

**CE4042**

**Neural Network & Deep Learning**

**Assignment 2 Report**

**Name: Neo Shun Xian Nicholas**

**Matric Number: U1820539F**

**Date: 11 November 2020**

Table of Contents

**Miscellaneous1**

**PART A: Object Recognition1**

Introduction2

Question 12

Question 22

Question 32

Question 42

Overall Conclusion2

**PART B: Text Classification4**

Introduction5

Question 12

Question 22

Question 32

Question 45

Question 52

Question 62

Overall Conclusion2

**Miscellaneous**

**Libraries/modules used in this assignment**

* numpy
* matplotlib
* imageio
* pickle
* tensorflow 2/keras
* csv
* re
* time
* pylab

**Coded in**

* Python
* Google Colaboratory (GPU instance)

**Files included in this project**

* Neo\_Shun\_Xian\_Nicholas\_A2\_report.pdf
* Neo\_Shun\_Xian\_Nicholas\_A2\_codes.zip
  + Part\_A.ipynb
  + Part\_B1.ipynb
  + Part\_B2.ipynb
  + Part\_B3.ipynb
  + Part\_B4.ipynb
  + data\_batch\_1
  + test\_batch\_trim
  + train\_medium.csv
  + test\_medium.csv

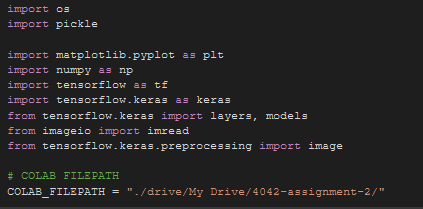
**PART A: Object Recognition**

**Introduction**

In this section, we are tasked to predict the correct class of the test dataset given the labelled training dataset. The dataset used in this section is the CIFAR-10 dataset. The dataset contains RGB colour images of size 32x32 and their corresponding label from 0 to 9.

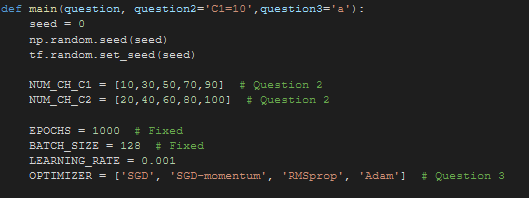
Import relevant libraries/modules

To execute and complete the analysis, these python libraries/modules are imported. Google Colaboratory (GPU instance) is used to run the analysis.



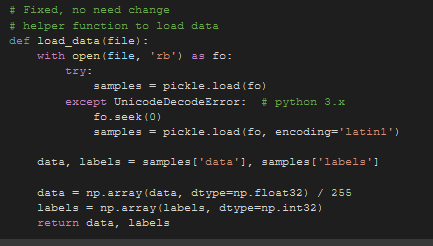
Constants used in Part A

For ease of coding and passing of parameters into the defined functions, these are the constants defined for all the questions in Part A.



Loading and pre-processing of data

Firstly, load both the dataset “data\_batch\_1” and “test\_batch\_trim” from Google Drive, using the helper function load\_data(file), as shown below.



For image dataset, the maximum value of each pixel of the RGB channel is 255. Hence, we will need to normalise our datasets that are passed into this function. We can do so by simply plickle.load our data and divide the numpy array value by 255. This ensures that the pixel values are normalised and stays within the range of 0 to 1 for better computations.

Helper functions

To ensure code readability and prevent repetition of similar codes, several helper functions are written:

***load\_data()***

* + Takes in the training and the testing data using pickle.load()
  + Returns the data in numpy array format and also the labels of the training data

***make\_model()***

* + Takes in the number of channels in the 2 layers and use it to train the model
  + Decide whether to implement dropout into the model or not
  + Returns the model

***train()***

* + Trains the model with different optimizers used
  + Saves the model

***plot\_image\_and\_feature\_map()***

* + Takes in a test image and use it to plot the feature map in various layers on the model

***plot()***

* + To plot the accuracy or loss of the model against the epoch count

***main()***

* + Acts as a switch to control which question should be executed in the code

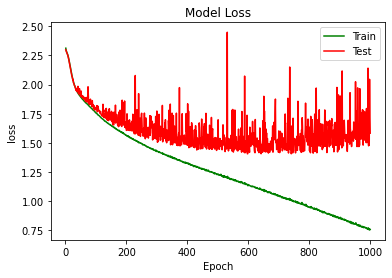
**Question 1**

Train the network using mini-batch gradient descent learning for 1000 epochs. Set the batch size to 128, and learning rate alpha=0.001.

a) Plot the (1) training cost, (2) test cost, (3) training accuracy, and (4) test accuracy

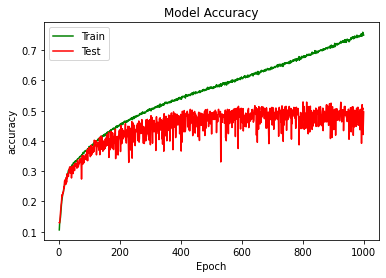
against learning epochs. One plot for the costs and one plot for the accuracies

**Training & Test Cost vs Epochs**

****

From the plot above, the train cost decreases gradually for 1000 epochs. The train cost can still be decreased further with more epochs. However, as can be observed from the test cost, the test cost plateaued and fluctuates for 1000 epochs at around the value of 1.50. Hence there may be signs of overfitting as the test cost begins to increase again from epoch 800 onwards.

**Training & Test Accuracy vs Epochs**



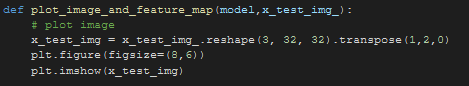
As seen from the plot above, the training accuracy increases gradually, while the test accuracy increases for the first 300 epochs and plateaued with an accuracy of 49.600%. There are still some fluctuations in the test accuracy as the training progresses.

b) For the first two test images, plot the feature maps at both convolution layers

(𝐶1 and 𝐶2) and pooling layers (𝑆1 and 𝑆2) along with the test images. (In total one

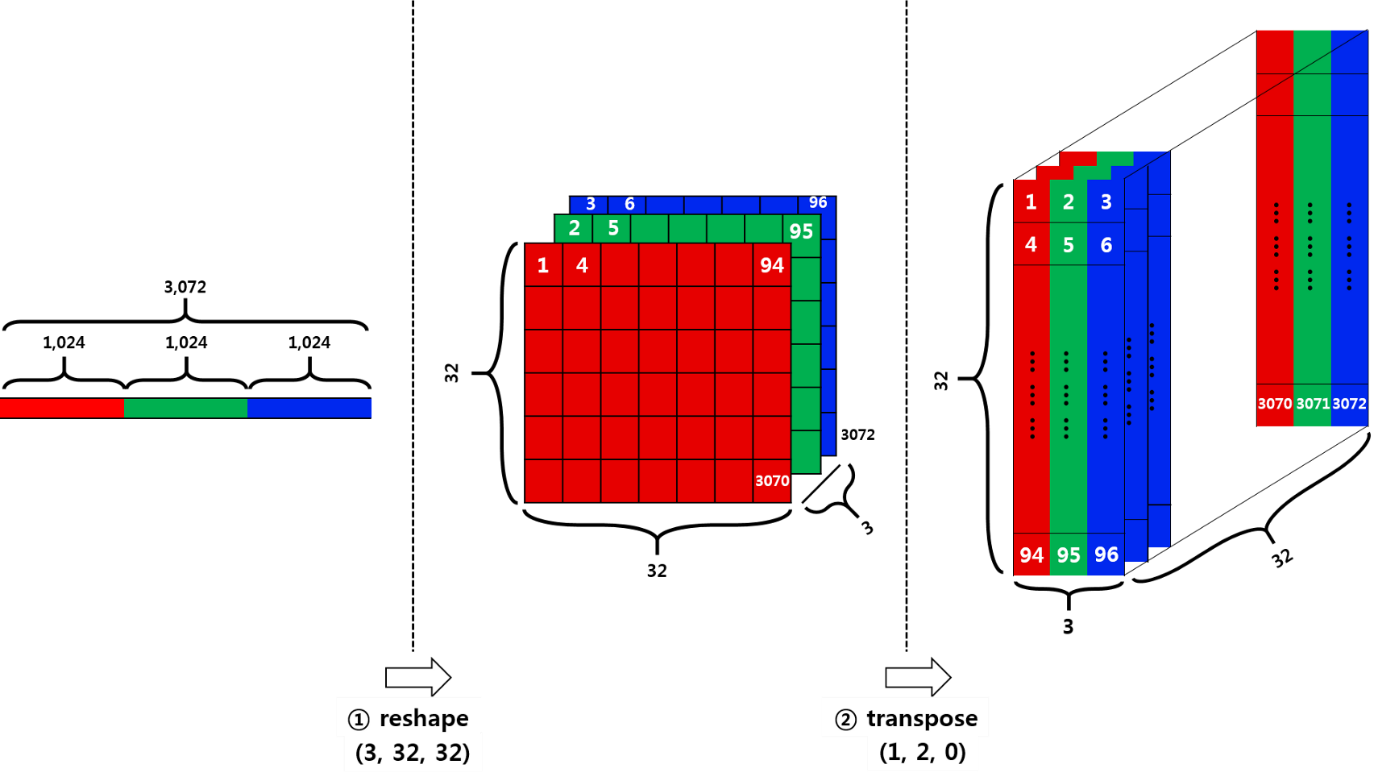
image and four feature maps)

To plot the test images, a function “plot\_image\_and\_feature\_map()” is defined and the row vector version of the test image is passed into this function. The row vector is then pre-processed, and it has 32\*32\*3=3072 elements. Two steps are required to reshape the row vector into the (width \* height \* num\_channel) from. The first step is to use the reshape function to reshape the image matrix. The second step is to use the transpose function in numpy. A code snippet of how the transformation is done is as shown from below:



From the CIFAR-10 dataset analysis by <https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c>, we first need to split the row vector into 3 pieces, each row represents a colour channel. For this, the resulting array will have (3x1024) matrix, making a (10000x3x1024) tensors in total. Then, divide the 3 pieces further by 32 because 32 is the width and height of the image. This results in (3x32x32) matrix, making it a (10000x3x32x32) tensors. As such, reshape function should be called with argument (10000x3x32x32).

From the pre-processing, observe that that one image data is represented as (num\_channel, width, height), a dimension that we are not expecting. We are expecting the data to be of the (width, height, num\_channel) dimension. To ensure that the image being fed in is the desired dimension, there is a need to swap the order of each axes using transpose. The transpose with argument (1,2,0) will be called as it will change the numpy array from (num\_channel, width, height) to (width, height, num\_channel). A diagram below illustrates the pre-processing.



Source: <https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c>

**Analysis for image 1**

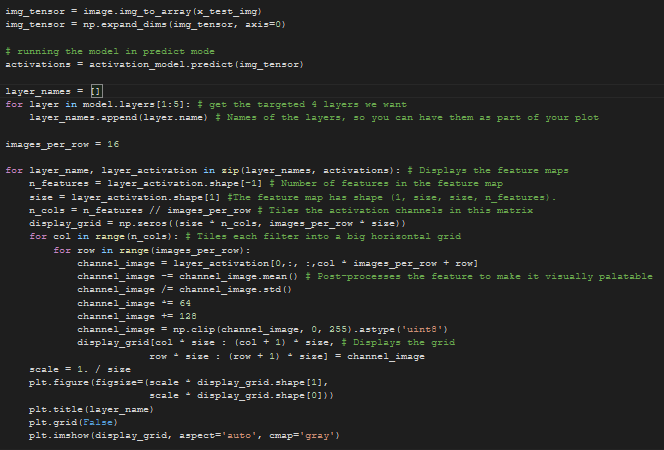
After pre-processing of the row vector, the first test image is as shown:



To plot the feature maps at both the convolution layers (C1 & C2) and pooling layers (S1 & S2), we first sift out the target layers first that we want to show (that is, C1, C2 , S1 & S2) as shown below.



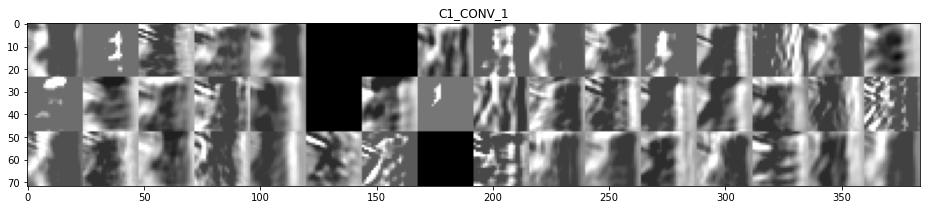
Then, we do some pre-processing to each of the layer so as to show the feature maps in the 4 layers. The corresponding code snippet is as shown below.



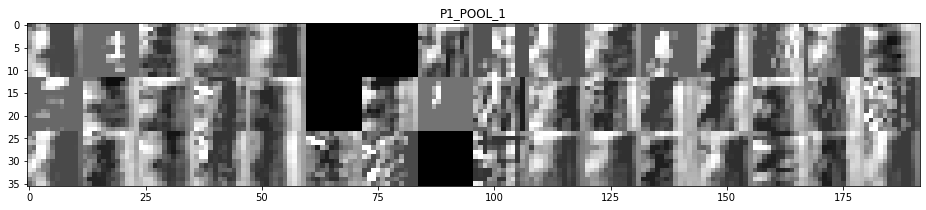
Source: <https://towardsdatascience.com/visualizing-intermediate-activation-in-convolutional-neural-networks-with-keras-260b36d60d0>

The four feature maps of the first image are as shown below:

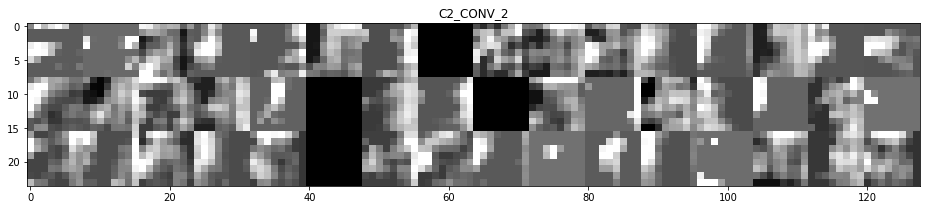
**Convolution layer C1**

****

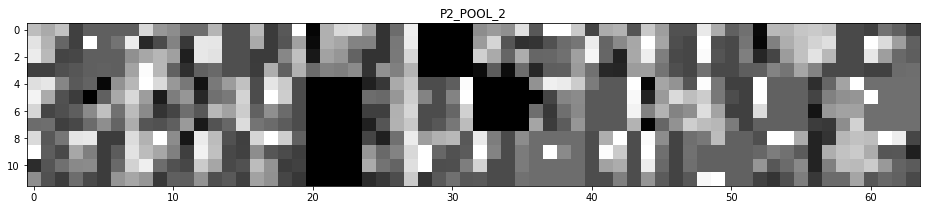
**Pooling layer S1**

****

**Convolution layer C2**

****

**Pooling layer S2**

****

As seen from the 4 feature maps above, full black images could be seen in the feature maps. This is because some of the filters are not activated yet. The first layer **C1,**retains almost all of the information present in the first test image. As we go deeper in the layers, the activations become increasingly abstract and less visually interpretable. They begin to encode higher-level concepts such as single borders, corners and angles. Higher presentations carry increasingly less information about the visual contents of the image, and increasingly more information related to the class of the image.

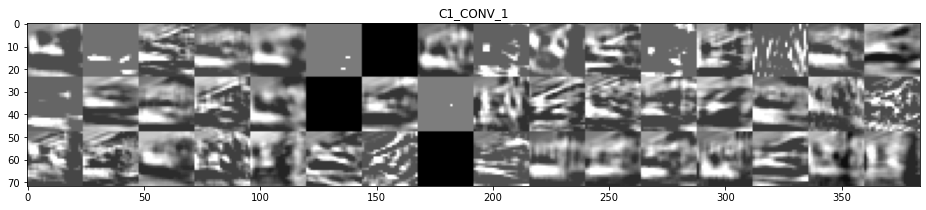
**Analysis for image 2**

Similarly, for the second test image, we obtained the following test image and feature maps.

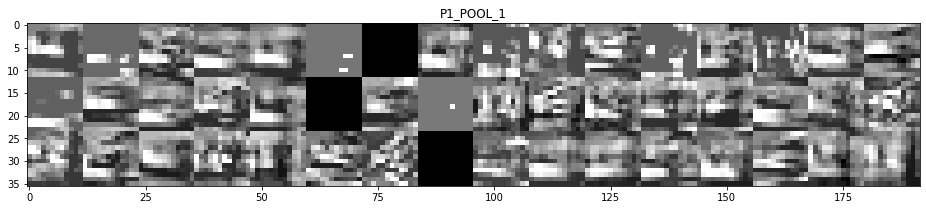


The four feature maps of the second image are as shown below:

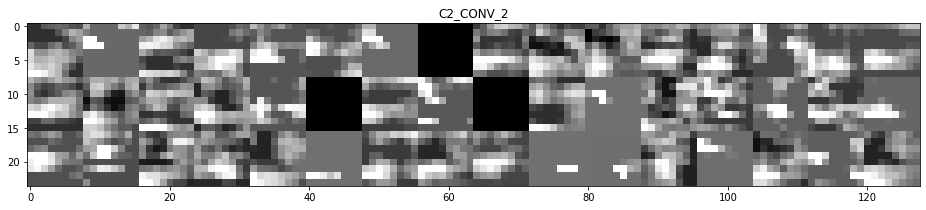
**Convolution layer C1**

****

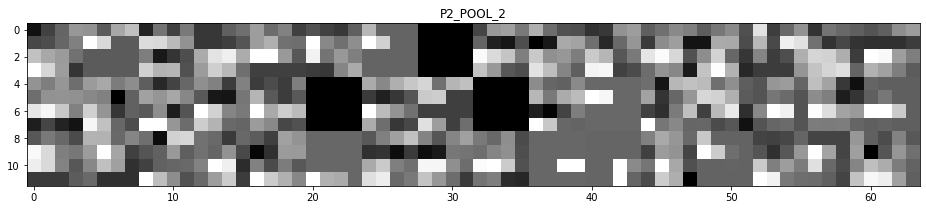
**Pooling layer S1**

****

**Convolution layer C2**

****

**Pooling layer S2**

****

**Question 2**

Use a grid search ( 𝐶1 = {10, 30, 50, 70, 90}, 𝐶2 = {20, 40, 60, 80, 100} , in total 25

combinations) to find the optimal combination of the numbers of channels at the

convolution layers. Use the test accuracy to determine the optimal combination. Report

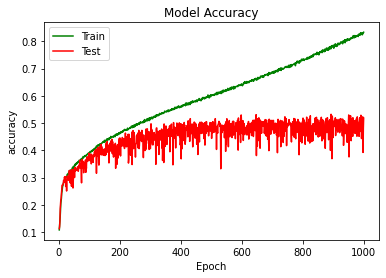
all 25 accuracies.

Using grid search, there will be 25 possible combinations of 𝐶1 & 𝐶2.Test accuracy is used to determine the optimal combination. Here are all the 25 test accuracies in terms of percentage (%).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 𝐶1 \ 𝐶2 | 20 | 40 | 60 | 80 | 100 |
| 10 | 46.550 | 49.250 | 47.500 | 48.700 | 48.900 |
| 30 | 46.350 | 46.700 | 46.250 | 51.200 | 48.750 |
| 50 | 47.350 | 49.350 | 48.700 | 47.100 | 51.350 |
| 70 | 48.400 | 51.000 | 51.050 | 50.550 | 48.050 |
| 90 | 49.250 | 50.300 | 50.450 | 49.850 | 51.850 |

From the table above, the highest test accuracy obtained is 51.850% with 𝐶1 and 𝐶2 equal to 90 and 100 respectively. Therefore, the optimal condition for 𝐶1 and 𝐶2 is 90 and 100 respectively.

Below is the accuracy plot against number of epochs of the most optimal condition.



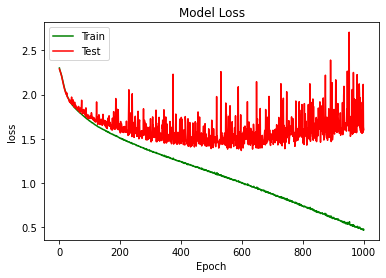
**Question 3**

**Optimal C1 = 90 and C2 = 100**

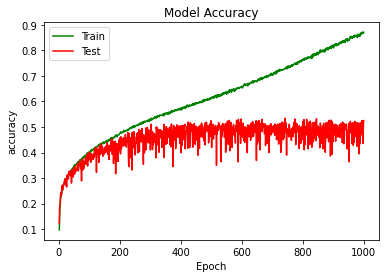
Using the optimal combination found in part (2), train the network by:

a) adding the momentum term with momentum 𝛾 = 0.1

**Costs:**

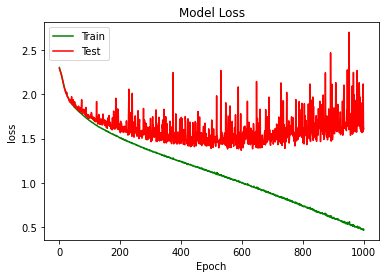
****

**Accuracy: 52.550%**

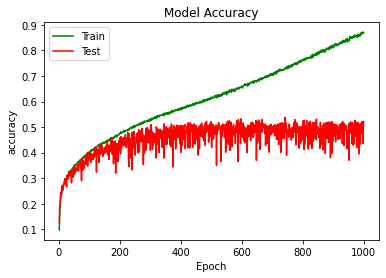
****

b) using RMSProp algorithm for learning

**Costs**

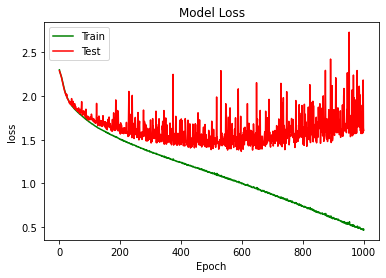


**Accuracy: 52.200%**

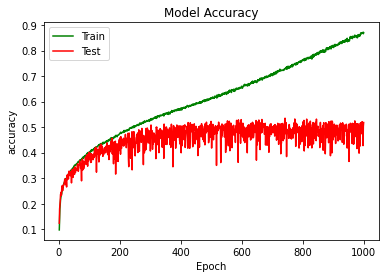


c) using Adam optimizer for learning

**Costs**

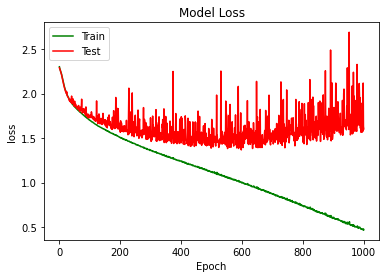


**Accuracy: 52.250%**



d) adding dropout (probability=0.5) to the two fully connected layers.

**Costs**



**Accuracy: 51.800%**

