21COC102 - Advanced Artificial Intelligent Systems

Student ID: B929107

Part A

Here I will be going over the tools and the methods I used, including sources, with a comprehensive description of the changes I made with respect to code available online

Tools

I decided to work using the CIFAR-10 dataset[1], which is a set of 50,000 32x32x3 (32x32 pixels with 3 channels RGB values)training images used to train the network with 10,000 testing images used to evaluate the networks performance. Later in the report I explain how I came to the decision of using the CIFAR dataset. The libraries I used for this project were Numpy[2], pandas[3], Scikit-learn[4], TensorFlow[5], keras[6], and matplotlib[7].

Method

When I first started this project, I was aware of the methods of Artificial Neural Network training, both from the content covered already in this module and the modules from previous years. However, I was clueless as how to implement the sorts of Neural Networks used to classify images or anything else.
I started by searching "How to build a classification model in Python" on youtube, and clicked on this video[8] by Data Professor. This video used the iris dataset and built a decision tree using the RandomForest algorithm. It was a very accessible video and, using this video and his jupyter notebook code[9], I taught myself the basics of how you could build a program that trains using a training data set and how to test the performance of it using a testing data set. As I didn't understand what the RandomForest algorithm was, I looked up a video that explained it in more detail[10]. Watching the RandomForest video allowed me to understand how it works on a majority vote of randomly created decision trees. Also, I was showed how RandomForest can be applied to more complex datasets such as the MNIST dataset, as well as how to use useful evaluation tools such as confusion matrices, and how to make tables that are more legible than raw data, by using pandas DataFrame[3].
After watching this video, I began to understand that what I had previously thought of as an acceptable dataset, the iris dataset, was actually too simple. Therefore, I started to look at the datasets that I could use, including the two recommended ones in the coursework specification as well as recommended datasets online for learning machine learning. In the end I decided on the CIFAR-10 dataset as I found the idea of it exciting and it was free and easy to install and use. Next, I followed a video by Binary Study[11] and read the online CIFAR-10 documentation[1] to learn how to "unpickle" the CIFAR-10 data once downloaded, as well as how to use matplotlib[7] to display the images the images from the dataset. I created an algorithm for combining the 5 separate training batch files into one so that they could all be accessed and used without having to be referred to separately. I did this using a tutorial video[12] on using glob[13] to import all the files from a folder.
I soon ran into some difficulties trying to fit the RandomForest algorithm to the CIFAR-10 dataset, and I realised it was not going to work for this dataset. Here, I looked up a video that explained how to do image classification using CNNs[14] and his previous video explaining how CNNs work[15]. Here I began to truly understand how the whole process works and, using the code from the video[16] I began to be able to implement the different learning architectures and models and play with them and understand how they work on a more technical level, and specifically in the python environment.

Models

The first method that I used was a dense artificial neural network that takes the 32x32x3 image as input, flattens it, then goes to 3000 neurons that use the ReLu activation function, then in the next layer 1000 ReLu neurons and then 10 softmax neurons for the 10 possible ouput categories (softmax is used instead of sigmoid as it normalises the output). After evaluating the network on training data, I got a loss of 1.46 and an accuracy of 47%, which is not effective at all. After getting these results, I decided to run the network on a smaller number of neurons in each layer and a higher number of epochs (as five could be considered a small number of epochs). Therefore, I decided to run the ANN training algorithm again with 1500 ReLu neurons in the first layer and 800 ReLu neurons in the second layer, and also I decided to use the sigmoid function for the last layer to experiment and see if it would improve results. The second algorithm took 300s, whereas the first took 273s so it needs to balance the increased fitting time with improved results. On the final epoch of the training data, it seemed to be doing so with an accuracy of 54%, but after evaluating it on the test data, this value dropped to 48%, only 1% better than the previous algorithm, with loss of 1.45. Therefore, the traditional dense neural network approach did not seem to work well for this type of dataset: the training was relatively slow, the accuracy percentages were low, and the results were quickly diminishing with higher numbers of epochs. Therefore, I needed to use a more effective training model to get higher accuracy and lower loss.
Convolutional Neural Networks work by identifying patterns to look for that distinguish different classes from each other, and then scan for those patterns when trying to identify a given input. This sounds like something that would be very effective for CIFAR-10 as this is essentially how humans learn to classify different classes of objects in everyday life.
The model of convolutional training that I used first was one that is known to work well, where there is a convolution layer with ReLu activation function to identify primary features, then pooling is done to make the image smaller and easier to process. Then, it goes through a second layer of convolution with ReLu to identify secondary features, and a second layer of pooling to reduce the image even more and make it easier to categorise. Finally it goes through a dense artificial network to give probability outputs for each of the possible 10 classes.
The first CNN model I used 32 filters of size 3x3 in the first layer, then 64 of the same size in the second convolution layer, with max pooling (taking the largest value from a set size square and mapping it to a single pixel) of size 2x2 after each. Then finally a layer of ReLu with 64 neurons and a layer with 10 softmax to give final outputs. This model executed in 253s (faster than any other model before) and had an accuracy on the test data of 70%! (much higher than any other model before) and a loss of only 0.94. However, although this was by far the best so far, I believed that my algorithm could do better. I trained the algorithm with more samples per layer (64 in the first, 96 in the next), and I did 12 epochs instead of 10, as I know there are diminishing returns with more epochs, but I wanted to see if there was more performance I could squeeze out of the algorithm, without overfitting, by running through the data a couple more times. After running the algorithm and evaluating the accuracy and loss. This 'improved' algorithm was negligibly different from the first algorithm despite taking significantly longer to train. Reflecting on this, I then thought this meant that I had to be more clever in the way I train the network to improve performance, and I had two ideas: implementing early stopping and cross validation to help reduce overfitting. I followed a tutorial online[17] that demonstrated how to apply a cross validation to a neural network in python, and I adapted his code to fit mine, as well as implementing a 'callback' from looking at the keras documentation. I then ran the code and evaluated it to find out to my surprise that: The algorithm had done very few epochs before meeting the 'min_delta' requirement I set, and that the resulting accuracy wasn't very different, and possibly worse, which was disappointing.

Part B

This section goes over the effectiveness of the Neural Network models used, and evaluates them using the tools mentioned earlier in Part A. My apologies that this section is longer than 3 pages, but it is filled with a lot of graphs and tables which take up a lot of space, and is actually quite sparse in text.

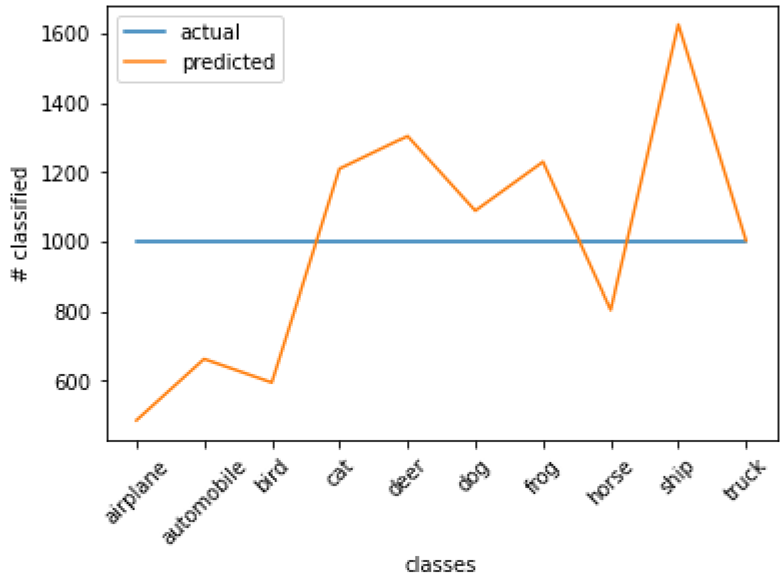
Metrics

The metrics that I decided to use were: A table with total values guessed vs actual values for each class, the table's values plotted into a graph, an sklearn "classification report" which contains precision (True positives/ True positives + False positives), recall (True positives/ True positives + False negatives), and f1 score (2*precision * recall/ (1/precision) + (1/recall)). Also, I show the whole confusion matrix using dataframe for each network.

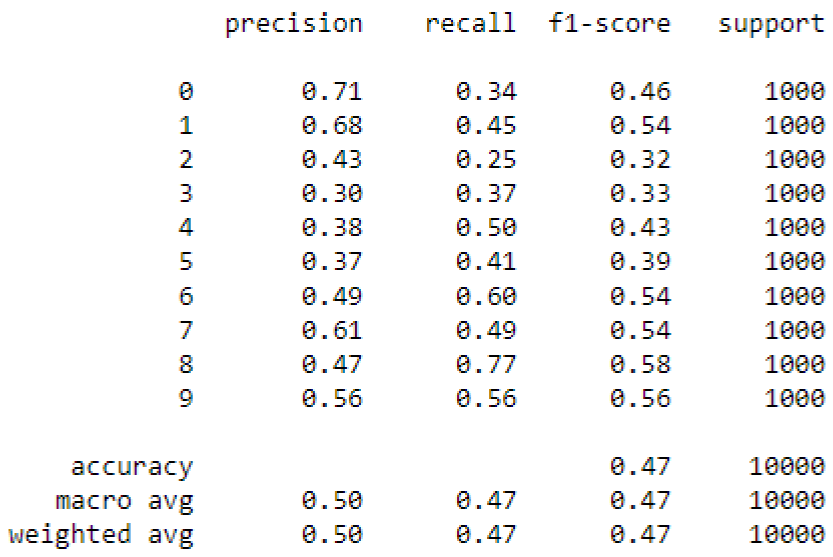
ANN 1

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
Test Data	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
Predicted Data	485	662	594	1209	1303	1088	1229	803	1625	1002

Above is shown the table for the first ANN. As you can see the network performed pretty poorly most of the classes are massively over-classified or under-classified. The classes that it seemed to classify well were truck and dog, but this will be shown more clearly in the confusion matrix later.

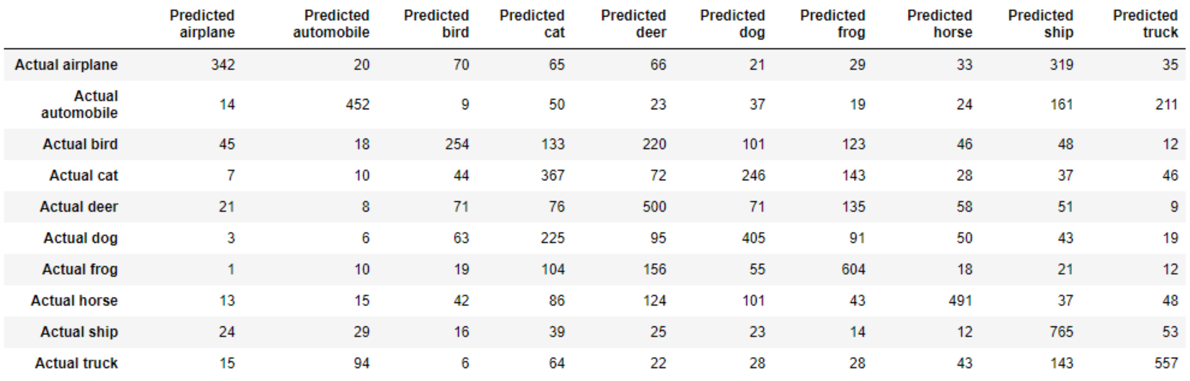


Here you can see the same information more clearly plotted in a graph format, and you can really see the large variance in classification values, ranging from 600 to 1600. This visually shows how poorly this network performed.



```
              precision    recall  f1-score   support

           0       0.71      0.34      0.46      1000
           1       0.68      0.45      0.54      1000
           2       0.43      0.25      0.32      1000
           3       0.30      0.37      0.33      1000
           4       0.38      0.50      0.43      1000
           5       0.37      0.41      0.39      1000
           6       0.49      0.60      0.54      1000
           7       0.61      0.49      0.54      1000
           8       0.47      0.77      0.58      1000
           9       0.56      0.56      0.56      1000

    accuracy                           0.47     10000
   macro avg       0.50      0.47      0.47     10000
weighted avg       0.50      0.47      0.47     10000
```

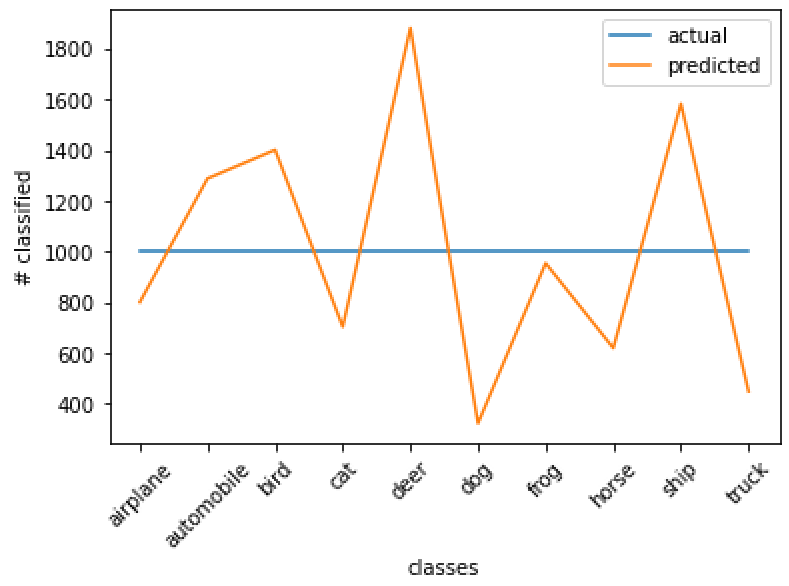
Here is the classification report for ann1: it had a precision of 0.5, recall of 0.47 and f1-score of 0.47, which shows that it was not very effective at classifying the dataset.



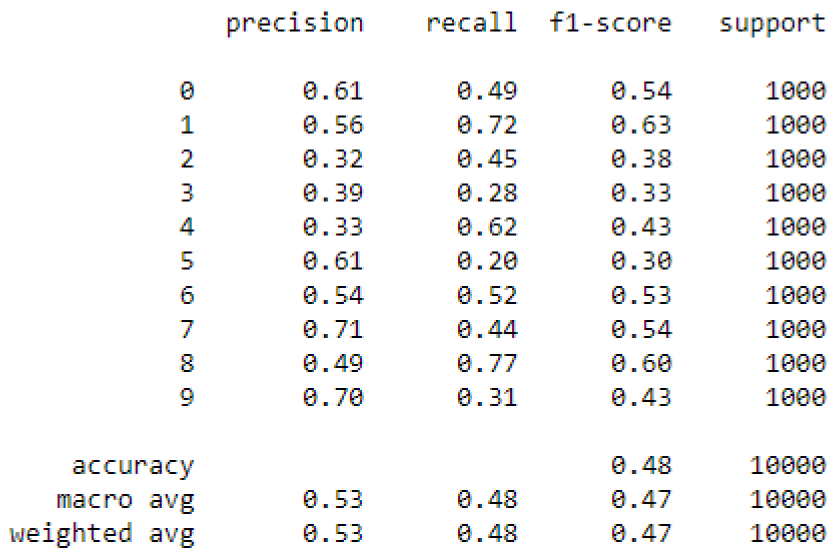
	Predicted airplane	Predicted automobile	Predicted bird	Predicted cat	Predicted deer	Predicted dog	Predicted frog	Predicted horse	Predicted ship	Predicted truck
Actual airplane	342	20	70	65	66	21	29	33	319	35
Actual automobile	14	452	9	50	23	37	19	24	161	211
Actual bird	45	18	254	133	220	101	123	46	48	12
Actual cat	7	10	44	367	72	246	143	28	37	46
Actual deer	21	8	71	76	500	71	135	58	51	9
Actual dog	3	6	63	225	95	405	91	50	43	19
Actual frog	1	10	19	104	156	55	604	18	21	12
Actual horse	13	15	42	86	124	101	43	491	37	48
Actual ship	24	29	16	39	25	23	14	12	765	53
Actual truck	15	94	6	64	22	28	28	43	143	557

Here you can see the 1st ANN's values for each possible outcome

ANN 2



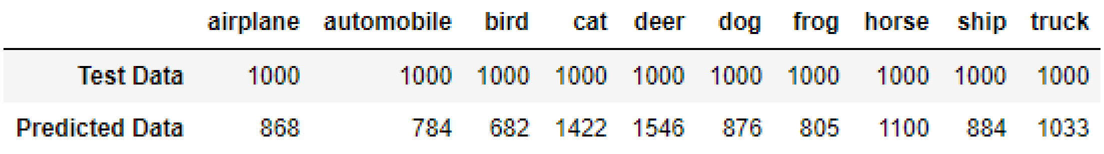
Here you can see that the outcome here was very different to the last, showing that the outcomes of these neural networks on this dataset is very random and varied.



```
              precision    recall  f1-score   support

           0       0.61      0.49      0.54      1000
           1       0.56      0.72      0.63      1000
           2       0.32      0.45      0.38      1000
           3       0.39      0.28      0.33      1000
           4       0.33      0.62      0.43      1000
           5       0.61      0.20      0.30      1000
           6       0.54      0.52      0.53      1000
           7       0.71      0.44      0.54      1000
           8       0.49      0.77      0.60      1000
           9       0.70      0.31      0.43      1000

    accuracy                           0.48     10000
   macro avg       0.53      0.48      0.47     10000
weighted avg       0.53      0.48      0.47     10000
```

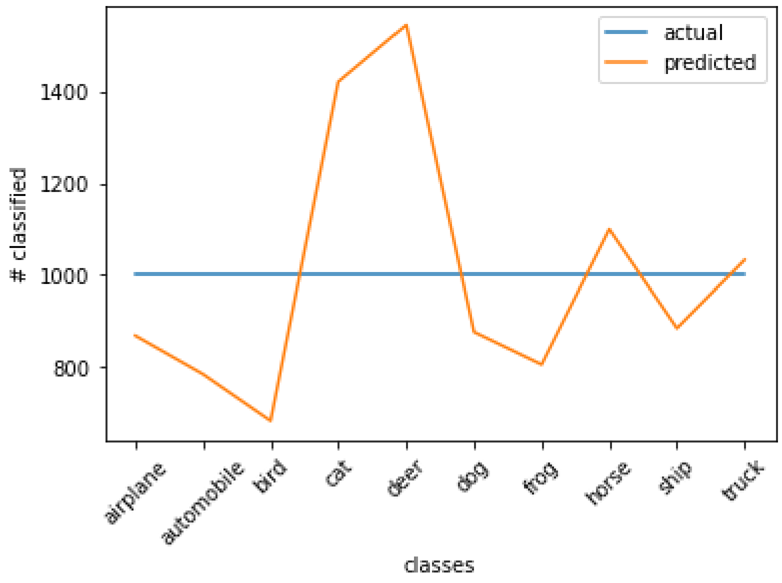
Here is the classification report for ANN2: it had a precision of 0.53 (slightly better), recall of 0.48 (slightly better) but an f1-score of 0.47, which shows that it is much the same as the last algorithm, with very little improvement for the extra time spent.

CNN 1



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
Test Data	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
Predicted Data	868	784	682	1422	1546	876	805	1100	884	1033

Above is shown the table for the first CNN. This time the network performed more consistently, and is less varied as will be shown by the graph below.



Here you can see that although the graph looks very similar visually, the y axis has a much smaller scale, showing that the algorithm is more consistent. However, the CNN did massively overclassify the cat and deer classes.

```
              precision    recall  f1-score   support

           0       0.72      0.70      0.71      1000
           1       0.76      0.81      0.79      1000
           2       0.58      0.56      0.57      1000
           3       0.47      0.48      0.47      1000
           4       0.63      0.59      0.61      1000
           5       0.54      0.63      0.58      1000
           6       0.73      0.78      0.75      1000
           7       0.70      0.70      0.70      1000
           8       0.81      0.79      0.80      1000
           9       0.81      0.69      0.75      1000

    accuracy                           0.67     10000
   macro avg       0.67      0.67      0.67     10000
weighted avg       0.67      0.67      0.67     10000
```

Here is the classification report for CNN1: it had a precision, recall and f1-score of 0.67, which shows that it is much more consistent and effective at classifying the dataset than the last two algorithms.

	Predicted airplane	Predicted automobile	Predicted bird	Predicted cat	Predicted deer	Predicted dog	Predicted frog	Predicted horse	Predicted ship	Predicted truck
Actual airplane	673	9	45	54	59	18	9	24	64	45
Actual automobile	24	698	12	35	15	13	10	22	35	136
Actual bird	49	3	452	127	212	51	41	46	8	11
Actual cat	7	5	34	593	125	145	30	47	6	8
Actual deer	9	1	20	76	779	25	19	65	3	3
Actual dog	9	0	31	269	83	524	14	66	4	0
Actual frog	3	2	45	109	124	31	661	15	3	7
Actual horse	7	0	19	57	100	41	4	767	1	4
Actual ship	67	20	11	58	31	13	11	11	739	39
Actual truck	20	46	13	44	18	15	6	37	21	780

CNN 2

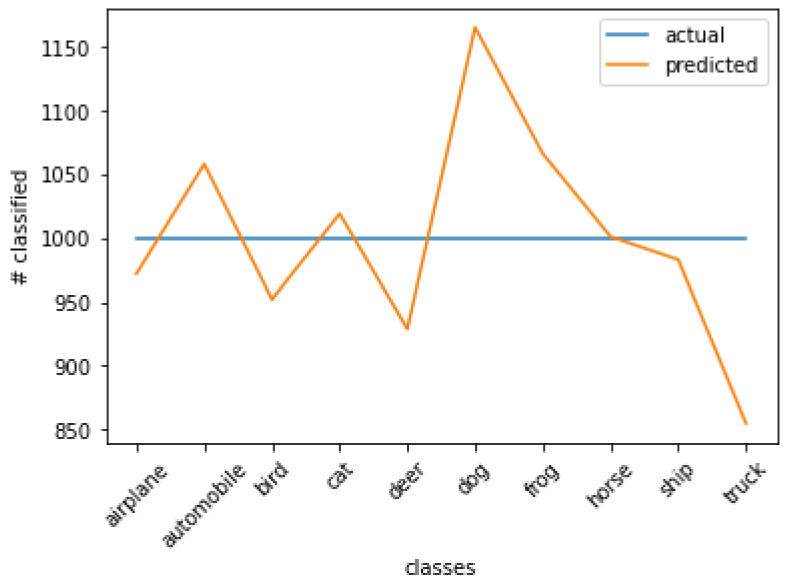
```
              precision    recall  f1-score   support

           0       0.79      0.71      0.75      1000
           1       0.90      0.70      0.79      1000
           2       0.63      0.58      0.60      1000
           3       0.49      0.52      0.50      1000
           4       0.60      0.68      0.64      1000
           5       0.51      0.69      0.58      1000
           6       0.78      0.75      0.76      1000
           7       0.80      0.68      0.74      1000
           8       0.81      0.83      0.82      1000
           9       0.78      0.80      0.79      1000

    accuracy                           0.69     10000
   macro avg       0.71      0.69      0.70     10000
weighted avg       0.71      0.69      0.70     10000
```

Here is the classification report for CNN2: it had a precision, recall and f1-score of 0.70, which shows improvement from the last algorithm, but only by a small amount that could change from execution to execution, and I would say is overall not worth the extra execution time.

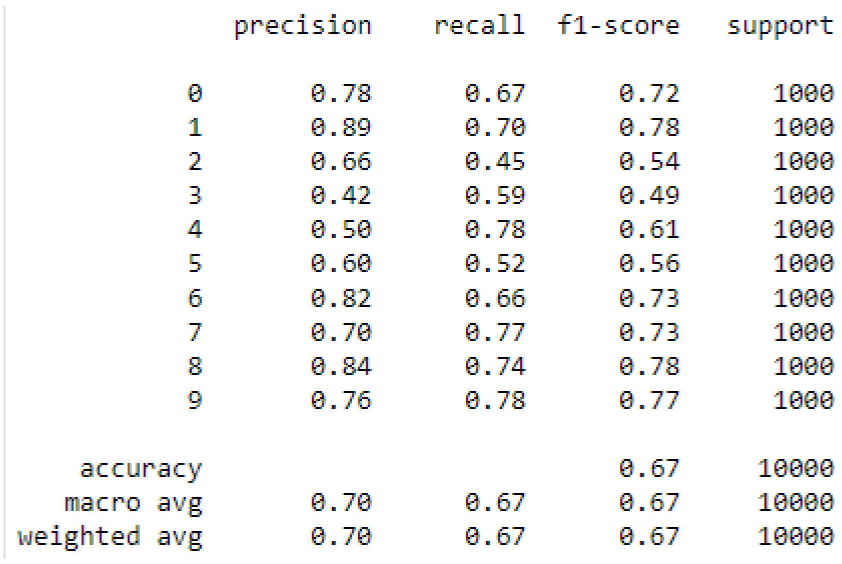
CNN 3

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
Test Data	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
Predicted Data	972	1058	952	1019	929	1165	1066	1001	983	855

Above is shown the table for the third CNN. The values here seem show a great deal of consistency, which is obviously good for a neural network.

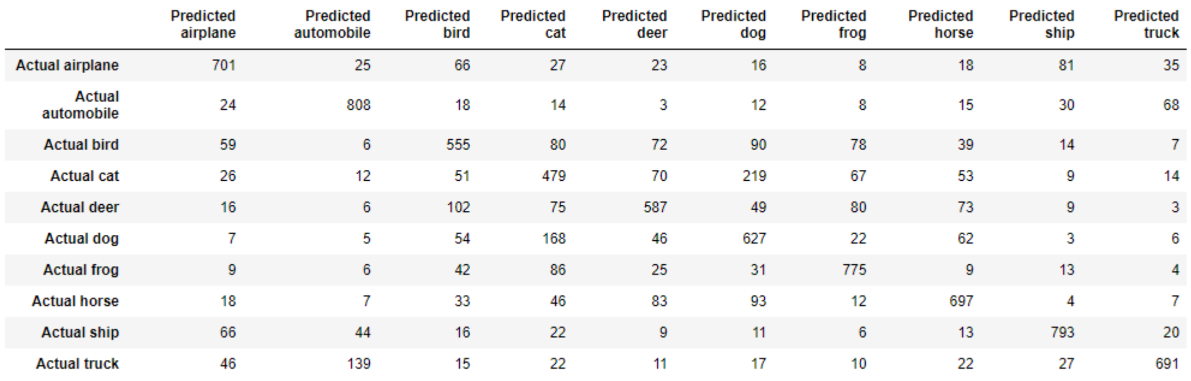


Here again you can see how the numbers of the classes classified has a much smaller range than before, and all seem to sit relatively close to 1000.



```
              precision    recall  f1-score   support

           0       0.78      0.67      0.72      1000
           1       0.89      0.70      0.78      1000
           2       0.66      0.45      0.54      1000
           3       0.42      0.59      0.49      1000
           4       0.50      0.78      0.61      1000
           5       0.60      0.52      0.56      1000
           6       0.82      0.66      0.73      1000
           7       0.70      0.77      0.73      1000
           8       0.84      0.74      0.78      1000
           9       0.76      0.78      0.77      1000

    accuracy                           0.67     10000
   macro avg       0.70      0.67      0.67     10000
weighted avg       0.70      0.67      0.67     10000
```

Here is the classification report for CNN3: it is disappointing to see that although the algorithm seems more effective at first, this hasn't actually made it more effective as the precision recall and f1-scores all are very similar as before.



	Predicted airplane	Predicted automobile	Predicted bird	Predicted cat	Predicted deer	Predicted dog	Predicted frog	Predicted horse	Predicted ship	Predicted truck
Actual airplane	701	25	66	27	23	16	8	18	81	35
Actual automobile	24	808	18	14	3	12	8	15	30	68
Actual bird	59	6	555	80	72	90	78	39	14	7
Actual cat	26	12	51	479	70	219	67	53	9	14
Actual deer	16	6	102	75	587	49	80	73	9	3
Actual dog	7	5	54	168	46	627	22	62	3	6
Actual frog	9	6	42	86	25	31	775	9	13	4
Actual horse	18	7	33	46	83	93	12	697	4	7
Actual ship	66	44	16	22	9	11	6	13	793	20
Actual truck	46	139	15	22	11	17	10	22	27	691

Here you can see the confusion matrix, and it is encouraging to see that a lot of classes predicted very few times when they shouldn't be, with numbers such as 4 (actual horse, predicted ship) and 3 (actual automobile, predicted deer), but this is balanced out by places where very large numbers are incorrectly categorised (139 actual truck, predicted automobile etc).

References

[1] Krizhevsky's, A., n.d. CIFAR-10 and CIFAR-100 datasets. [Online]
Available at: https://www.cs.toronto.edu/ kriz/cifar.html

[2] Oliphant, T., n.d. Numpy Webpage. [Online]
Available at: https://numpy.org/

[3] Many Contributors, n.d. pandas - Python Data Analysis Library. [Online]
Available at: https://pandas.pydata.org/

[4] Many Authors, n.d. scikit-learn: machine learning in Python. [Online]
Available at: https://scikit-learn.org/stable/index.html

[5] Many Contributors, n.d. TensorFlow Webpage. [Online]
Available at: https://www.tensorflow.org/

[6] Many Contributors, n.d. Keras: the Python deep learnin API. [Online]
Available at: https://keras.io/

[7] Hunter, J. D., 2007. Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), pp. 90–95.

[8] Data Professor, 2020. Machine Learning in Python: Building a Classification Model, Online video: YouTube.

[9] Data Professor, 2020. [Online]
Available at: https://github.com/dataprofessor/code/tree/master/python/iris

[10] codebasics, 2018. Machine Learning Tutorial Python - 11 Random Forest, Online video: YouTube.

[11] Binary Study, 2021. How to load and visualize CIFAR-10 and CIFAR-100 datasets, Online video: YouTube.

[12] The Trader's Code, 2021. In python how to import access multiple files from a folder, Online Video: YouTube.

[13] Many Authors, n.d. glob — Unix style pathname pattern expansion. [Online]
Available at: https://github.com/python/cpython/blob/3.10/Lib/glob.py

[14] codebasics, 2020. Image classification using CNN (CIFAR10 dataset) — Deep Learning Tutorial 24 (Tensorflow Python), Online video: YouTube.

[15] codebasics, 2020. Simple explanation of convolutional neural network — Deep Learning Tutorial 23 (Tensorflow Python), Online video: YouTube.

[16] codebasics, 2020. Small Image Classification Using Convolutional Neural Network (CNN). [Online]
Available at: https://github.com/codebasics/deep-learning-keras-tf-tutorial/blob/master/16_cnn_cifar10_small_image_classification/cnn_cifar10_dataset.ipynb
Accessed 2022
.

[17] Hussain, F., 2020. Tutorial 14: K-Fold Cross Validation using Keras Python — K-Fold Cross-Validation in Neural Network. s.l.,Online video: YouTube.