Project Report

Dataset and Pre-processing:

After researching about different datasets for the task at hand, I found two datasets that were closest to what was required.

1. The Stanford Cars Dataset which contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images, and 8041 testing images where each class has been split into a 50-50 split. The classes are at the level of make, model, and year.
2. The second dataset that I found was the Vehicle Make Model Recognition dataset (VMMR) which contains 9170 classes consisting of 291,752 images. It covers car models manufactured between 1960 and 2016.

Reasoning for choosing the Stanford Cars Dataset:

At a first look, the VMMR dataset seems like an obvious choice since it has many more classes than the Stanford dataset and the number of images that can be used for training is also more. However, there were a few issues with using this dataset:

1. Availability of the dataset: After browsing the internet for some time I realized that there were numerous research papers, github repositories, and even Kaggle postings about the VMMR dataset but almost all of these sources did not include the actual dataset. Even when some did the dataset was incomplete and download instructions were ambiguous.
2. Computational Resources: The sheer scale and size of the dataset meant that the computational resources required to arrange the dataset and preprocess it would be a lot. Since, I am working on Google Collab it would take me days to even get the dataset ready for training.

On the other hand, the Stanford Cras dataset is relatively much smaller, and more readily available. There are a few issues with this dataset too:

1. There is a bit of class imbalance which means that the model we obtain might be better at recognizing some classes of cars than others.
2. It only contains models of cars till 2012 so the data is not that recent.
3. The amount of training data is relatively small which means that we may not be able to obtain a good classification accuracy with a new model. Thus, I realized to obtain a good accuracy we need to use transfer learning on a pre trained model.

Pre-processing Steps

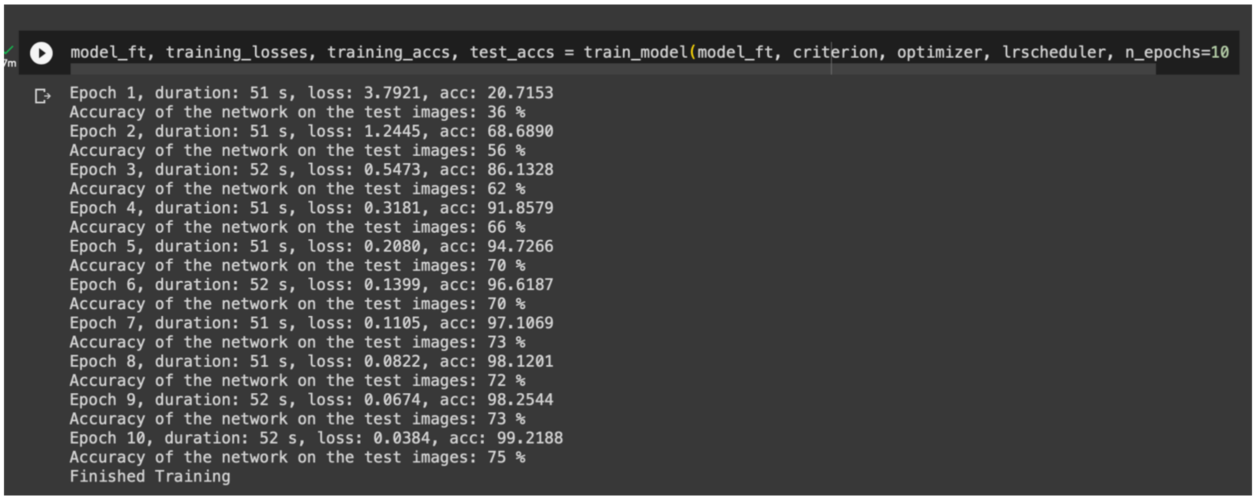
1. The Stanford cars dataset comes with an annotation file that has bounding box co-ordinates for each image. I used these co-ordinates to crop the images in the training dataset so that the cars would be more visible and easier to detect in the image. I did not do this for the testing dataset because I wanted to see if the model can still perform well on the testing dataset.
2. Then I structured the dataset so that each image is in a folder belonging to its own class. One important note is that while downloading the dataset it seems that some images are missing so some folders were empty and I had to deal with that error to ensure that there are no empty folders.
3. Then I applied data augmentation techniques to the dataset so that the dataset resembles real life scenarios where we might not get the images in the correct orientation. I applied the following transformations:
    a. I resized the images to 400 x 400.
    b. Applied a Random Horizontal Flip Transformation from Pytorch.
    c. Random Rotation of 15 degrees.
    d. For the testing dataset, I also normalized the images.
4. Lastly, I created a validation set from the testing set with a 20-80 split. So, I used 80% of the testing data as the test_set and 20% as the validation set to be used while training.

Training and Model Architecture:

1. Due to the small size of the dataset I decided to use transfer learning. As the first choice, I used the pre-trained ResNet-34 Model from pytroch which has been trained on the ImageNet Dataset which contains 100,000+ images across 200 different classes. This seemed like a good first option because the ResNet model is very adept at detecting objects.
2. I did change the last fully connected layer of the model and replaced it with a linear layer that outputs 196 classes since that is how many classes there are in our dataset. I used cross-entropy loss as the criterion and stochastic gradient descent as the optimizer.

Experiment 1

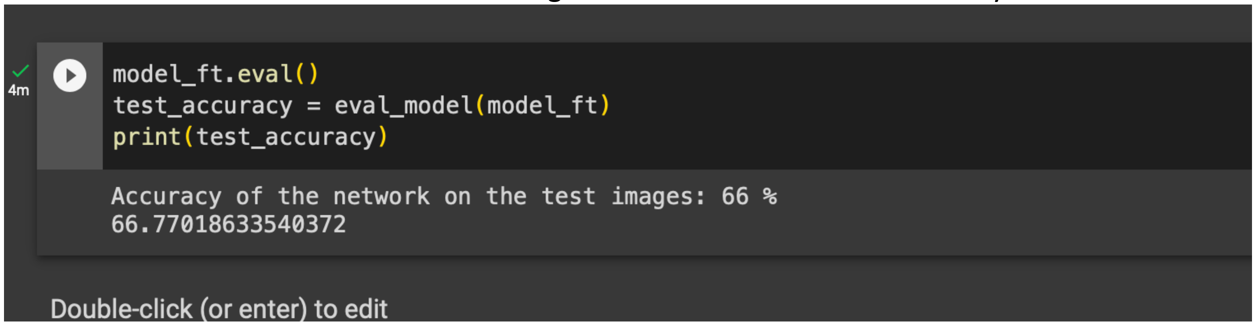
For, the first experiment I trained the model for 10 epochs and got the following results:



```
model_ft, training_losses, training_accs, test_accs = train_model(model_ft, criterion, optimizer, lrscheduler, n_epochs=10

Epoch 1, duration: 51 s, loss: 3.7921, acc: 20.7153
Accuracy of the network on the test images: 36 %
Epoch 2, duration: 51 s, loss: 1.2445, acc: 68.6890
Accuracy of the network on the test images: 56 %
Epoch 3, duration: 52 s, loss: 0.5473, acc: 86.1328
Accuracy of the network on the test images: 62 %
Epoch 4, duration: 51 s, loss: 0.3181, acc: 91.8579
Accuracy of the network on the test images: 66 %
Epoch 5, duration: 51 s, loss: 0.2080, acc: 94.7266
Accuracy of the network on the test images: 70 %
Epoch 6, duration: 52 s, loss: 0.1399, acc: 96.6187
Accuracy of the network on the test images: 70 %
Epoch 7, duration: 51 s, loss: 0.1105, acc: 97.1069
Accuracy of the network on the test images: 73 %
Epoch 8, duration: 51 s, loss: 0.0822, acc: 98.1201
Accuracy of the network on the test images: 72 %
Epoch 9, duration: 52 s, loss: 0.0674, acc: 98.2544
Accuracy of the network on the test images: 73 %
Epoch 10, duration: 52 s, loss: 0.0384, acc: 99.2188
Accuracy of the network on the test images: 75 %
Finished Training
```

As we can see, the training accuracy is very good at almost 99% and the validation accuracy is 75%. However, this difference also means that the model is overfitting to an extent and not generalizing as well to unseen data. This is also expected since we don't have a lot of data.

Then I ran the trained model on the testing dataset and obtained an accuracy of 67%.



```
model_ft.eval()
test_accuracy = eval_model(model_ft)
print(test_accuracy)

Accuracy of the network on the test images: 66 %
66.77018633540372
```

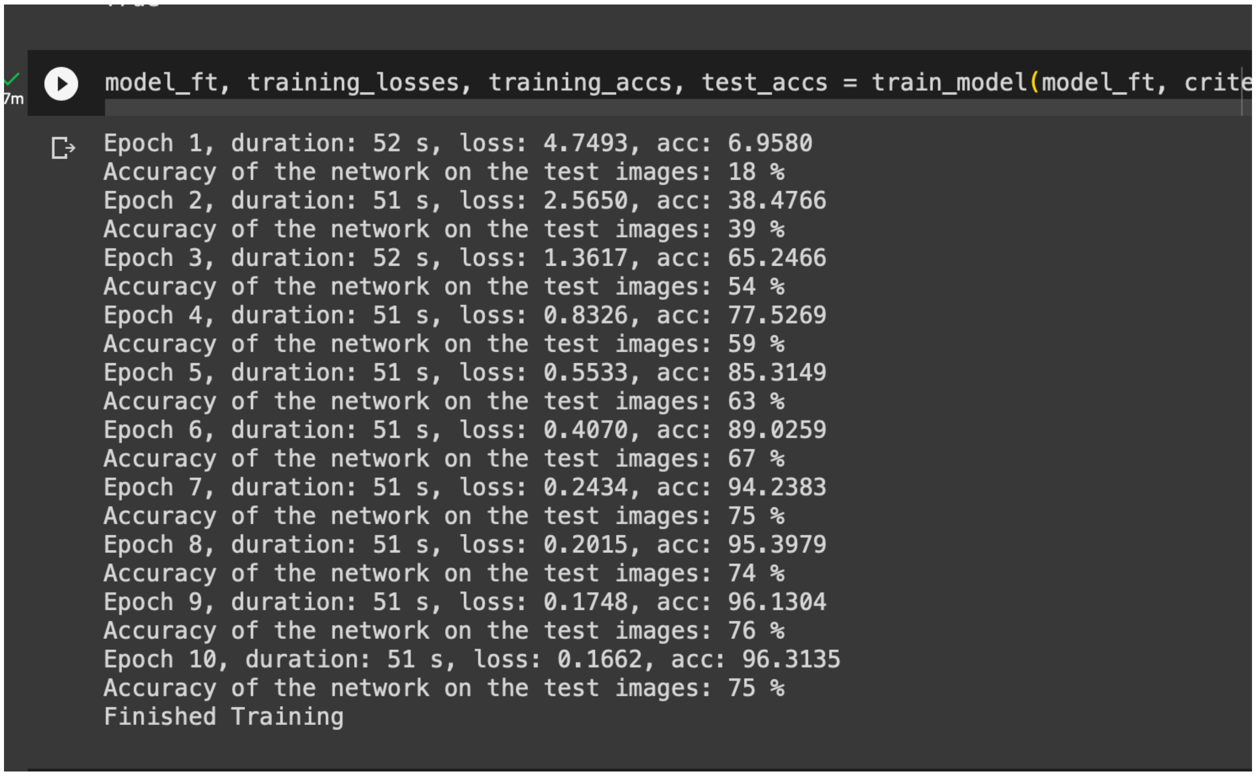
Even though, these results are promising, there is still the issue of overfitting and relatively lower accuracy on the testing set. Due to this, I decided to do a second experiment:

Experiment 2

In the second experiment, I made the following changes:

1. Added a dropout layer with a drop-out rate of 0.5 as an attempt to reduce overfitting.
2. Added a weight_decay rate of 0.001 to the learning_rate as another way to tackle overfitting.
3. Lastly, added a softmax layer at the very end.
4. Added a Vertical Flip data augmentation in the pre-processing phase so that the model becomes more accustomed to unseen data while training.
5. Changed the batch_size to 64.

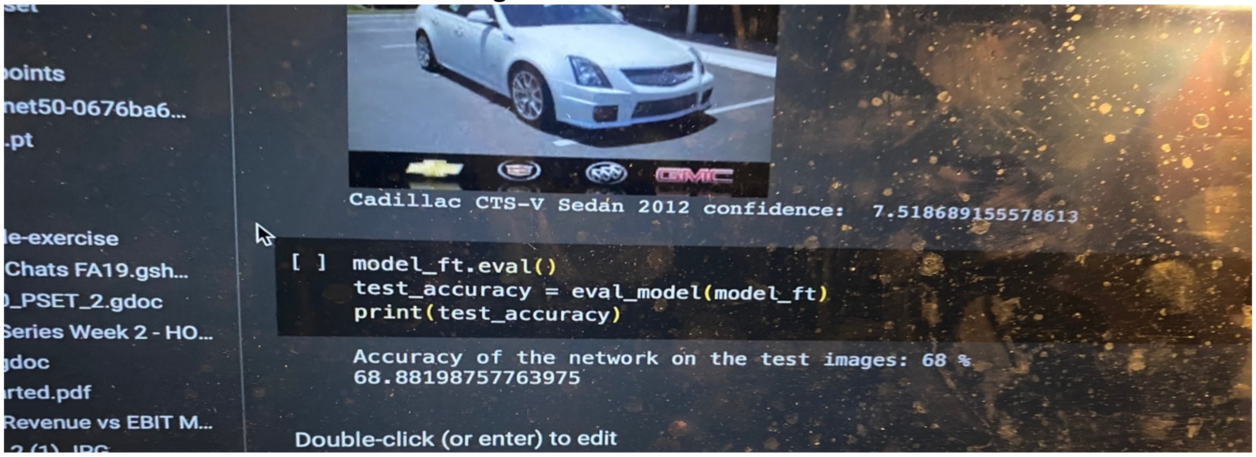
This experiment produced the following training and validation results:



```
model_ft, training_losses, training_accs, test_accs = train_model(model_ft, crite

Epoch 1, duration: 52 s, loss: 4.7493, acc: 6.9580
Accuracy of the network on the test images: 18 %
Epoch 2, duration: 51 s, loss: 2.5650, acc: 38.4766
Accuracy of the network on the test images: 39 %
Epoch 3, duration: 52 s, loss: 1.3617, acc: 65.2466
Accuracy of the network on the test images: 54 %
Epoch 4, duration: 51 s, loss: 0.8326, acc: 77.5269
Accuracy of the network on the test images: 59 %
Epoch 5, duration: 51 s, loss: 0.5533, acc: 85.3149
Accuracy of the network on the test images: 63 %
Epoch 6, duration: 51 s, loss: 0.4070, acc: 89.0259
Accuracy of the network on the test images: 67 %
Epoch 7, duration: 51 s, loss: 0.2434, acc: 94.2383
Accuracy of the network on the test images: 75 %
Epoch 8, duration: 51 s, loss: 0.2015, acc: 95.3979
Accuracy of the network on the test images: 74 %
Epoch 9, duration: 51 s, loss: 0.1748, acc: 96.1304
Accuracy of the network on the test images: 76 %
Epoch 10, duration: 51 s, loss: 0.1662, acc: 96.3135
Accuracy of the network on the test images: 75 %
Finished Training
```

As we can see, the changes brought down the training accuracy to about 96% and brought the validation accuracy up to 76%. Thus, there is a very small improvement in the overfitting but not a lot. The loss is also more for this experiment than for the other one.

The results of the model on the testing data were:



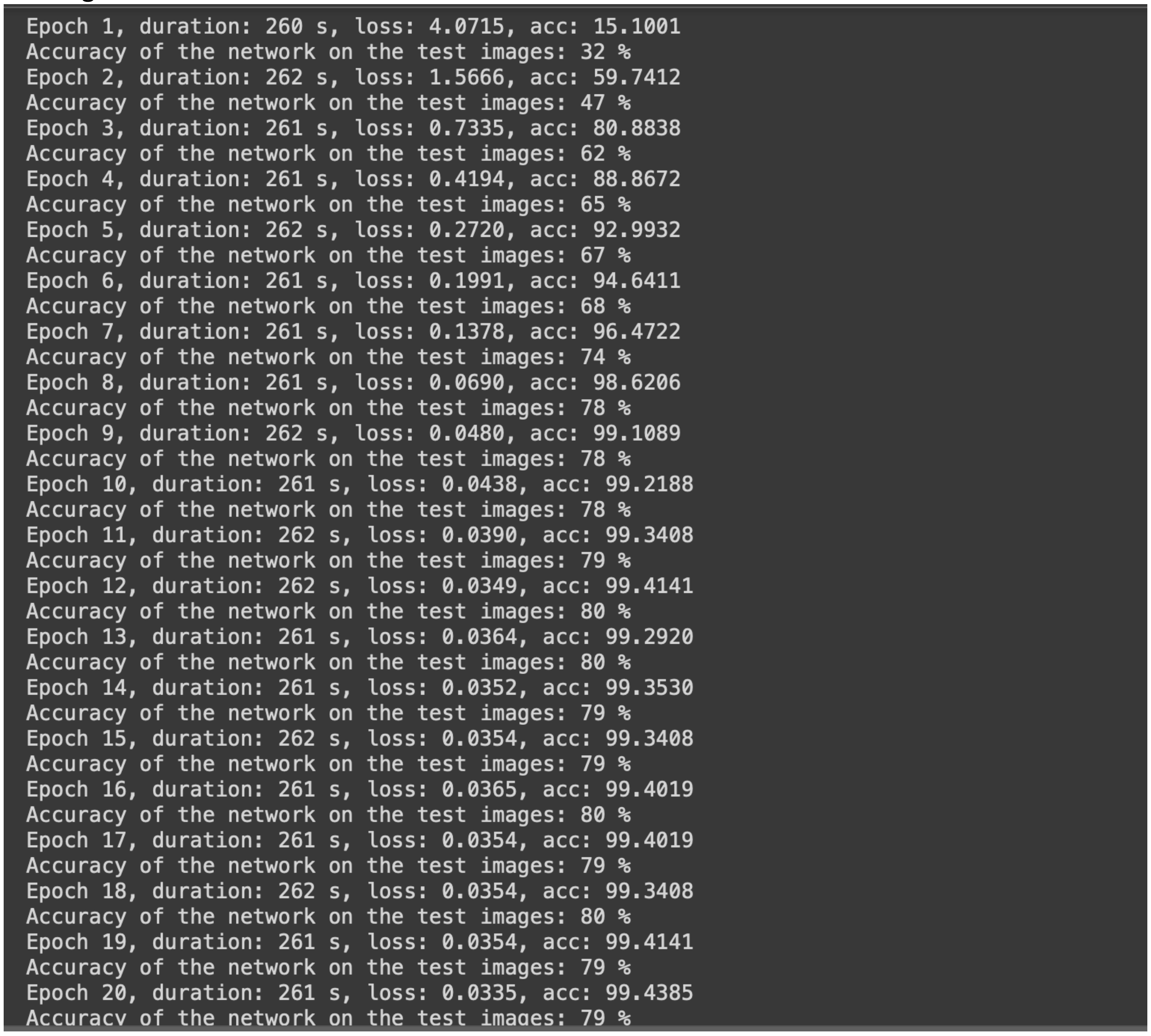
```
Cadillac CTS-V Sedan 2012 confidence:  7.518689155578613

[ ] model_ft.eval()
    test_accuracy = eval_model(model_ft)
    print(test_accuracy)

    Accuracy of the network on the test images: 68 %
    68.88198757763975
```

Double-click (or enter) to edit

So, the testing accuracy did improve to 69% but it is still not very high.

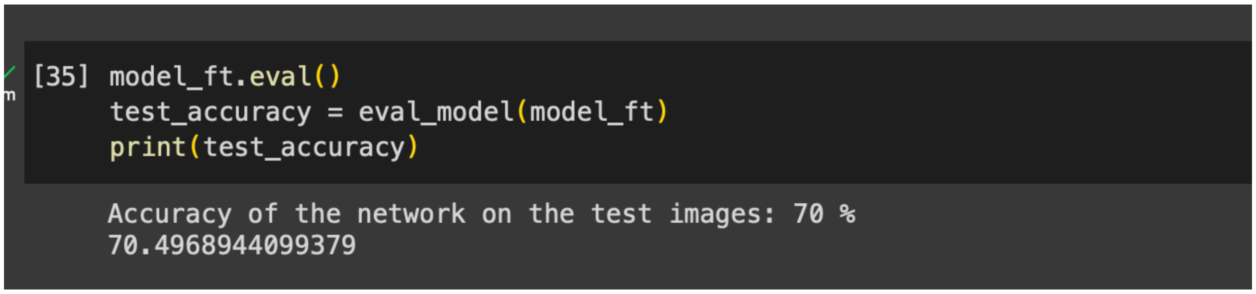
Experiment 3

After looking at the training data I thought that it might be a good idea to let the model run for more epochs and see how the accuracy responds. I also decided to use the ResNet 50 model for this experiment as the ResNet34 was not showing signs of improvement. The results of the training were:



```
Epoch 1, duration: 260 s, loss: 4.0715, acc: 15.1001
Accuracy of the network on the test images: 32 %
Epoch 2, duration: 262 s, loss: 1.5666, acc: 59.7412
Accuracy of the network on the test images: 47 %
Epoch 3, duration: 261 s, loss: 0.7335, acc: 80.8838
Accuracy of the network on the test images: 62 %
Epoch 4, duration: 261 s, loss: 0.4194, acc: 88.8672
Accuracy of the network on the test images: 65 %
Epoch 5, duration: 262 s, loss: 0.2720, acc: 92.9932
Accuracy of the network on the test images: 67 %
Epoch 6, duration: 261 s, loss: 0.1991, acc: 94.6411
Accuracy of the network on the test images: 68 %
Epoch 7, duration: 261 s, loss: 0.1378, acc: 96.4722
Accuracy of the network on the test images: 74 %
Epoch 8, duration: 261 s, loss: 0.0690, acc: 98.6206
Accuracy of the network on the test images: 78 %
Epoch 9, duration: 262 s, loss: 0.0480, acc: 99.1089
Accuracy of the network on the test images: 78 %
Epoch 10, duration: 261 s, loss: 0.0438, acc: 99.2188
Accuracy of the network on the test images: 78 %
Epoch 11, duration: 262 s, loss: 0.0390, acc: 99.3408
Accuracy of the network on the test images: 79 %
Epoch 12, duration: 262 s, loss: 0.0349, acc: 99.4141
Accuracy of the network on the test images: 80 %
Epoch 13, duration: 261 s, loss: 0.0364, acc: 99.2920
Accuracy of the network on the test images: 80 %
Epoch 14, duration: 261 s, loss: 0.0352, acc: 99.3530
Accuracy of the network on the test images: 79 %
Epoch 15, duration: 262 s, loss: 0.0354, acc: 99.3408
Accuracy of the network on the test images: 79 %
Epoch 16, duration: 261 s, loss: 0.0365, acc: 99.4019
Accuracy of the network on the test images: 80 %
Epoch 17, duration: 261 s, loss: 0.0354, acc: 99.4019
Accuracy of the network on the test images: 79 %
Epoch 18, duration: 262 s, loss: 0.0354, acc: 99.3408
Accuracy of the network on the test images: 80 %
Epoch 19, duration: 261 s, loss: 0.0354, acc: 99.4141
Accuracy of the network on the test images: 79 %
Epoch 20, duration: 261 s, loss: 0.0335, acc: 99.4385
Accuracy of the network on the test images: 79 %
```

This experiment definitely gave promising results as the validation accuracy increased to 80% and the training accuracy was pretty much the same. Then I tested the trained model on the testing set and it gave an accuracy of 70% which was once again higher than what I got in Experiment 2.



```
[35] model_ft.eval()
     test_accuracy = eval_model(model_ft)
     print(test_accuracy)

     Accuracy of the network on the test images: 70 %
     70.4968944099379
```

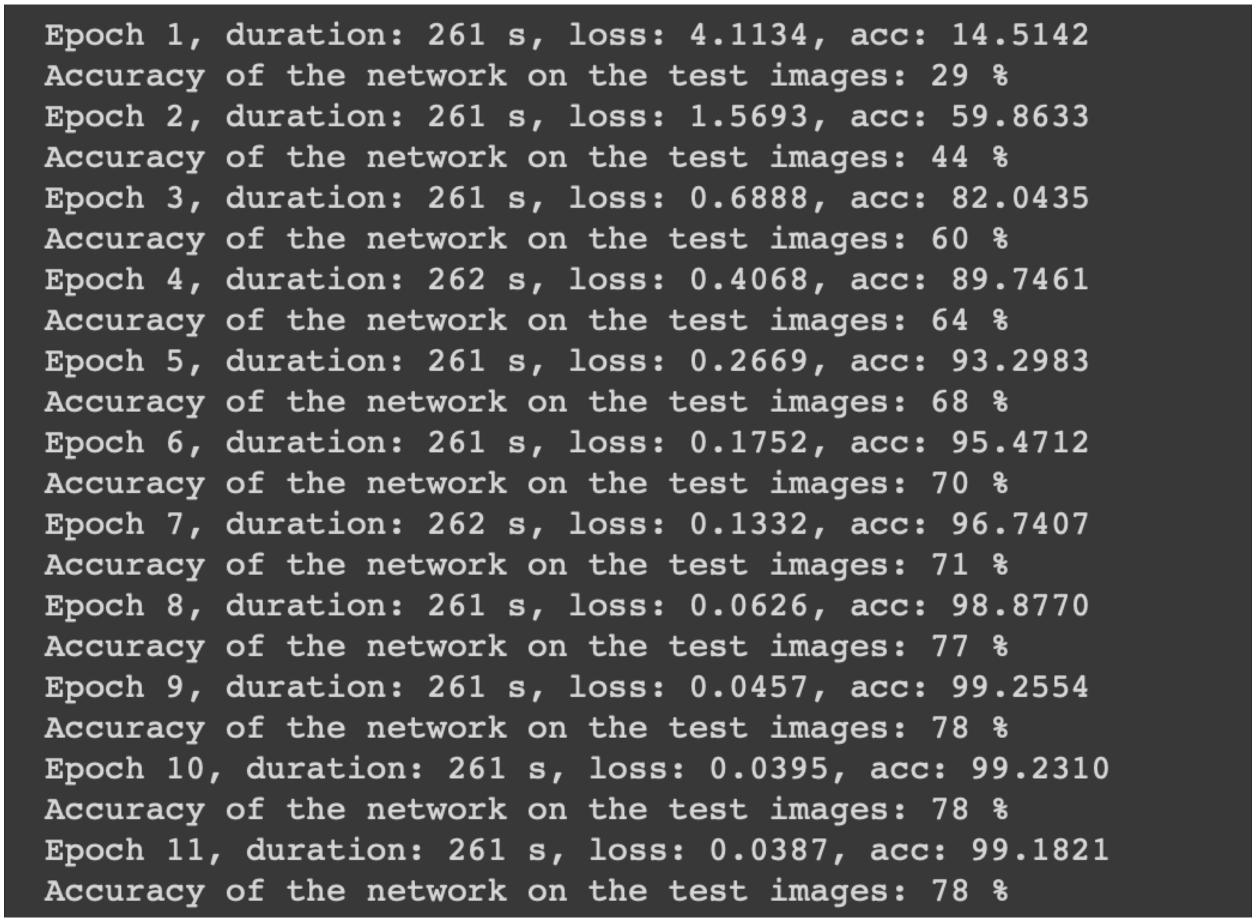
This definitely convinced me that the ResNet-50 might perform a bit better than the ResNet 34 version.

Experiment 4

At this point, I tried a lot of model fine-tuning with different dropout and weight decay factors but the performance of the model worsened. Thus, I decided to look over the dataset again and see what could be causing an issue. It seems to me that since the dataset is already really small, when I split the test dataset into a validation and test dataset the class imbalance increases as some classes might not be in the validation set at all. Thus, I decided to try another experiment but this time I did not make a validation set. Instead, I tested the model on the whole testing dataset after every epoch.

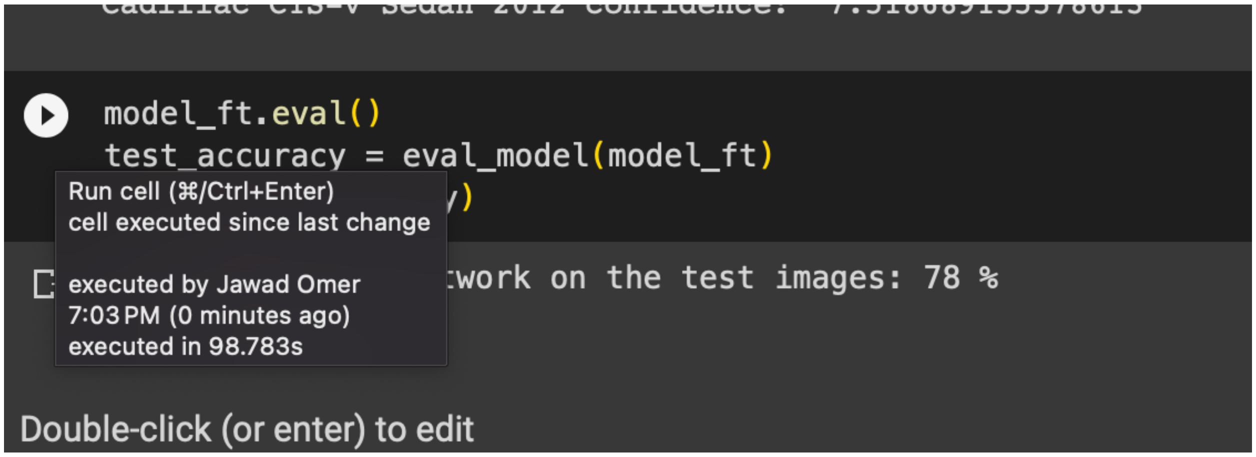
I also reverted back to a batch_size of 32 and used the pre-trained ResNet 50 model instead of the pre-trained ResNet 34 model. I trained the model for 11 epochs. There was no dropout or regularization applied.

The results were:



```
Epoch 1, duration: 261 s, loss: 4.1134, acc: 14.5142
Accuracy of the network on the test images: 29 %
Epoch 2, duration: 261 s, loss: 1.5693, acc: 59.8633
Accuracy of the network on the test images: 44 %
Epoch 3, duration: 261 s, loss: 0.6888, acc: 82.0435
Accuracy of the network on the test images: 60 %
Epoch 4, duration: 262 s, loss: 0.4068, acc: 89.7461
Accuracy of the network on the test images: 64 %
Epoch 5, duration: 261 s, loss: 0.2669, acc: 93.2983
Accuracy of the network on the test images: 68 %
Epoch 6, duration: 261 s, loss: 0.1752, acc: 95.4712
Accuracy of the network on the test images: 70 %
Epoch 7, duration: 262 s, loss: 0.1332, acc: 96.7407
Accuracy of the network on the test images: 71 %
Epoch 8, duration: 261 s, loss: 0.0626, acc: 98.8770
Accuracy of the network on the test images: 77 %
Epoch 9, duration: 261 s, loss: 0.0457, acc: 99.2554
Accuracy of the network on the test images: 78 %
Epoch 10, duration: 261 s, loss: 0.0395, acc: 99.2310
Accuracy of the network on the test images: 78 %
Epoch 11, duration: 261 s, loss: 0.0387, acc: 99.1821
Accuracy of the network on the test images: 78 %
```

As we can see the change definitely had a big impact on accuracy as the training accuracy was 99.18% and the testing accuracy was 78% up from the previous high of 70% in Experiment 2.



```
    Cadillac CTS-V Sedan 2012 Confidence:  7.518689153578615
```

```
model_ft.eval()
test_accuracy = eval_model(model_ft)
                                   y)
```

Run cell (⌘/Ctrl+Enter)
cell executed since last change

executed by Jawad Omer
7:03 PM (0 minutes ago)
executed in 98.783s

```
twork on the test images: 78 %
```

Double-click (or enter) to edit

I also tried experiments with adding a softmax layer, adding multiple linear layers, adding leakyRelu activation layers but they all brought down training and testing accuracy, so I decided to not put their results in the report.

Future Improvements

If we compare the accuracy obtained to a baseline model that always predicts the most frequent class then such a model would have an accuracy of around 5%. Thus, the accuracy of 78% is definitely promising.

However, I think that the following future changes might help in improving the performance of the model:

1. Increasing the training data by moving 5 to 6 images from each test folder to its respective train folder.
2. Trying transfer learning with another pre-trained model like Inception.
3. Making the model architecture more complex.
4. Access to more computational resources so that the model can be trained for more epochs.
5. Finding a complete version of the VMMR dataset.