Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	POW	Student ID:	19614652
Other name(s):	JIA SON		
Unit name:	MACHINE LEARNING	Unit ID:	COMP3010
Lecturer / unit coordinator:	SENJIAN AN	Tutor:	SENJIAN AN
Date of submission:	23/05/2022	Which assignment?	1 (Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: JIA SON POW Date of signature: 23/05/2022

(By submitting this form, you indicate that you agree with all the above text.)

Machine Learning Assignment Report

Jia Son Pow
19614652

Introduction

For this assignment, I used Skorch, which is a scikit-learn compatible neural network library on PyTorch, as that I was most familiar with that from the lecture videos posted by Harrison Outram on YouTube.

To help creating graphs and charts much easier, I have developed a list of Helper functions first.

Graph functions:
1. drawTrainingGraph - produces a training graph based off a Skorch NeuralNet Classifier.
2. drawBarChart – produces a bar chart.

Model functions:
1. trainModel – declares and fits a Skorch Classifier, then returns a trained Classifier.
2. testModel – calculates the prediction accuracies of models based on the testing data.
3. create_output_layer – initialise a new output layer
4. add_output_layer – adds a new output layer to a model

Data Extraction functions:
1. dataSetExtract – extracts data from the Pytorch built-in datasets into a numpy array.

Task 1: Model from Scratch vs Finetuned Pretrained Model

For this task, I have chosen both the model (ResNet18) and dataset (CIFAR-10) by random.

First, I downloaded the CIFAR-10 dataset. I split it into two sets: training dataset (cifar10_train) and testing dataset (cifar10_test).

Then, I extracted the needed data from the two datasets. The first time I attempted this, it crashed my runtime as it was using too much RAM, I decided to just purchase Colab Pro, and it was much better at the second attempt. For people who do not have access to that, the entries can simply be lowered to prevent potential RAM crashes.

After that, I downloaded both the pretrained and untrained versions of ResNet18 model. As CIFAR-10 is a dataset that has 10 classes, I adjusted both model versions to also have the same number of output channels. (From 1000 to 10)

Next, I trained the pretrained model, I split the training into categories based on the different hyperparameters I tested on it.

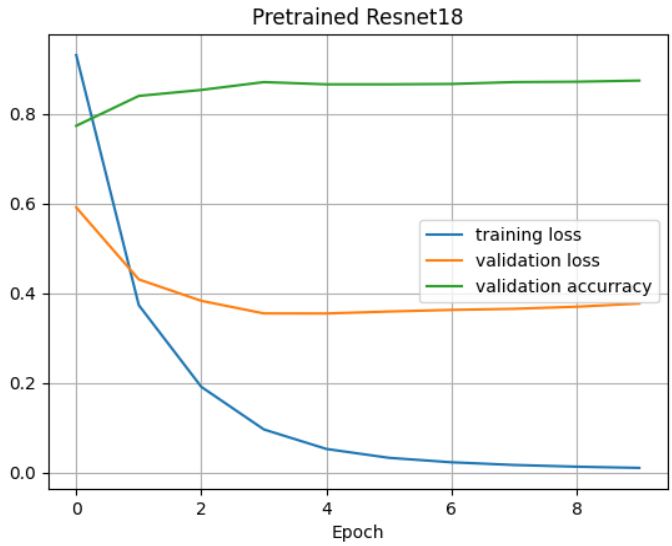
1. Number of epochs – small = 6, mid = 10, big=14
2. Learning rates – small=0.001, mid = 0.01, big=0.1
3. Optimizers – Adam, Stochastic Gradient Descent (SGD)
4. Loss functions – CrossEntropyLoss(CEL), Negative Log Likelihood Loss(NLLLoss)

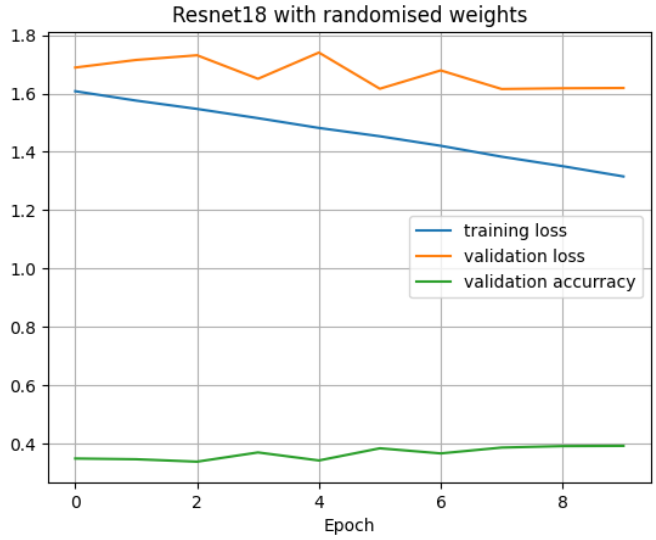
Each epoch took approximately 10 seconds to run, which made it take about an hour to run completely.

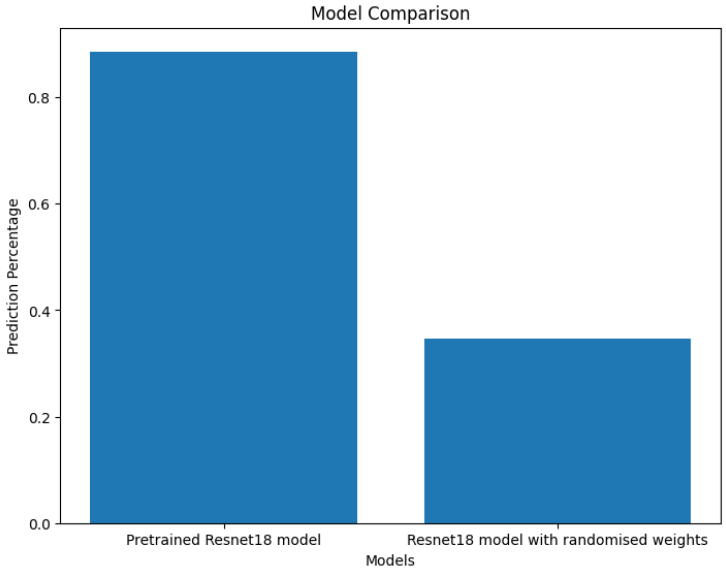
I made a list for the runs I've made through this dataset and found that the best hyperparameters that most suited the model was the combination of mid-epoch (10), mid learning rate(0.01), SGD optimiser and CEL loss function, which gave me a prediction accuracy of about 88.1%. This shows

that there needs to be a good balance of learning rate and number of epochs for a data to run through a model to produce the best results.

Using this set of best hyperparameters, I trained both the pretrained and untrained versions of the model against each other. Below is the graph of the results found:







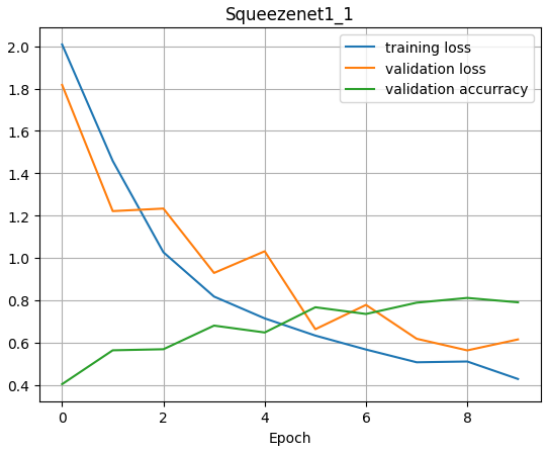
From this, we can deduce that the pretrained version performs much better than the ‘from-scratch’ version as it has better validation accuracy, a very low validation loss close to 0 compared to the ‘from-scratch’s 1.3, and a much higher prediction accuracy on test data.

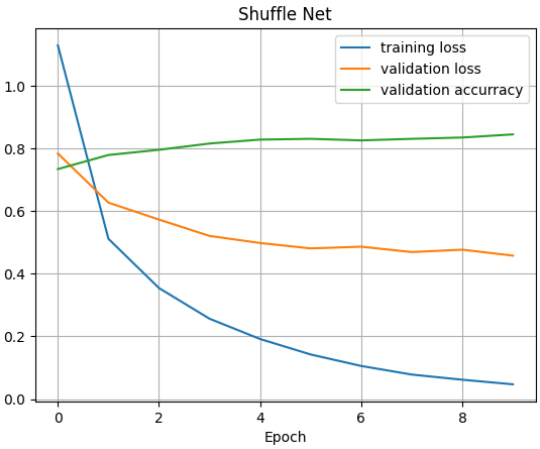
Task 2: Comparing the performance of different pretrained models

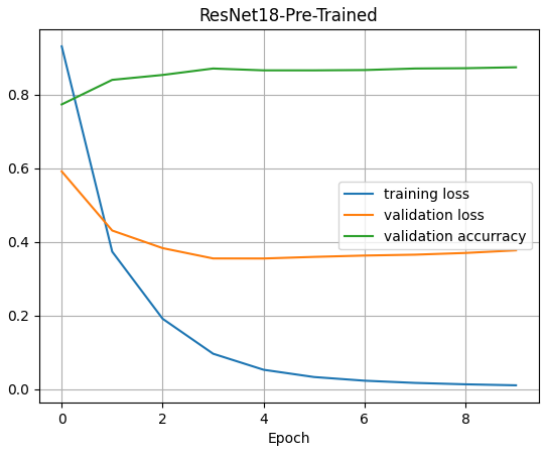
I used the pretrained ResNet18 model here again since it was the best model from task 1, alongside two other models. They are SqueezeNet and Shuffle Net.

To accommodate for the 10 classes of CIFAR-10 dataset, I changed the final conv2d layer of SqueezeNet and added an additional layer to Shuffle Net so that they can both have the same amount of output channels.

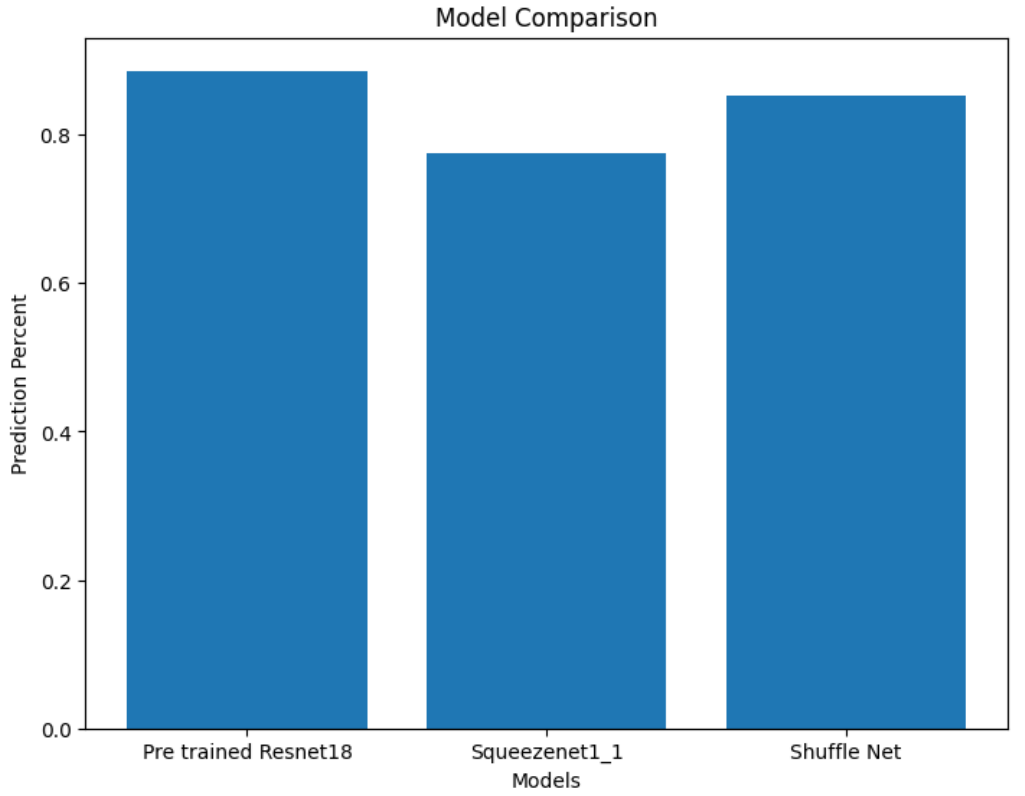
Then, I trained all 3 models using the same dataset and plotted graphs of the results I got:







Then, I compared the prediction accuracies of the models against each other on a bar graph:



From this, we can deduce that the Pretrained ResNet18 still performs the best out of the 3. Why? This is most likely because the hyperparameters set on the models for the tests were tuned to best suit ResNet18 (from task 1).

Task 3: Investigate the learnt CNN features from the last convolution layer and one intermediate layer.

I will be using ResNet18 again for this task as it is still the best performing model of the ones I have tested.

For this task, I first need to extract the features from the last conv2d layer and an intermediate layer. For ResNet18, the last convolutional layer is called the average pooling layer 'avgpool', and the intermediate layer I chose was layer2.

I then defined a hook function that I learnt from Harrison Outram's Tutorial 7 video to be used.

Next, I run each of these sets (original input, last conv layer & intermediate conv layer) through the model so that the hook function can be called and have the data added to the 'extracted_features' dictionary.

I needed to transform the intermediate conv layer features so that it can have the same dimensions as the output channels of the final conv layer. To do this, I need to make it so that the intermediate layer needs to have a total dimension of 512.

Since the intermediate layer has 64 channels, the (x,y) dimensions need to be equal to 8, which can be created by having average pooling input of size (56,56) to the size of (4,2).

The formula here
is used to calculate how much stride, padding and kernel size needed to get the desired output size.

Shape:

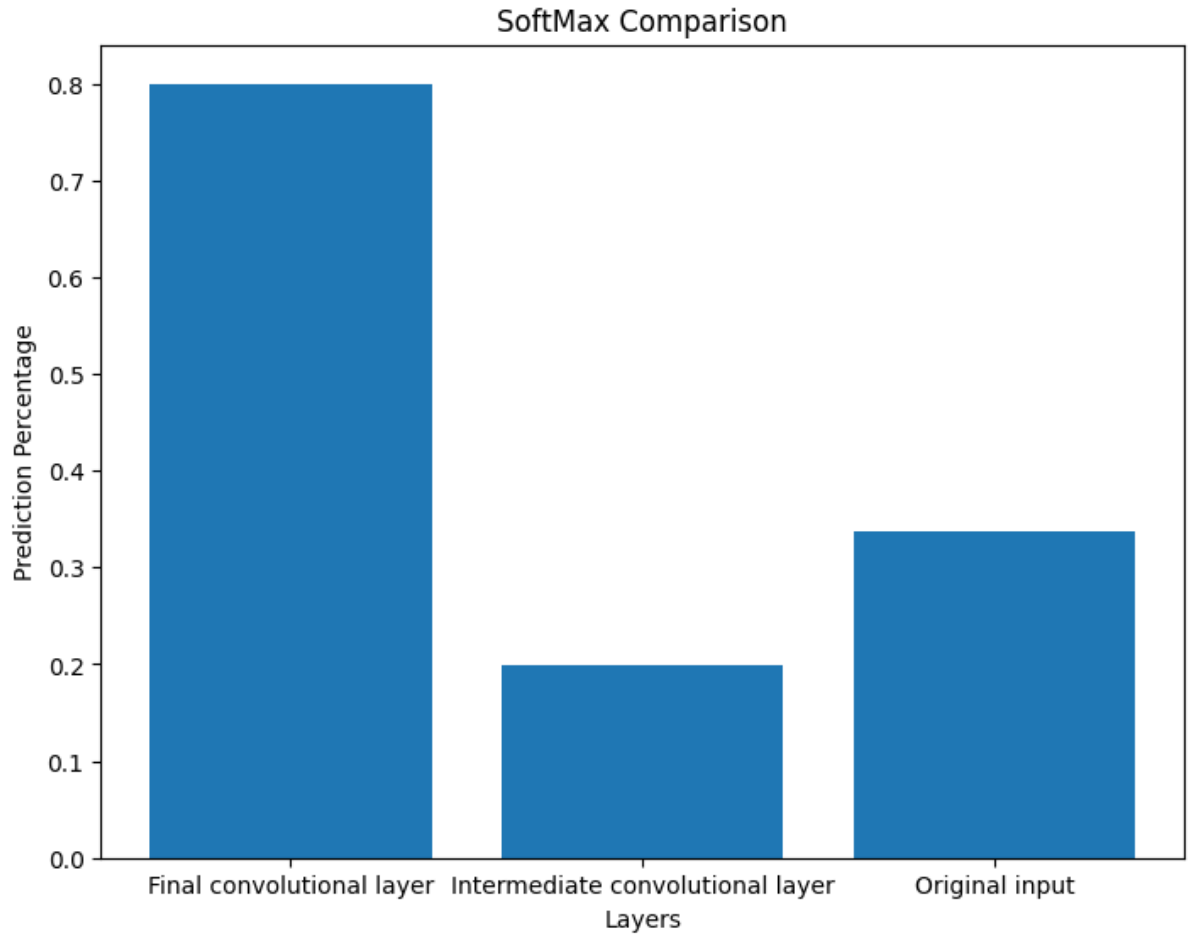
- Input: (N, C, H_{in}, W_{in}) or (C, H_{in}, W_{in}).
- Output: (N, C, H_{out}, W_{out}) or (C, H_{out}, W_{out}), where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{kernel\_size}[0]}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{kernel\_size}[1]}{\text{stride}[1]} + 1 \right\rfloor$$

Then, I implemented a softmax regression model that also has a flatten function to the final conv layer.

I tested all three sets against the model, and plotted a bar chart to show the results:



From this, it can be deduced that the Softmax Regression models performed better when trained on features from the last convolutional layer. It can be concluded that the features from the last convolutional layer provide information that is better suited to finding the correct label. This contrasts with the performance of similar models trained on feature from an intermediate layer, which was significantly lower.