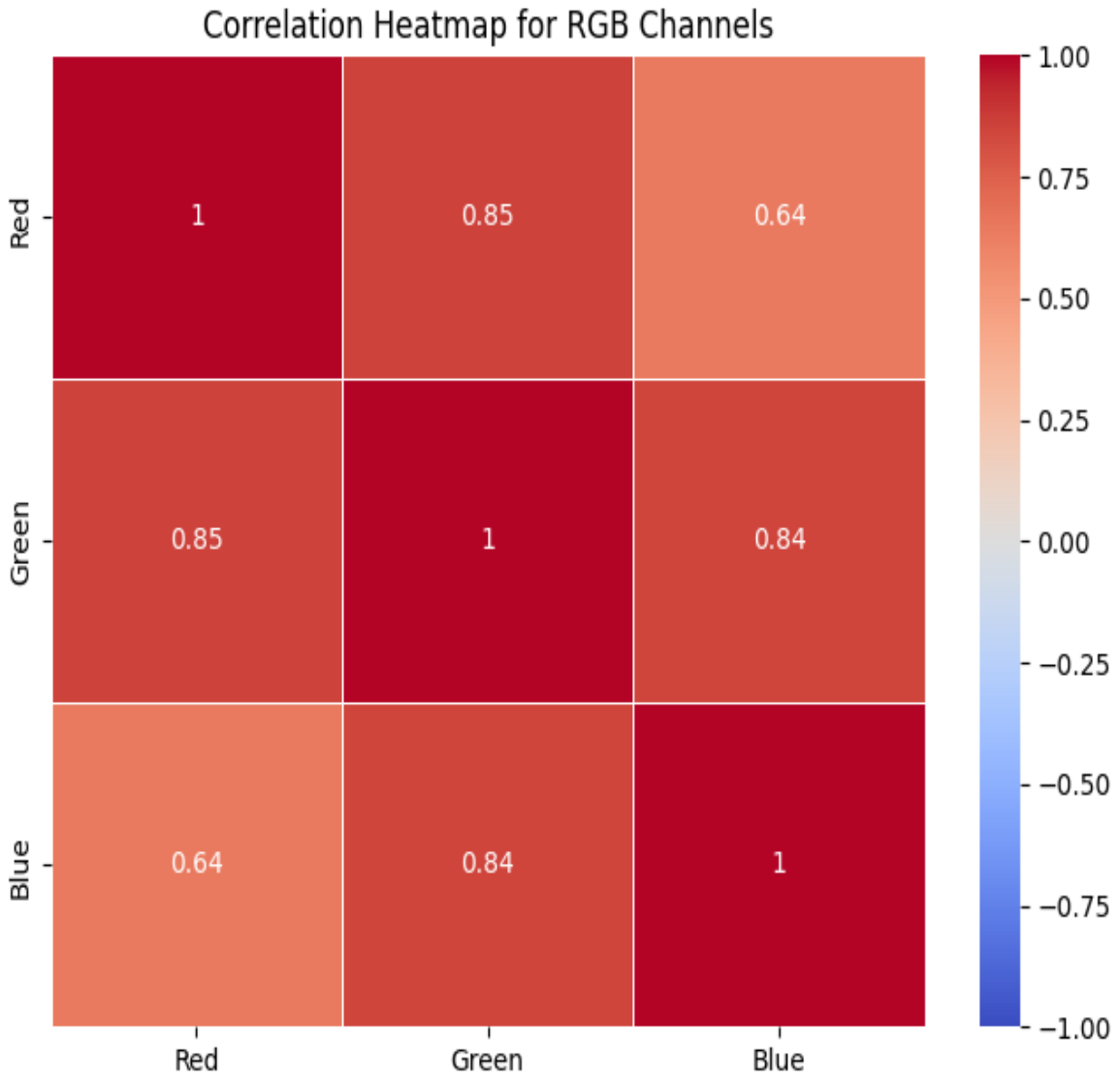


The following boxplot also visualises the pixel intensity distribution which shows the same results as the histograms. The Red and Green pixel intensity distribution box plots indicate more even distribution while the Blue pixel intensity distribution indicates negatively skewed distribution. And as seen below, the violin plot shows the same results as the histograms and the boxplots for the same colours except that the violin plot is for pixel intensity distribution against the RGB channel.


Boxplots of Pixel Intensity Distributions
Pixel Intensity
1.0
0.8
0.6
0.4
0.2
0.0
Red
Green
Blue
Color Channel


Violin Plot for RGB Channels
Pixel Intensity Distribution
1.0
0.8
0.6
0.4
0.2
0.0
Red
Green
Blue

Correlation Matrix



The correlation heatmap for RGB channels is very accurate as each colours has a score 1 which is very close to the value zero which indicates very high accuracy.

Random Forest

Random Forest Accuracy: 0.42

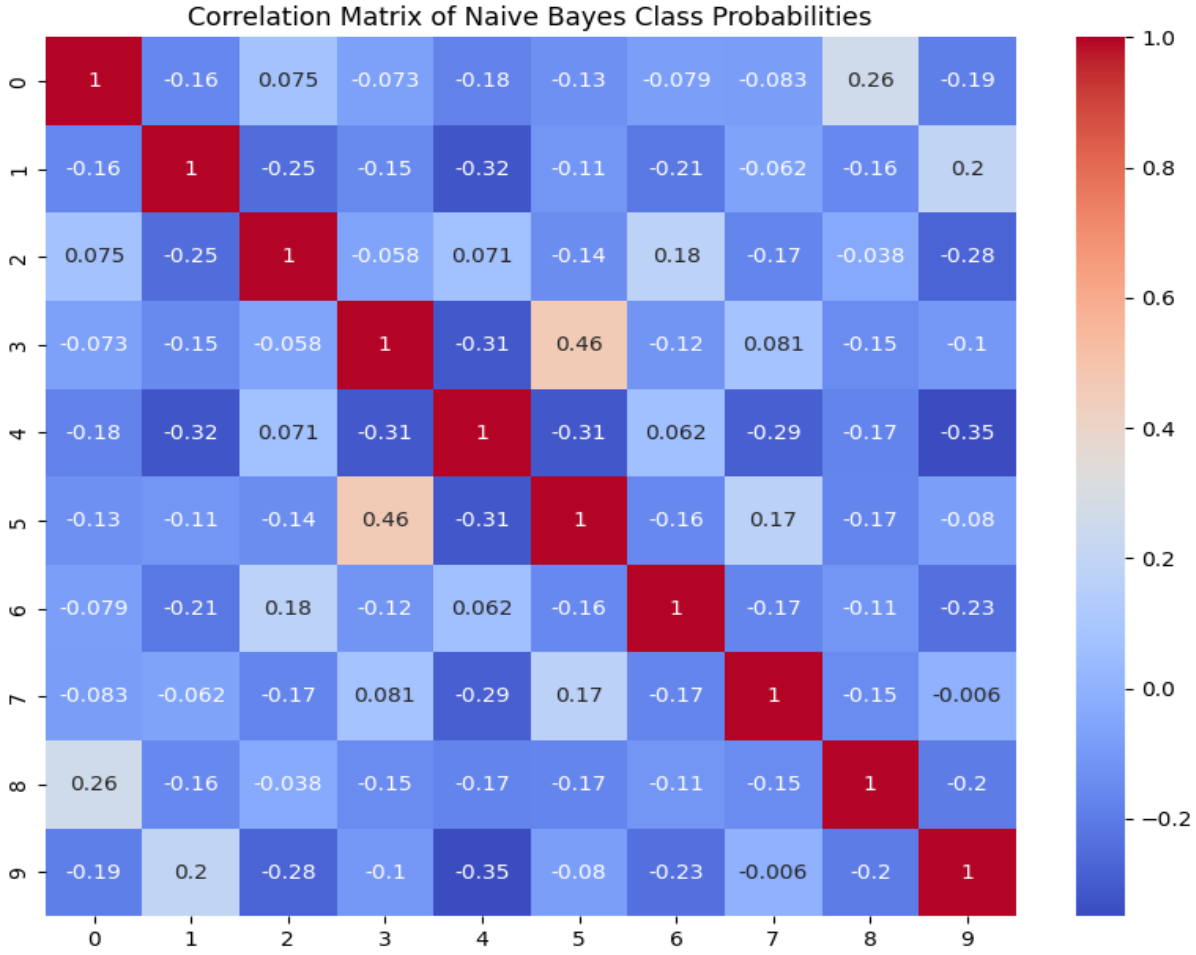
The Random Forest feature importances scored a score of 1 which indicate very high accuracy and the feature against importance and visa versa scored a score of -0,59 which is means that the correlation matrix of the random forest was very accurate. The value are considerably low indicating accuracy and precision of the random forest model. Moreover, there is a 1 in the top left which is a true positive and a 1 in the bottom right that the importance was correctly predicted and the feature was correctly predicted

Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.50	0.52	0.51	1181
1	0.48	0.56	0.52	1188
2	0.39	0.31	0.34	1232
3	0.34	0.27	0.30	1274
4	0.41	0.39	0.40	1179
5	0.40	0.38	0.39	1230
6	0.45	0.56	0.50	1123
7	0.52	0.43	0.47	1217
8	0.55	0.59	0.57	1189
9	0.45	0.51	0.47	1187
accuracy			0.45	12000
macro avg	0.45	0.45	0.45	12000
weighted avg	0.45	0.45	0.45	12000

Naïve Bayes

```
Naive Bayes Accuracy: 0.34
```

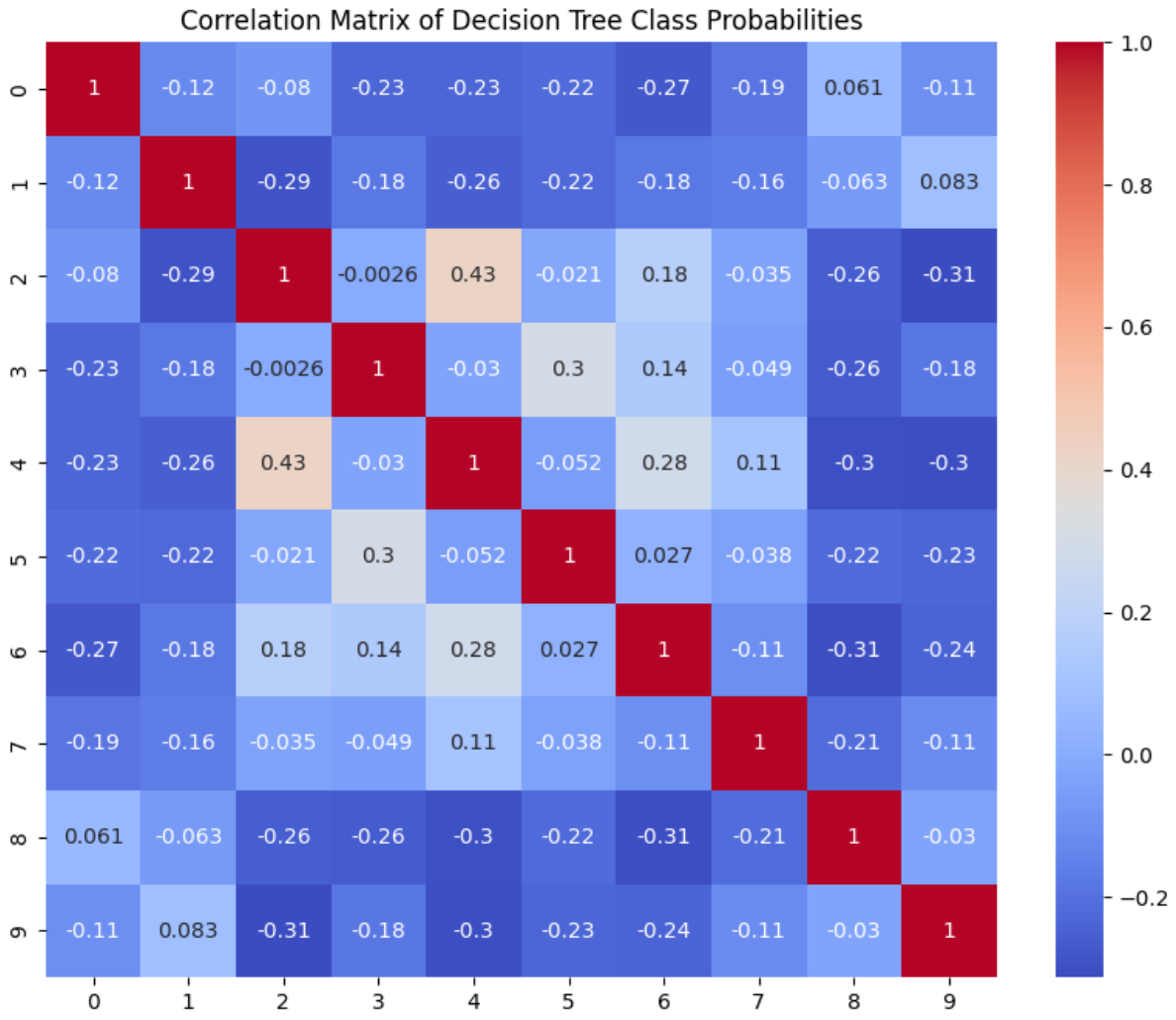


Naive Bayes Classification Report:

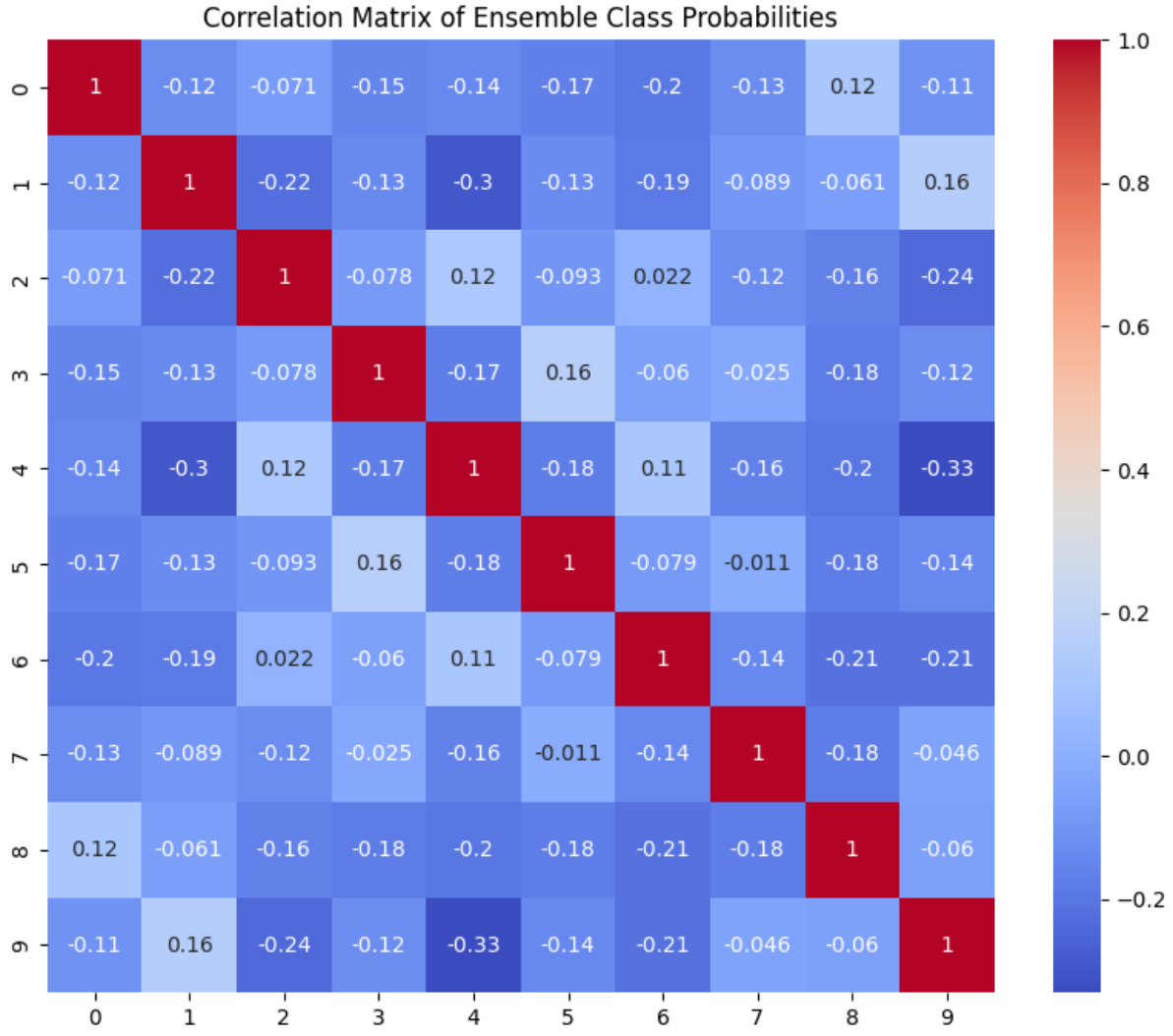
	precision	recall	f1-score	support
0	0.42	0.23	0.30	1181
1	0.42	0.47	0.44	1188
2	0.21	0.11	0.14	1232
3	0.27	0.19	0.22	1274
4	0.23	0.59	0.33	1179
5	0.37	0.24	0.29	1230
6	0.42	0.32	0.36	1123
7	0.44	0.34	0.39	1217
8	0.42	0.40	0.41	1189
9	0.34	0.48	0.40	1187
accuracy			0.34	12000
macro avg	0.35	0.34	0.33	12000
weighted avg	0.35	0.34	0.33	12000

Decision Tree

```
Decision Tree Accuracy: 0.31
```



```
Ensemble Learning Accuracy: 0.31
```



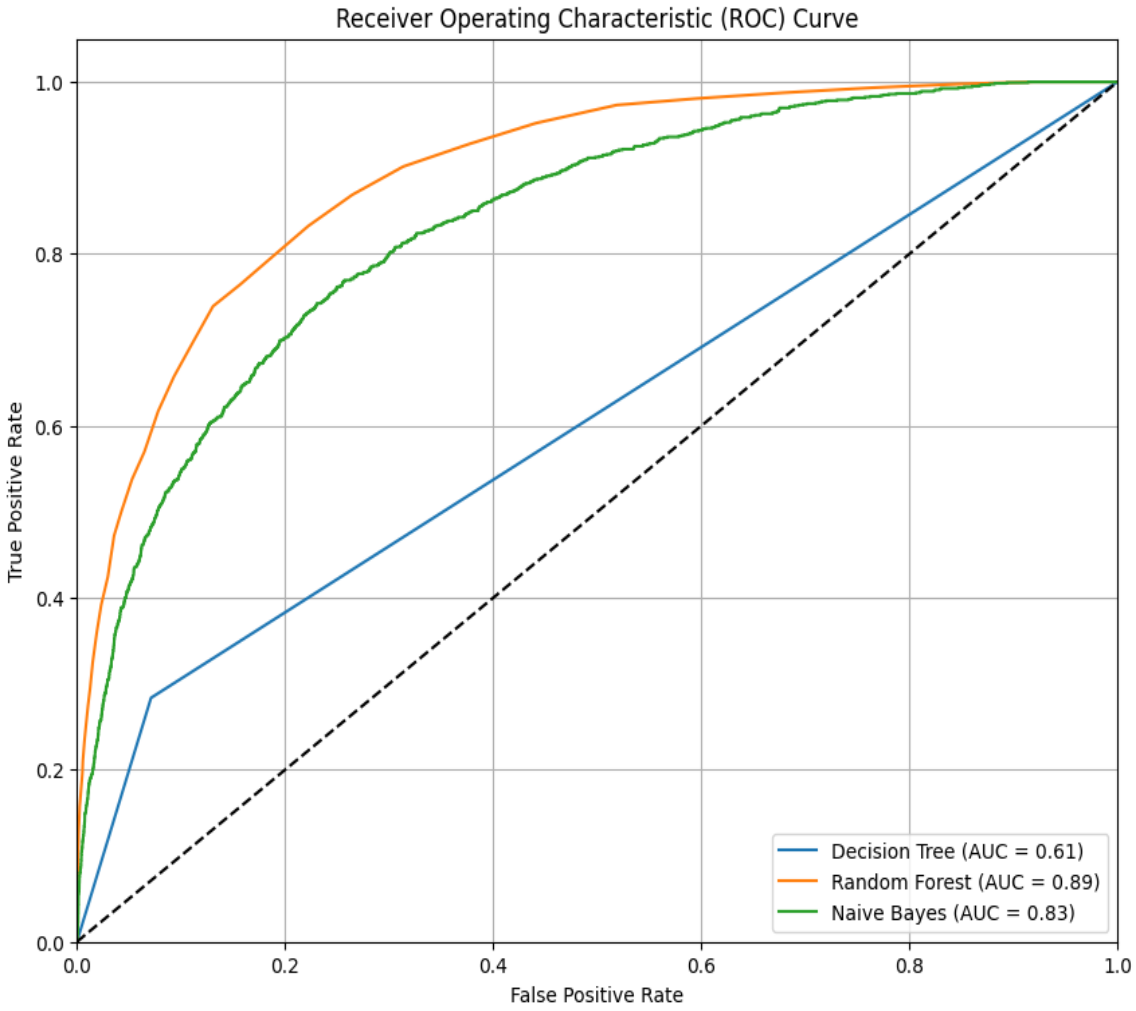

```
Ensemble Learning Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.36      0.42      1181
           1       0.42      0.50      0.46      1188
           2       0.31      0.15      0.21      1232
           3       0.31      0.20      0.24      1274
           4       0.26      0.58      0.36      1179
           5       0.40      0.28      0.33      1230
           6       0.44      0.37      0.41      1123
           7       0.48      0.36      0.41      1217
           8       0.49      0.52      0.51      1189
           9       0.37      0.52      0.44      1187

    accuracy                           0.38     12000
   macro avg       0.40      0.39      0.38     12000
weighted avg       0.40      0.38      0.38     12000
```

The Naïve Bayes, Decision Tree and Ensemble models all scored a score 1 across all the classes in the dataset which is a positive meaning that the models were very accurate and that all the classes were moving together in a perfect positive linear relationship which there was no deviation and that their relationship is predictable.

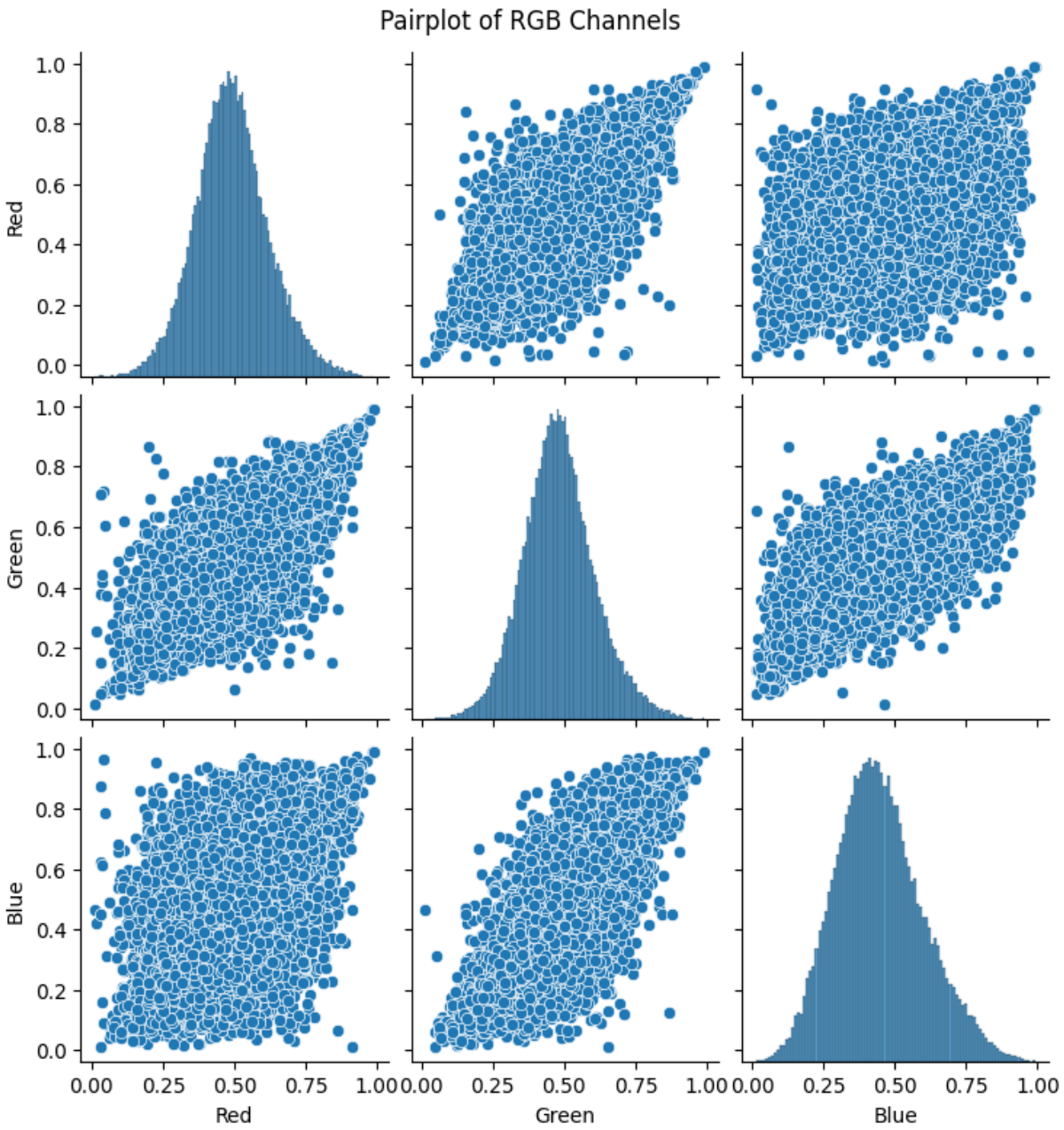
This ROC (Receiver Operating Characteristic) curve visualises the performance of three different classifiers: The algorithms used are Decision Tree, Random Forest and Naive Bayes. The curve aims at showing the true positive, which is the sensitivity of the models, against the false positive on the x-axis. In this plot, the Random Forest model (orange curve) achieves the most accurate result with the biggest AUC of 0.89 as an evidence. The Naive Bayes model (green curve) comes next with an AUC of 0.83 which shows a relatively compact form with slightly less accuracy than Random Forest. The Decision Tree model (blue curve) has a comparatively poor performance with the AUC of only 0.61 that testifies to relatively low capability of this model to distiguish between the classes. The dotted diagonal line at the bottom of the chart shows the performance of a perfectly random classifier cone at AUC of 0.5, Because all models do better than this baseline, all these models are contributing useful predictive ability. In all, it is seen that the Random Forest classifier is the best classifier among the ones in consideration and the Naive Bayes classifier is the second-best classifier, and the Decision Tree is the worst of the lot.

Code explanation for ROC Evaluation

For the ROC evaluation metrics graphs, the data was loaded, then the test and training data was combined using the concatenate() function with the test data and training data as the parameters being passed into it and normalised. The data was then reshaped into 2D and split into training and testing set. The dimensionality was reduced using PCA and then the classifiers were initalised for random forest, decision tree and naïve bayes. The predicted probabilities were fetched, and the ROC curve and AUC was calculated for each class. The ROC curve and ROC details were then plotted.

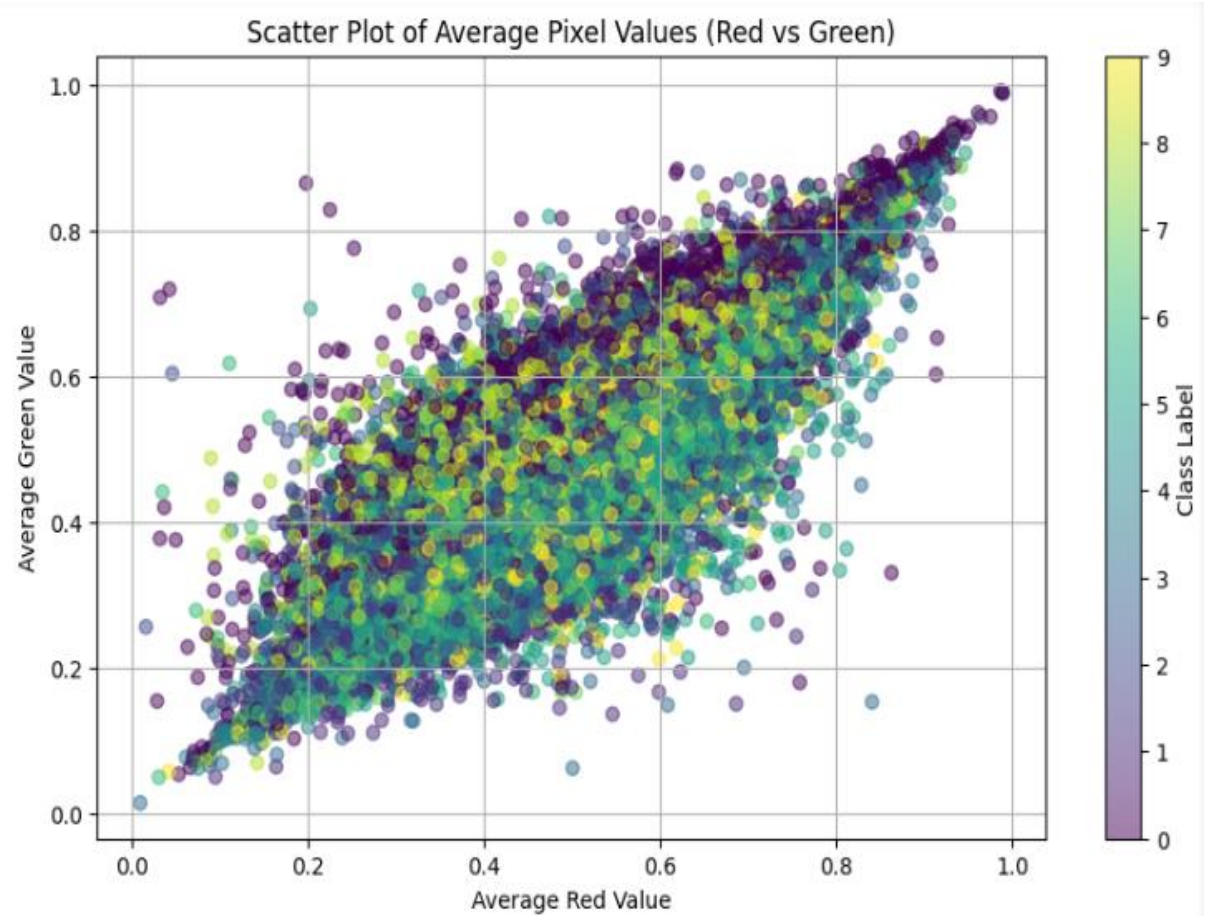


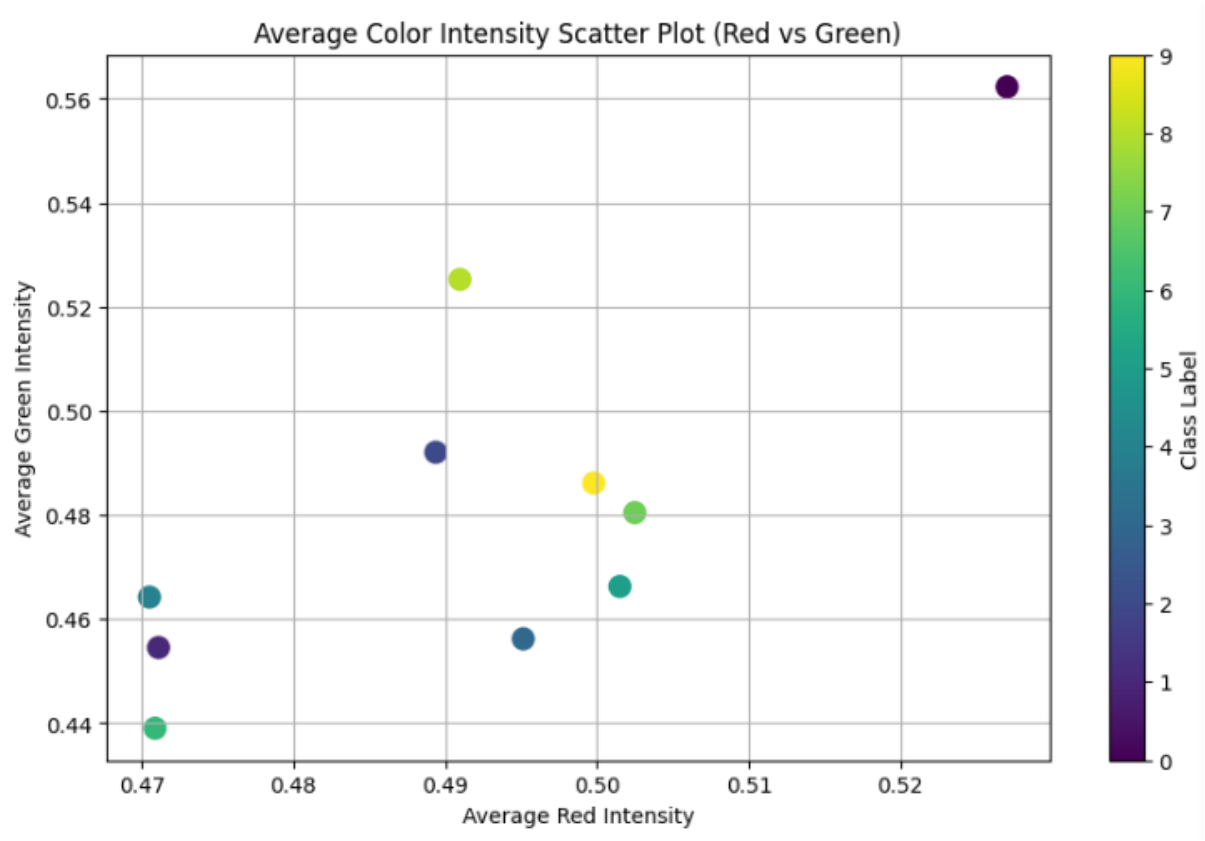
The below image is an image of a pairplot of the RGB Channels. The Red to Red graph shows an even distribution similar to the Green to Green graph. However, the Blue to Blue graph shows that the graph is skewed to the left showing a negative distribution. The Red to Blue distribution channel has more dots on it and its distribution is very wide showing that compared to images that have more the colour Red in them, there are more images that have Blue in them. However, the scatter plot showing the Blue and Red channel shows somewhat an even distribution. Overall, the pairplot shows that the Red and Blue channel is more common across the images in the dataset which is similar to the Blue and Red channel.



Red vs Green

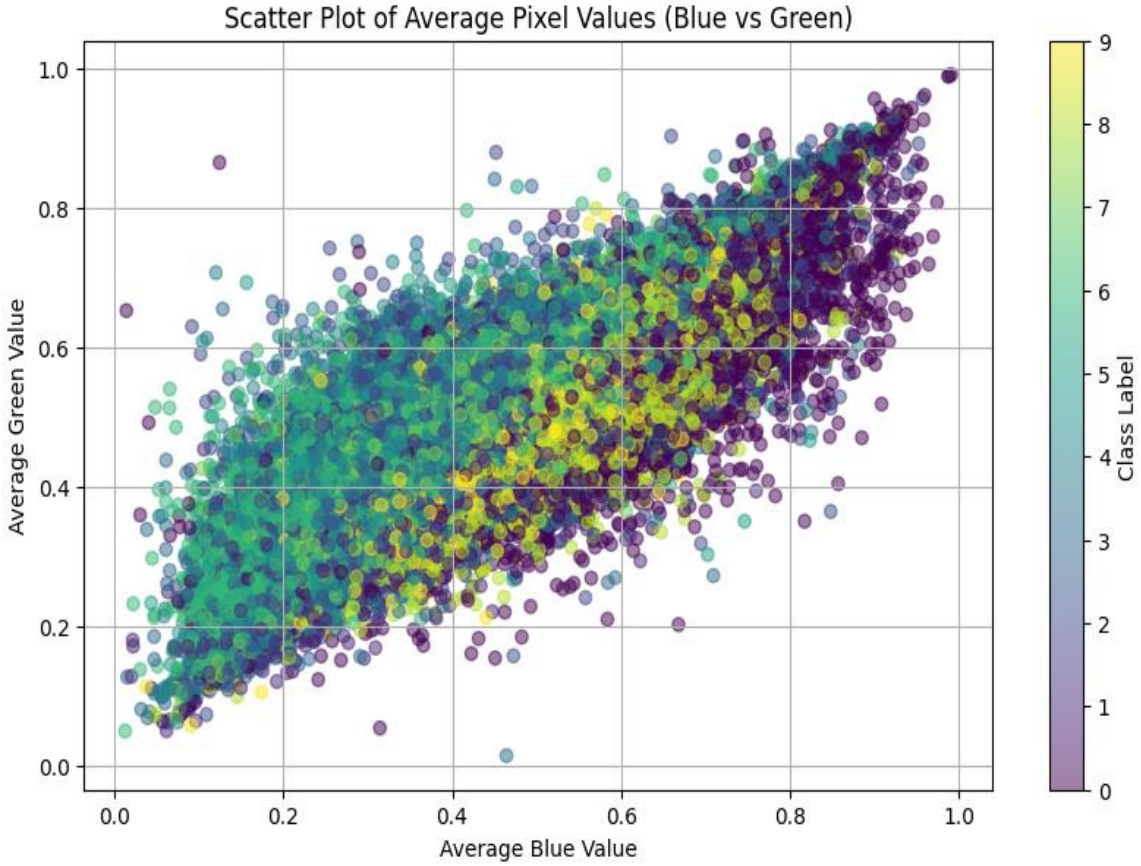
Based on the image below it is evident that the Classes from 3 to 4 are the ones that have the more images that have the Red colour pixels. Followed by images in the classes 0 to 4. This shows that the classes 7 to 9 have few images that have the Red colour pixels compared to the other images in the classes ranging from 0 to 7.

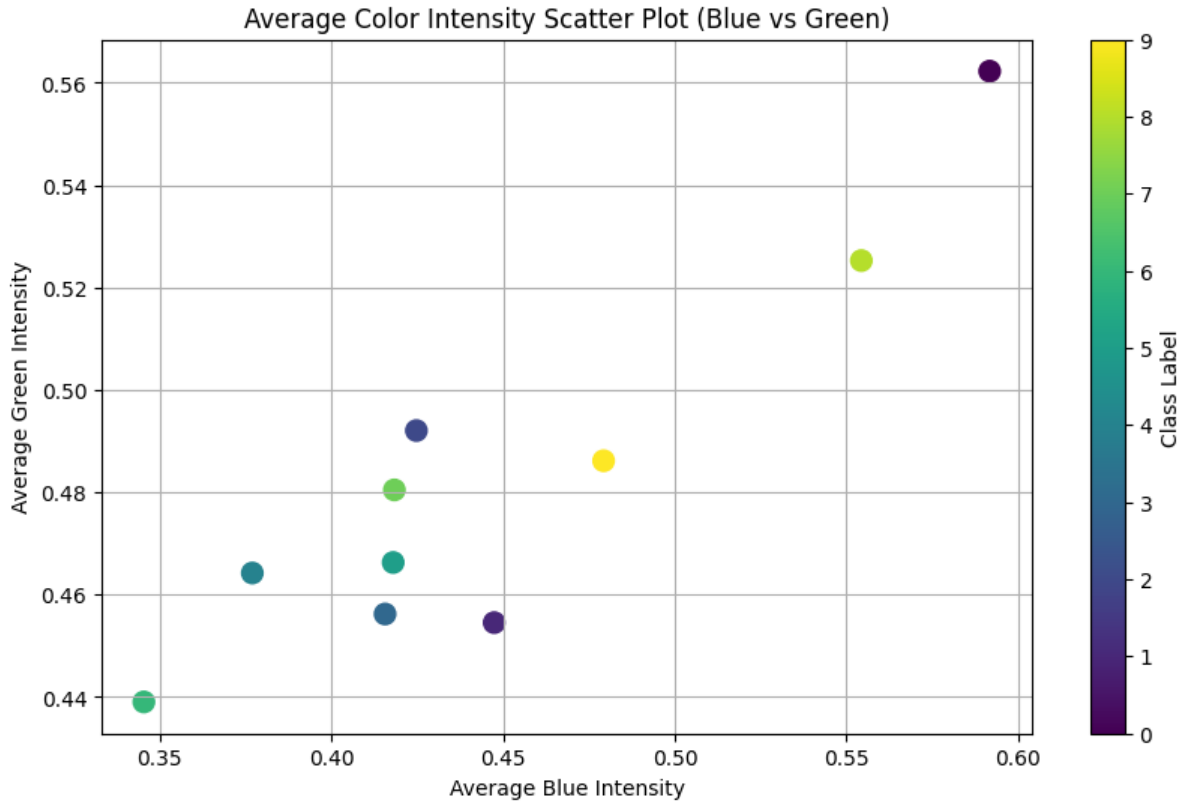




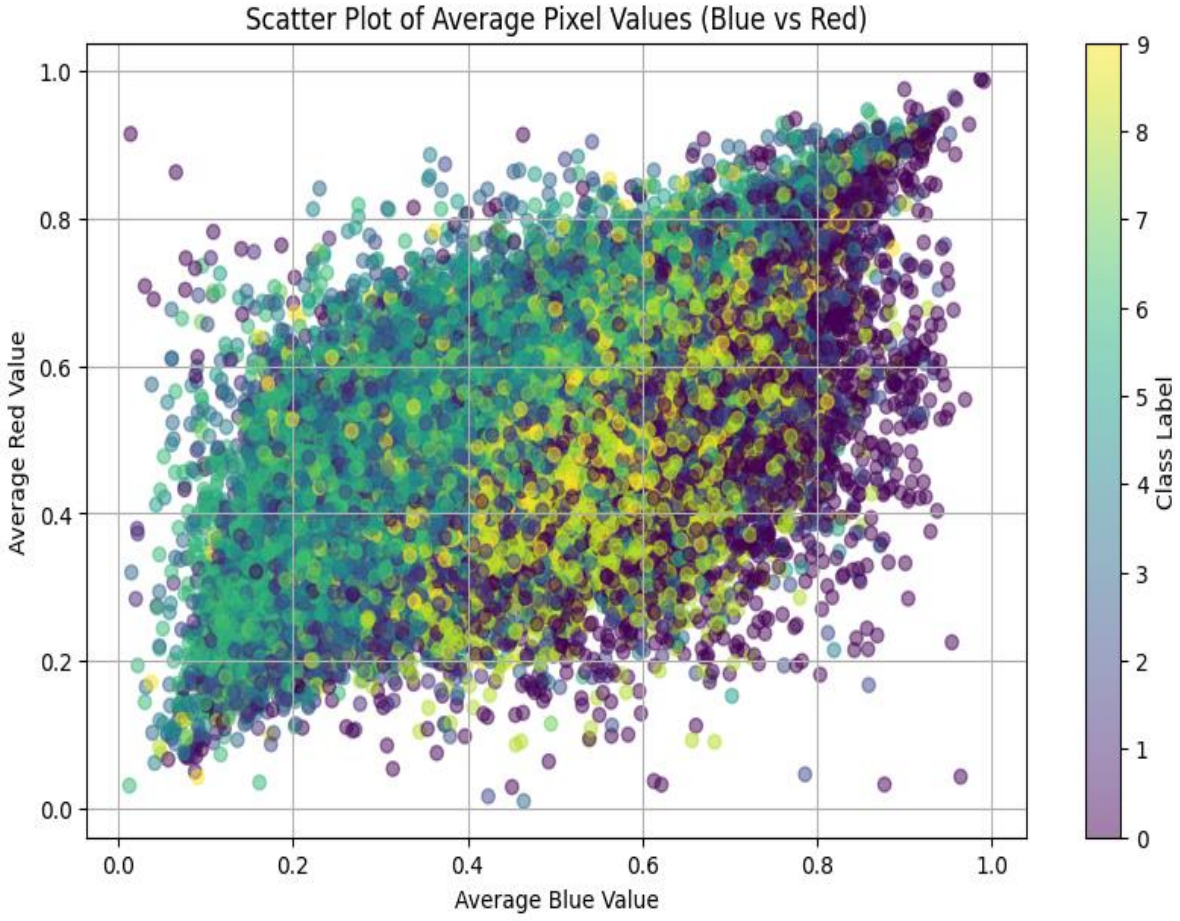
Blue vs Green

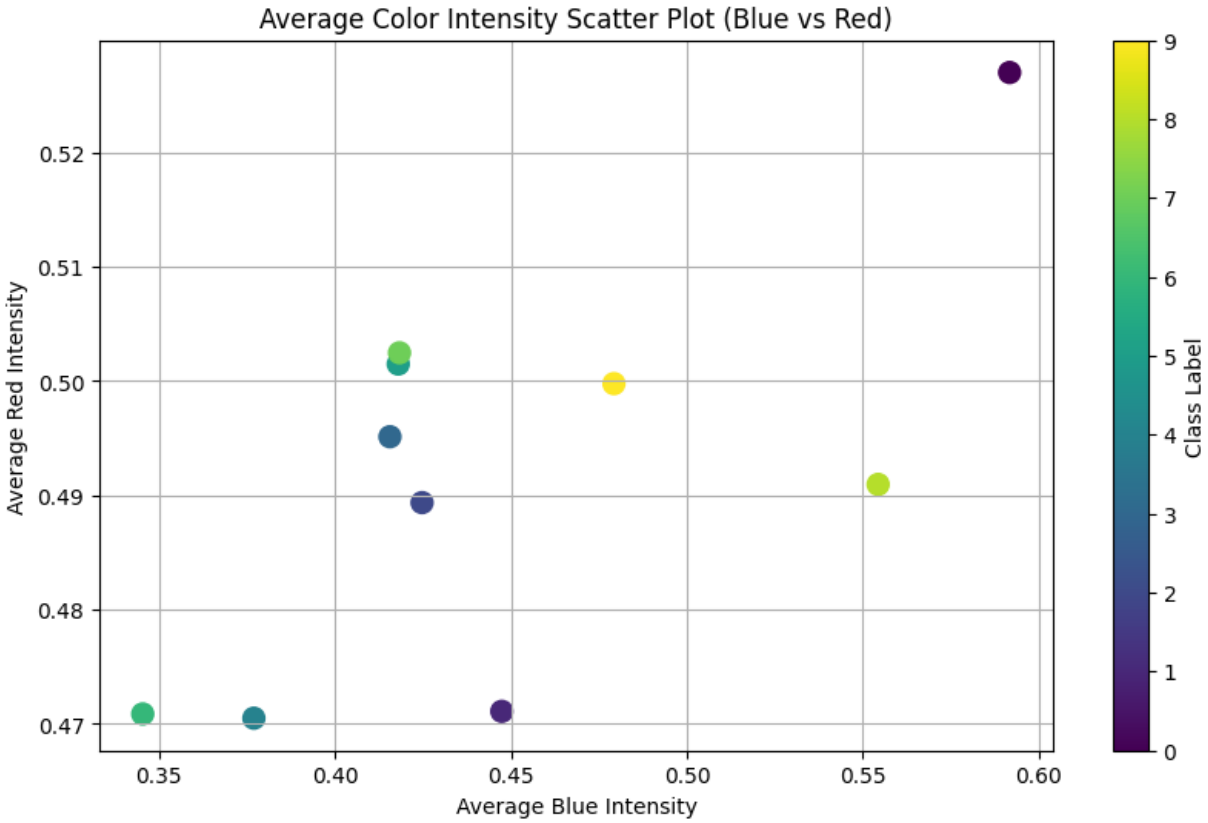
The graph below shows that the most images in the classes ranging from 0 to 2 have more pixels with the colours Red and Green. images in the classes 3 to 6 have less pixels of the colour Blue and pixels of the colour Green. In order words the CIFAR-10 dataset needs more images in the classes 3 to 6 with the colour Green. However, the images of the classes 7 to 9 with the colour pixels of both Blue and Green are very few judging by the few dots on with the colour yellow in the graph.




Average Color Intensity Scatter Plot (Blue vs Green)
Average Green Intensity
Average Blue Intensity
Class Label
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.35
0.40
0.45
0.50
0.55
0.60
9
8
7
6
5
4
3
2
1
0

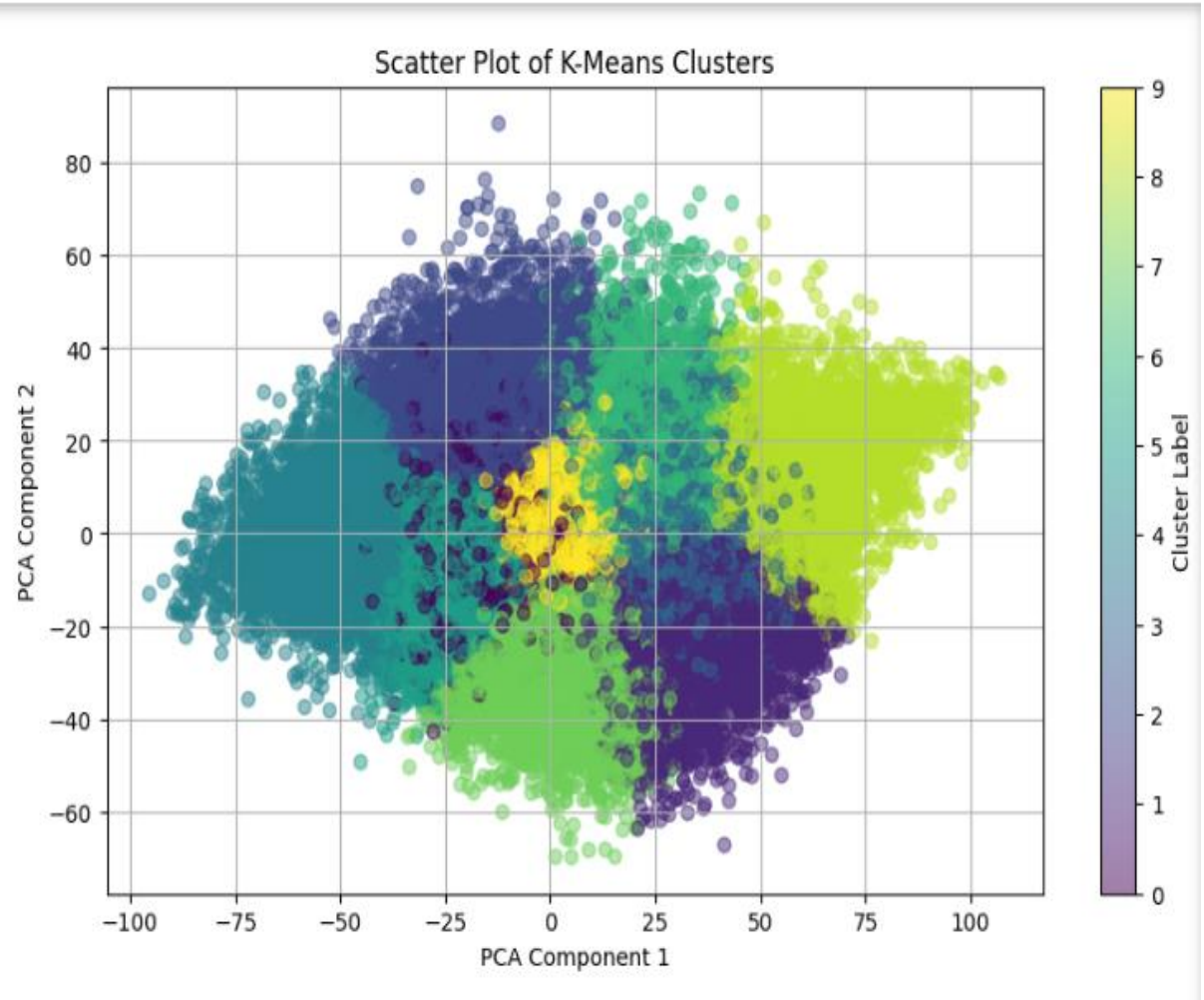
Blue vs Red


Scatter Plot of Average Pixel Values (Blue vs Red)
Average Red Value
Average Blue Value
Class Label
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
9
8
7
6
5
4
3
2
1
0



K-Means Clustering

The image below is the K-means clustering graph and as seen from the graph shows all the clusters of all the classes found in the Cifar-10 dataset along with their PCA Component 1 and PCA Component 2.



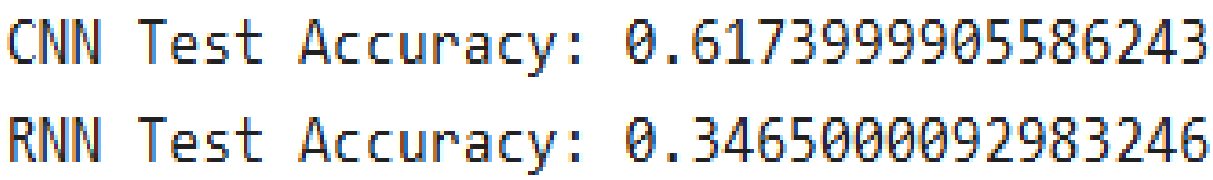
The scatter plot shows a K-Means clustering algorithm, dividing the data into ten distinct clusters. The plot shows some overlap between clusters, especially in the central region where colours blend. Outer regions, like green and light yellow, have more isolated clusters, while the center, particularly dark blue and purple, has significant overlap which suggests that less clear distinction between clusters. The plot's colour bar indicates different clusters.

RNN and CNNS AutoEncoder Explanation

All the necessary libraries are imported and the dataset is loaded, then the data is normalised into float type both the training and testing data using the astype(). Then the data is reshaped for the RNN autoencoder model and the split into train and validation sets. After that the CNN model is implemented using the sequential() method where the input is adjusted. Then same is done for the RNN autoencoder model. After that the both models are evaluated on test set using the .evaluate() method. Then the results are printed using the print() function. Then the training history is plotted with the epochs on the x axis and the accuracy on the y axis.

RNN and CNNS AutoEncoder Results

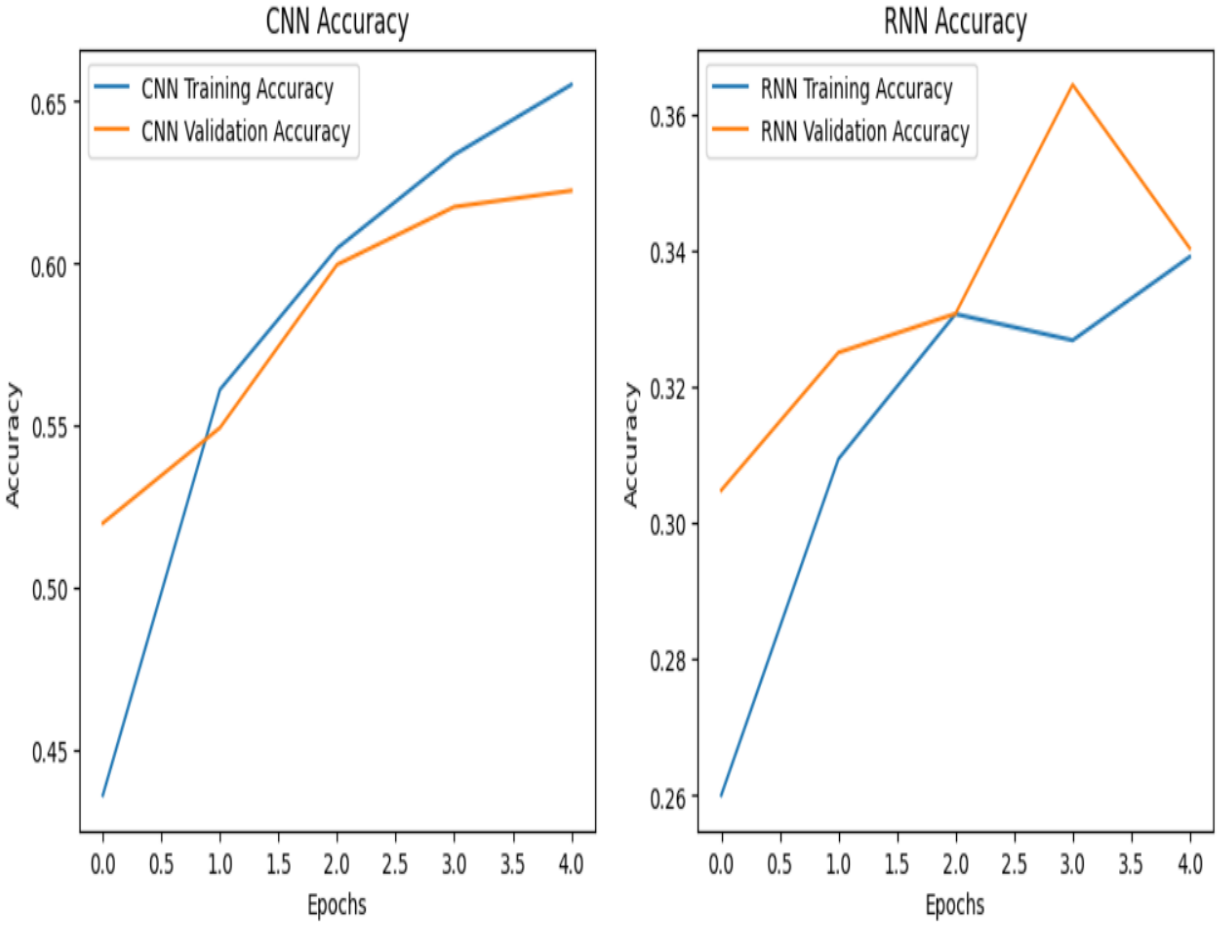
```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.3515 - loss: 1.8155 - val_accuracy: 0.5200 - val_loss: 1.3455
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.5535 - loss: 1.2735 - val_accuracy: 0.5494 - val_loss: 1.2559
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.6025 - loss: 1.1467 - val_accuracy: 0.5997 - val_loss: 1.1645
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.6333 - loss: 1.0628 - val_accuracy: 0.6175 - val_loss: 1.1012
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.6578 - loss: 0.9856 - val_accuracy: 0.6225 - val_loss: 1.0943
Epoch 1/5

313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.2197 - loss: 2.1200 - val_accuracy: 0.3048 - val_loss: 1.8969
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3033 - loss: 1.9021 - val_accuracy: 0.3250 - val_loss: 1.8318
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3384 - loss: 1.8114 - val_accuracy: 0.3307 - val_loss: 1.8534
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3216 - loss: 1.8541 - val_accuracy: 0.3643 - val_loss: 1.7380
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3382 - loss: 1.8050 - val_accuracy: 0.3403 - val_loss: 1.8063
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6214 - loss: 1.0827
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.3491 - loss: 1.7925
```



ResultsCIFAR10CNN Test Accuracy: 0.6173RNN (MLLSTM) Test Accuracy: 0. 3465. CNN Test Accuracy: 0.0512 RNN Test Accuracy: 0.0833

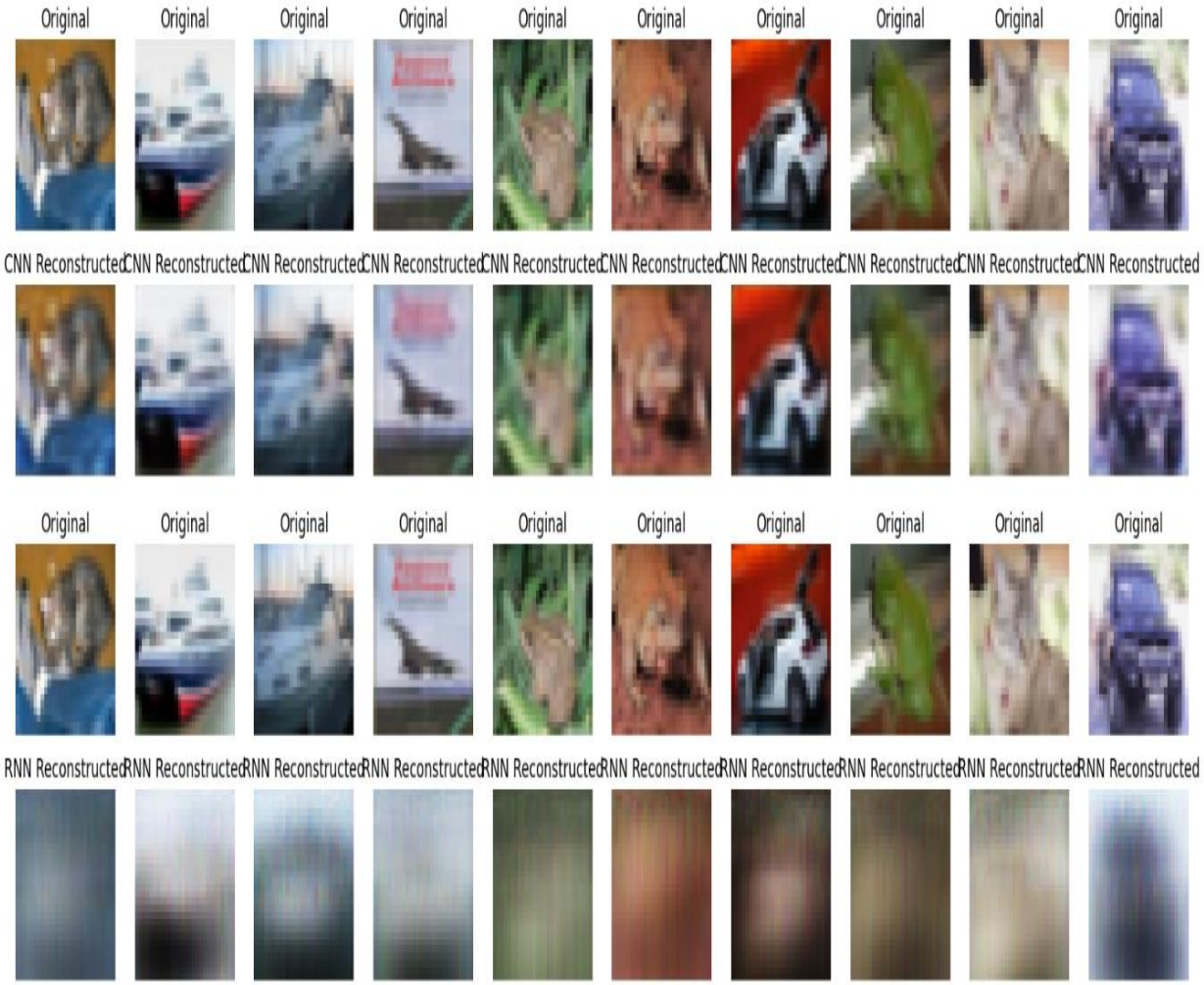
CNN model gives Training and Validation accuracy with quite difference to each other, but it is improving rapidly after each iteration. Training accuracy improves more gradually (from ~0.45 to around 0.65 by epoch 4) Validation accuracy also increase; it starts around 0.52 and plateau at the middle of training to about 0.60. Which indicates model is learning well, there seems to be no strong sign of overfitting since the training and validation are relatively close. On the other hand, with RNN model, the training process seems to be not so stable. It's accuracy, which begins hovering around 0.28, slowly and drastically rises to about 0.34 by epoch 4. The validation accuracy does start higher at.32 and reach a peak of around.36 by epoch 2 but fall slightly after that which suggests the model may not be overfitting well beyond

2 epochs. The separation between train and validation accuracy after 2 epochs indicates that the RNN may be overfitting to the training data or that it was originally unstable in learning its pattern. Considering that the accuracy is seemingly tracked more consistently in this way across passed epochs (more normal and straight lines) we can conclude overall from this simple visualisation, that the CNN will be outperforming the RNN.



On the next step, the data was loaded from the CIFAR-10 dataset and then normalised. After the normalisation, the data images were resized to 28 x 28. The data was then reshaped for CNN and RNN. After that the CNN autoencoder model and the RNN autoencoder model were created, followed by a reconstruction of images using both the CNN and RNN autoencoder models. Then the original and reconstructed images were plotted. Below are the results of the implementation of the LSTM encoders and as seen from the image that original images are more visible compared to the constructed images which are very blurry and not that visible. Moreover, the accuracy for the RNN and CNN autoencoder models was still considerably low. The results are the original images from each class followed by the CNN and RNN reconstructed images. The resulting images are one image from each class.

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 7s 20ms/step - loss: 0.6086 - val_loss: 0.5632
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.5607 - val_loss: 0.5606
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 7s 21ms/step - loss: 0.5593 - val_loss: 0.5595
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.5577 - val_loss: 0.5588
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - loss: 0.5578 - val_loss: 0.5586
```



For the LSTM Encoder

The data from the CIFAR-10 dataset was loaded using the load.data() method and pre-processed. The images were the resized to a 28 x 28 dimensions. The CNN autoencoder model was defined using the cnn_autoencoder = sequential() method, followed by the defining of the RNN autoencoder model which was defined using the rnn_autoencoder = sequential() method. After that the LSTM autoencoder model was also defined using the lstm_autoencoder = sequential() method. Then model were compiled using the compile() function and the printed using the print() method and .summary() function. As seen from the Autoencoder summary of all the autoencoder models, that the CNN Autoencoder summary parameters which were trainable were 11, 011 out of a total of 11.011 parameters which means that all the parameters were successfully trained as 0 parameters were not untrainable. Similar to the CNN autoencoder model, the RNN autoencoder model has 70, 996 parameters and 0 were untrainable which means that the model successfully trained all the parameters. The LSTM autoencoder was also successful as it shows that there were 251, 476 trainable parameters and 251, 476 were trained by the model.

CNN Autoencoder Summary:

Model: "sequential_6"

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 28, 28, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_8 (Conv2D)	(None, 14, 14, 32)	9,248
up_sampling2d_2 (UpSampling2D)	(None, 28, 28, 32)	0
conv2d_9 (Conv2D)	(None, 28, 28, 3)	867

Total params: 11,011 (43.01 KB)
Trainable params: 11,011 (43.01 KB)
Non-trainable params: 0 (0.00 B)

RNN Autoencoder Summary:

Model: "sequential_7"

Layer (type)	Output Shape	Param #
simple_rnn_5 (SimpleRNN)	(None, 128)	27,264
repeat_vector_2 (RepeatVector)	(None, 28, 128)	0
simple_rnn_6 (SimpleRNN)	(None, 28, 128)	32,896
time_distributed_2 (TimeDistributed)	(None, 28, 84)	10,836

Total params: 70,996 (277.33 KB)

Trainable params: 70,996 (277.33 KB)

Non-trainable params: 0 (0.00 B)

LSTM Autoencoder Summary:

Model: "sequential_8"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 128)	109,056
repeat_vector_3 (RepeatVector)	(None, 28, 128)	0
lstm_1 (LSTM)	(None, 28, 128)	131,584
time_distributed_3 (TimeDistributed)	(None, 28, 84)	10,836

Total params: 251,476 (982.33 KB)

Trainable params: 251,476 (982.33 KB)

Non-trainable params: 0 (0.00 B)

It has four layers with descriptions of their type, shape of output and size of the parameters as follows: The first layer is LSTM (Long Short-Term Memory) layer with 128 neurons. This has 109,056 trainable parameters which include weight, bias, and recurrency used in processing sequential data. The second layer is a RepeatVector which indicates the input sequence, thus it has no trainable parameters. The next layer is another LSTM layer with 128 cells which gives 131,584 trainable parameters. This layer is employed in reconstructing the input in the decoder section of the autoencoder. The fourth and the last layer of the model is also TimeDistributed layer which consists of a Dense layer with 84 units for each time step in the input. This layer has 10836 trainable parameters out of which 3233 are learnable. There are 251,476 parameters in the model and all of the parameters are trainable, that is, these parameters will be adjusted during the training phase. There are no free parameters meaning that as in any neural network every parameter in this model is learnable. This model may have used time

series data as input-data with sequential input output structure for anomaly detection or reconstructions since it uses LSTMs.

Conclusion (With recommendations)

Unsupervised learning on the CIFAR-10 dataset can be improving by increasing the quality of the images by using using methods such as clustering and reducing the dimensions of the images themselves. This can be done by making use of autoencoders such as the RNN and CNN autonencoders. Autoencoders have the ability to learn representations of images that have been compressed. Autoencoders allow input images to be reconstructed while their key features are captured in a reduced space. They remove noise from images that are in the dataset which allows for data that can be easily interpreted and visualised. Clustering is also a method that can be used for example K-means clustering. Clusters allows for the identification of any outliers in the data and therefore, clustering is recommended for the CIFAR-10 dataset. K-means clustering was done for this report, and it is highly recommended. Clustering improves the agility and accuracy of the RNN, CNN AND LSTM autoencoder models. Visualising the images before doing anything with the data in the dataset is recommended as it allows for the confirmation of the kind of images are found in the CIFAR-10 dataset and to see original quality of the images, so it becomes easier to see if the RNN, CNN and LSTM autoencoders are working properly. Moreover, resizing the images and the normalisation of the data in the dataset is highly recommended. Resizing the images ensures that the CNN, RNN and LSTM autoencoders are done correctly without any error in terms of compatibility and it also increases computational accuracy and normalising the data allows the models to be faster and more efficient.

REFERENCES

Berg-Kirkpatrick, T., Bouchard-Côté, A., Denero, J., & Klein, D. (2010). *Painless Unsupervised Learning with Features* (pp. 582–590). Association for Computational Linguistics.

Dayan, P. (1999). *Unsupervised Learning*.

James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). Unsupervised Learning. *Springer Texts in Statistics*, 503–556. https://doi.org/10.1007/978-3-031-38747-0_12

Krizhevsky, A. (2009). *CIFAR-10 and CIFAR-100 datasets*. Toronto.edu. https://www.cs.toronto.edu/~kriz/cifar.html

Tyagi, K., Rane, C., Sriram, R., & Manry, M. (2022, January 1). *Chapter 3 - Unsupervised learning* (R. Pandey, S. K. Khatri, N. kumar Singh, & P. Verma, Eds.). ScienceDirect; Academic Press. https://www.sciencedirect.com/science/article/abs/pii/B9780128240540000125

Valkenborg, D., Rousseau, A.-J., Geubbelmans, M., & Burzykowski, T. (2023). Unsupervised learning. *American Journal of Orthodontics and Dentofacial Orthopedics*, *163*(6), 877–882. https://doi.org/10.1016/j.ajodo.2023.04.001