Rosa Salih, 920250

**Introduction**

We are provided with the CIFAR-10 image dataset, which contains 1000 training images and 100 testing images, all of the same size. The problem with this dataset is that none of the images are sorted into categories. There are several algorithms that can be used to train the machine to find similarities in these images’ features, and sort them into 10 categories based on these findings. The strength of the models created by these algorithms is based on how high the accuracy is when sorting.

My proposed solution is by using a neural network, since it gave a higher accuracy than the other algorithms. By storing all the predictions from the other models, the machine was able to decide a final output of 51.6% accuracy using a neural network. The SVM and LDA models closely followed, with SVM being accurate 50.3% of the time and LDA 49.4%. K-Means and GMM weren’t very accurate in comparison, with the maximum accuracy being 11.1% between them, even when using the PCA transformed data.

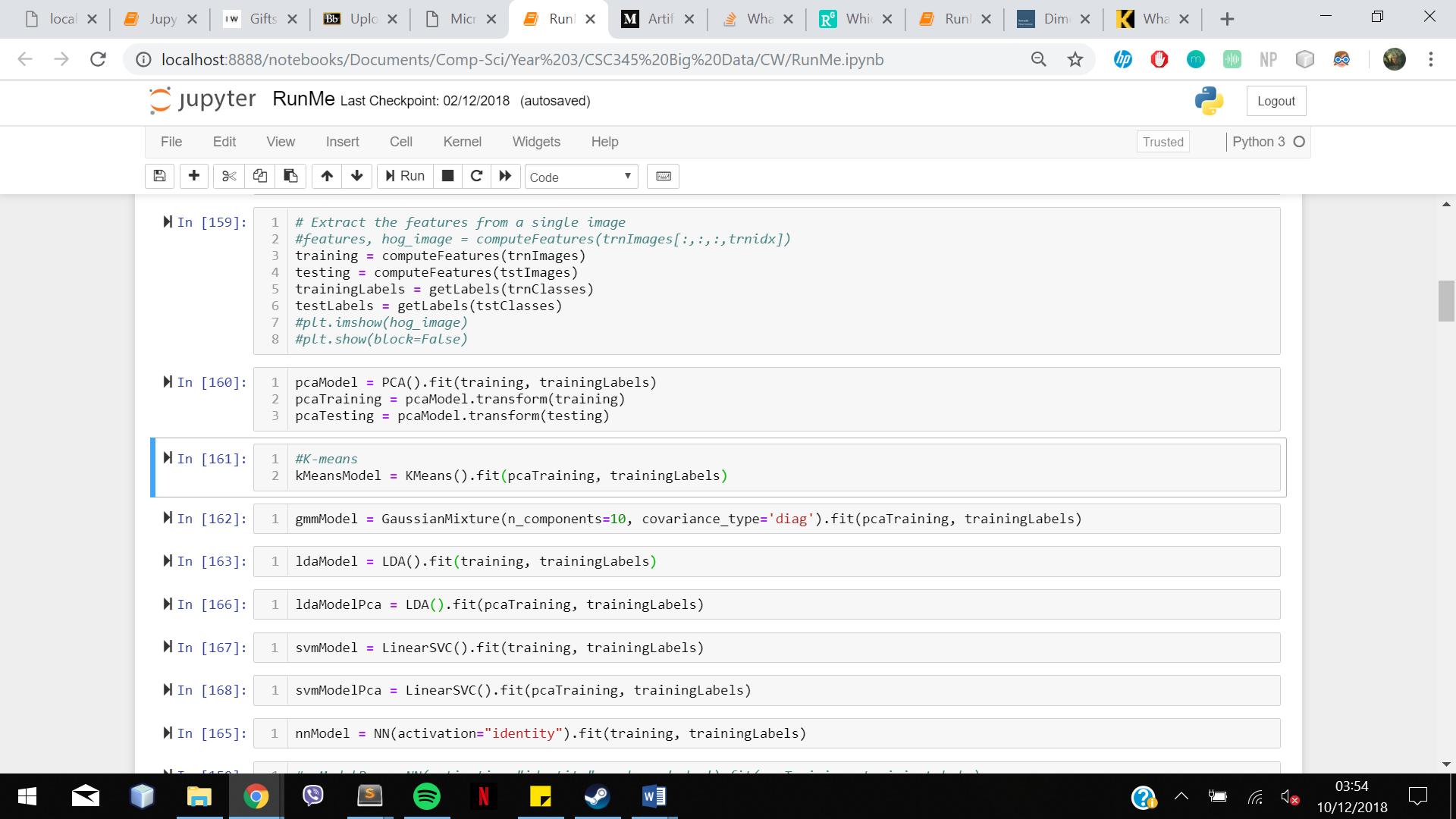
**Method**

1. I loaded the training and testing image data, and their labels, using Numpy.
2. Within the computeFeatures method, I added a for loop that goes through the whole image dataset and creates a list of their features.
3. I then passed the training and testing images through computeFeatures to calculate the features in each one. I also passed the training and testing labels through a method called getLabels to retrieve them.
4. I transformed the training set using PCA and made models using K-Means, GMM, LDA, SVM and NN by passing the transformed set as a parameter, with their labels.
5. I also made these models by passing the original training data, in order to compare accuracies with the models that were transformed.
6. To produce a confusion matrix, I evaluated the predicted classes against the actual classes.

There are 10 classifications: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. When training the data, the rows from the training data correspond to each of the 10 rows of classifiers. Therefore, if the model is trained properly, once the testing data is passed through, it should correctly guess which classification each image comes under.

The features are extracted via the for loop in computeFeatures. This method goes through the whole image data set and computes the hog features, appending them to a list as it runs. These features then need to be categorised into the classifications. I did this by filtering out the features that have the greatest variance, as these are more likely to correspond to a smaller group of classes. I used K-Means to cluster the data into distinct groups; using PCA on this was useful as it finds the features that show as much variation across the dataset as possible. I also used PCA for GMM to generate histograms - I used the ‘diagonal’ option to constrain the covariance of the estimated classes. PCA allowed the models to reduce dimensionality, removing redundancy of specific features, making it easier for the images to be classified.

I created models for LDA, SVM and NN too. At first, I passed through the PCA transformed data set, however when creating these models using the original training images, the accuracy was much higher. Therefore, I used built-in sklearn methods to create the models, by passing in both the training images dataset and the training labels dataset. This is shown in the image below. Each model transforms and finds an optimal boundary between all possible outputs, so when put together, all these algorithms enable the neural network to achieve a higher accuracy. The learned patterns are applied to the testing images in order to predict which category each image falls under, based on how each feature has been classified.



I visualised my findings in a confusion matrix to show exactly how many times each model gets the testing images in the right category.

**Results**

To evaluate my results, I created a confusion matrix for each model and used this to calculate classification accuracy. The confusion matrix compares the predicted categories (Y axis) to the actual categories (X axis). For example, in figure 1, the model predicts the image is an airplane correctly 4% of the time. The classification accuracy is then calculated

For K-Means and GMM, the results were more accurate when the dataset was transformed using PCA, however for the other models, the predictions were more accurate when the original training images were used. An example of this is shown in figure 1. Because of this, I kept these two models transformed when creating my neural network.

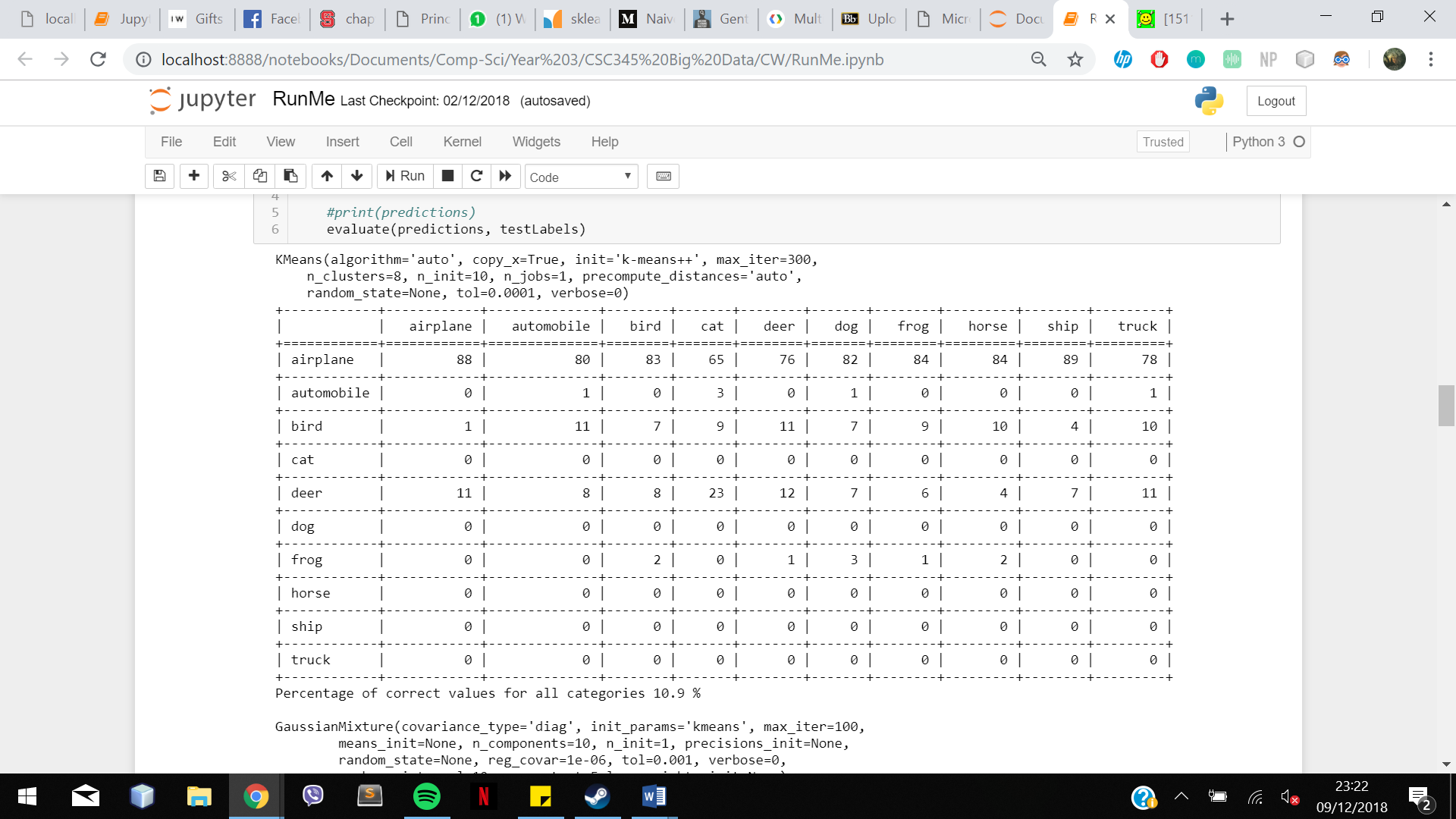
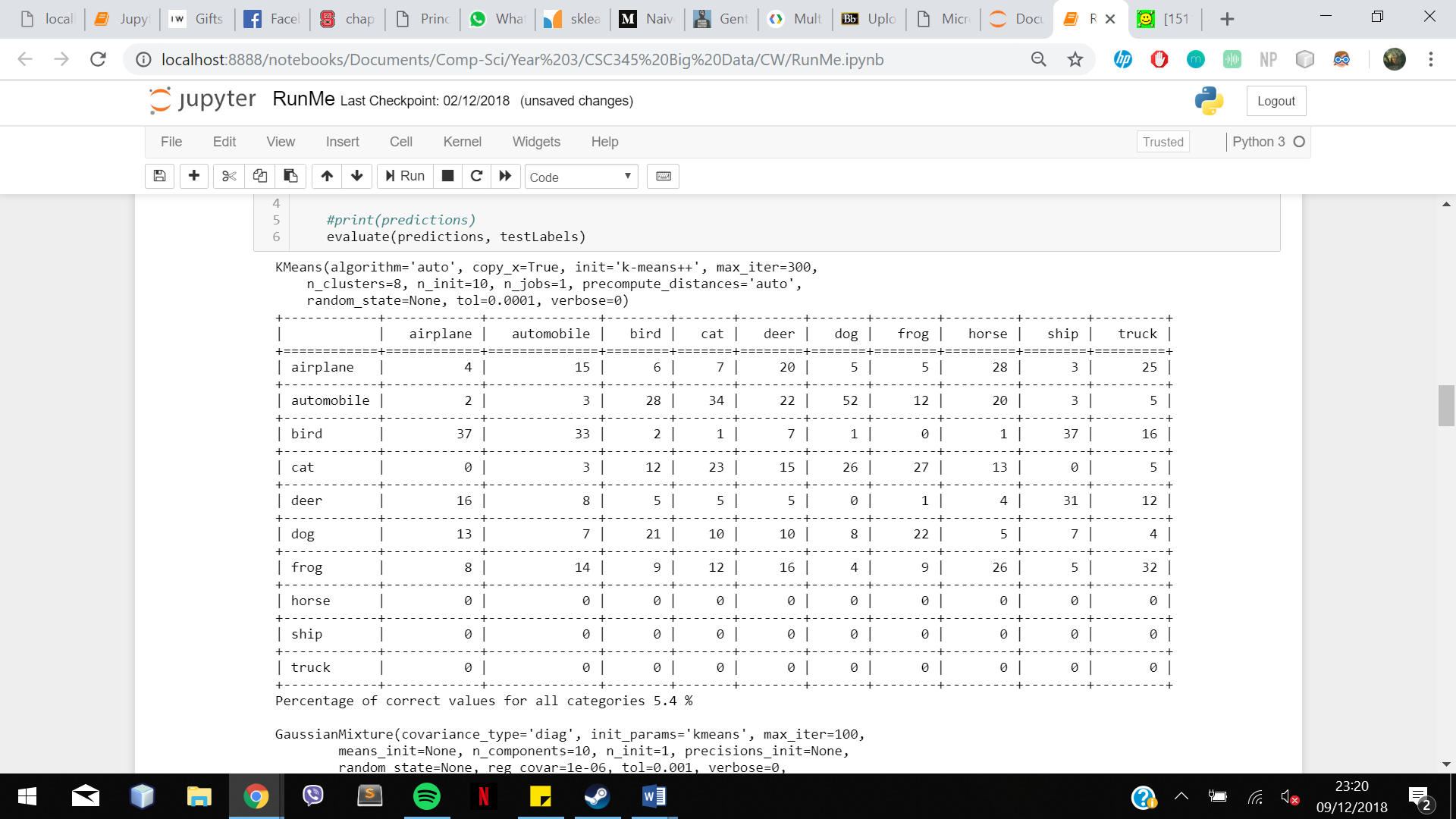


Figure 1: K-Means with PCA compared to K-Means without PCA

The final confusion matrix of my proposed method is shown in figure 2. The average accuracy across all 10 categories was 51.6%, which is higher than the benchmark of 44.68%, making it a fairly accurate method. It most accurately identified the trucks (63%), ships (62%), airplanes (61%) and horses (60%). However, the model failed to produce as many accurate results with the other classifiers, with it least accurately identifying the birds (31%) and cats (33%). This is makes sense since the other models also failed to accurately identify these categories as well.

I also tried creating another neural network by passing through the dataset that had been transformed by PCA, and testing them on the transformed test images, but these results weren’t as accurate. This could be due to convergence failure due to its low dimensionality[1]. Figure 3 shows this confusion matrix. By transforming the data, I was able to get more accurate results for birds (43%) but not anything else. I tried normalising the data before using PCA, but somehow this made the accuracy lower. On the other hand, by creating models both with and without PCA for LDA and NN, but comparing it to the original testing images, I was able to get slightly higher accuracies for all the models.

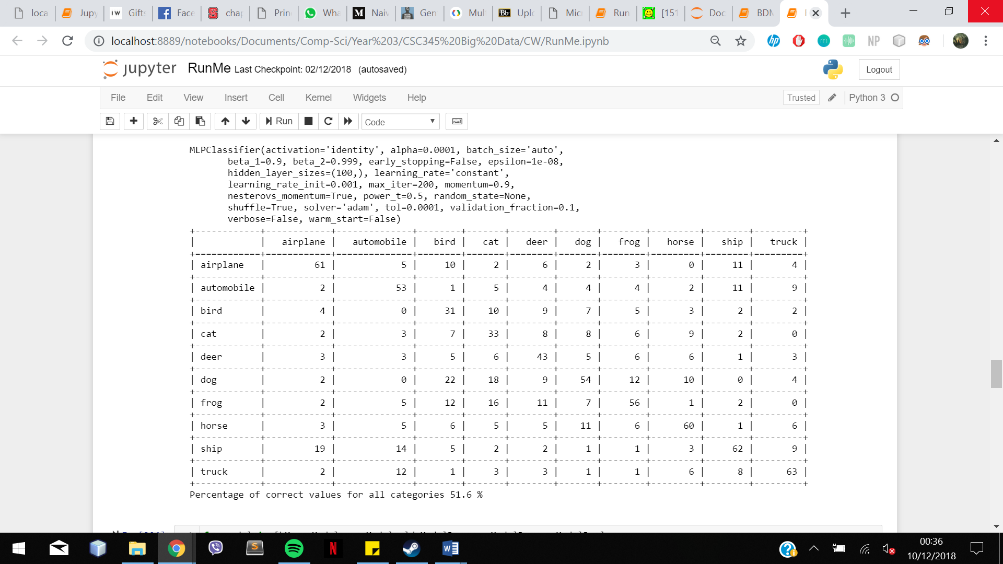


Figure 2: Neural Network Confusion Matrix with no dimensionality reduction

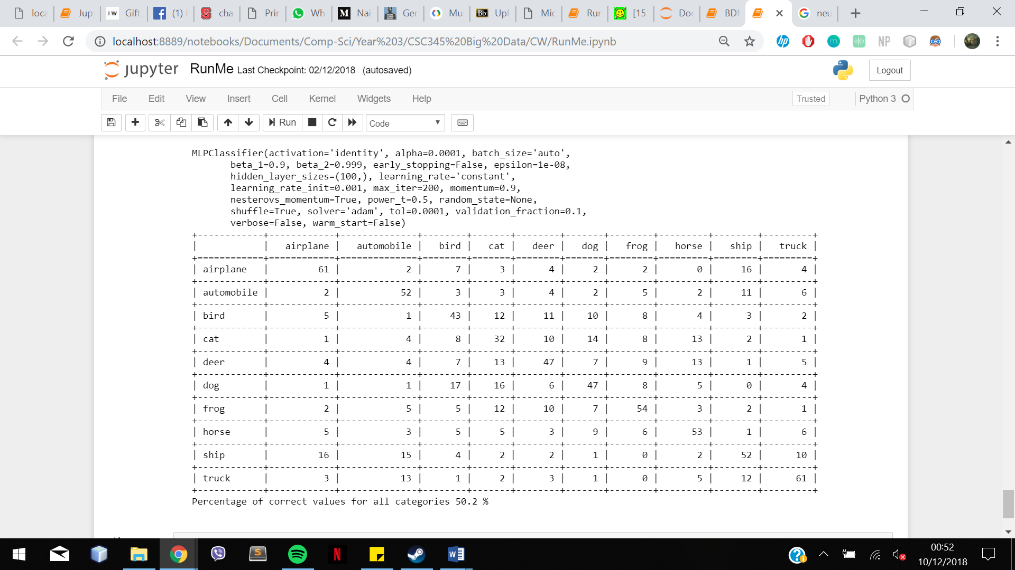


Figure 3: Neural Network Confusion Matrix using PCA

If we were to compare these results to the results of SVM, since it was the second most accurate method, we can see that although the overall accuracy is lower, this model was able to identify the birds (34%), deer (50%) and trucks (64%) more accurately to the neural network. Therefore, a higher accuracy could be achieved if both SVM and NN are used together.

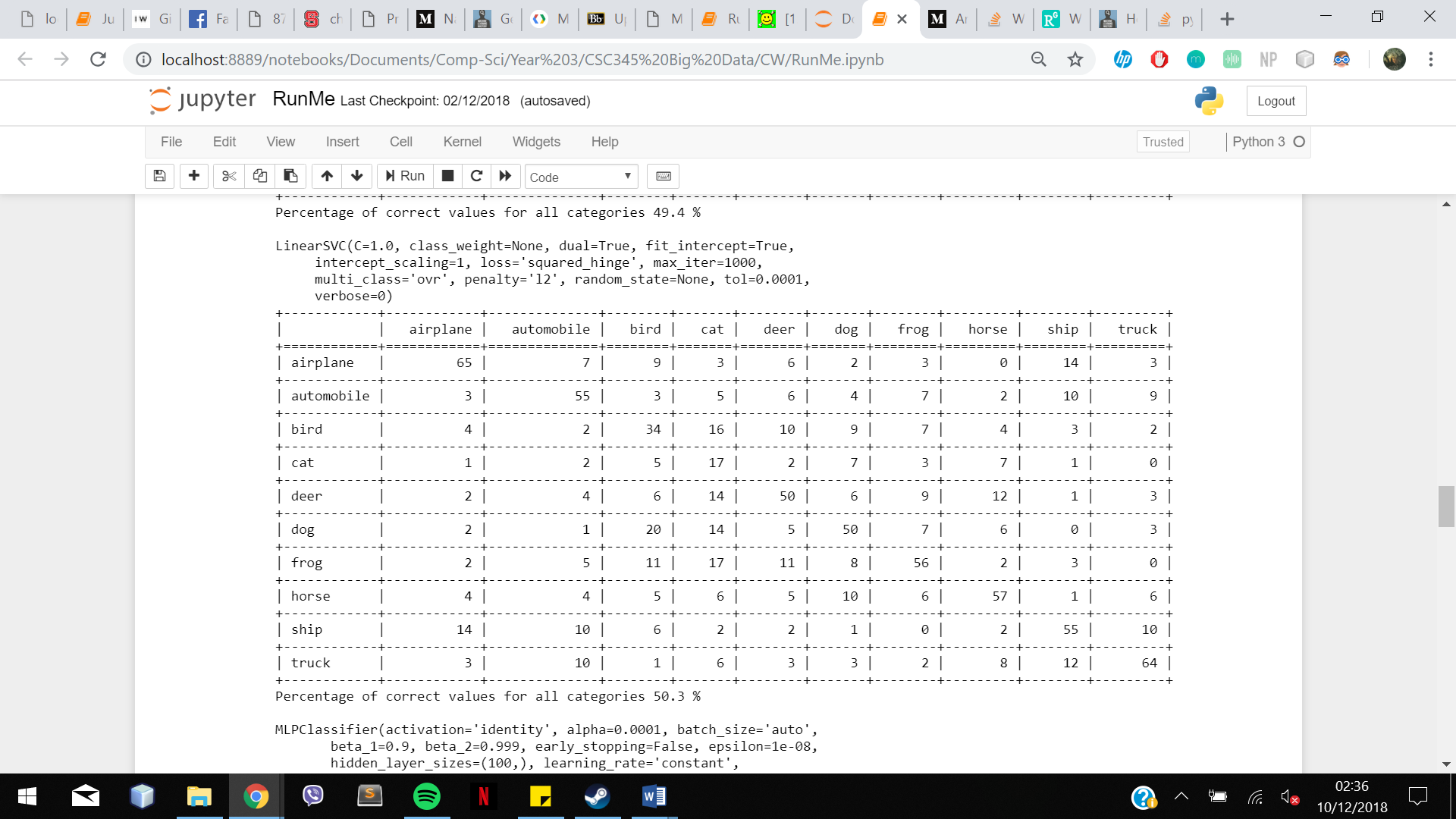


Figure 4: SVM Confusion Matrix

**Conclusion**

In conclusion, my method of categorising these images is by making several different models using the training data and using this to predict classification of the testing images. By making these models, the neural network was able to decide on which category each image comes under. The problem I had with my model is that certain categories were not as accurately recognised as others, due to the models not being able to distinguish certain features as well as they should.

Since this dataset is quite small, its accuracy could be further improved if more images are tested on, as more features can be compared. In addition, Exponential Linear Units[2] can be used to get higher classification accuracies since it speeds ups learning in neural networks. They have negative values that allow them to push mean unit activations closer to zero, normalising the data, but using lower computational complexity. This reduces bias, increasing overall accuracy. Using ELUs has been done previously and has actually yielded the best published results on CIFAR-100.

**References**

[1] <https://d4datascience.wordpress.com/2016/09/29/fbf/>

[2] <https://arxiv.org/abs/1511.07289>

[3] https://medium.com/@Killavus/naive-approaches-to-solving-cifar10-dataset-image-classification-problem-overview-and-results-636048ff1cc1