Project implementation steps:

1. Load the data into the colab environment
2. Split the data in train, validation and test
3. Load images data
4. Finding mean and standard deviation
5. Transforming data
6. Creating a network
7. Training and validating the network
8. Experiments and results

1. Load the data into the colab environment

The first step for the implementation was to have an easy access to the dataset. Therefore, the dataset was stored in the google drive and it using google.colab module, it was mounted and used everytime. This also made saving and loading the data easy. Once the drive is mounted it is similar to saving and loading files/dir as doing it on a local machine.

2. Split the data in train, validation and test

Although torch has other ways to split the dataloader into train, validation and test. It was preferred to use the old school sklearn's way of splitting the dataset. It was decided that a separate dataloader would be created for train, validation and test as and when required using the split dataframe using train_test_split by sklearn. The split was done in the following fashion as suggested for the assignment

- **Total instances**: 11617
- **Train**: 10484 (~90.25%)
- **Validation**: 552 (~4.75%)
- **Test**: 581 (~5.00%)

3. Load images data

Using the tutorial from on pytroch's website, a similar data loading method was implemented [1]. The only major difference was that instead of reading the csv inside the Dataset class (CarTypeDataset), a pandas dataframe was directly passed to it. There were also other additional changes made in the class in order to incorporate some transformation that would be explained in later sections.

4 Finding mean and standard deviation

In order to normalize the images, it was necessary to find the mean and standard deviation across the training data. The process of calculation is commented at the moment in the notebook as it takes too long for execution. **The value of mean across the three channels is 0.4961, 0.5154, 0.5685 and value of standard deviation across the three channels is 0.0538, 0.0556, 0.0510.**

5. Transforming data

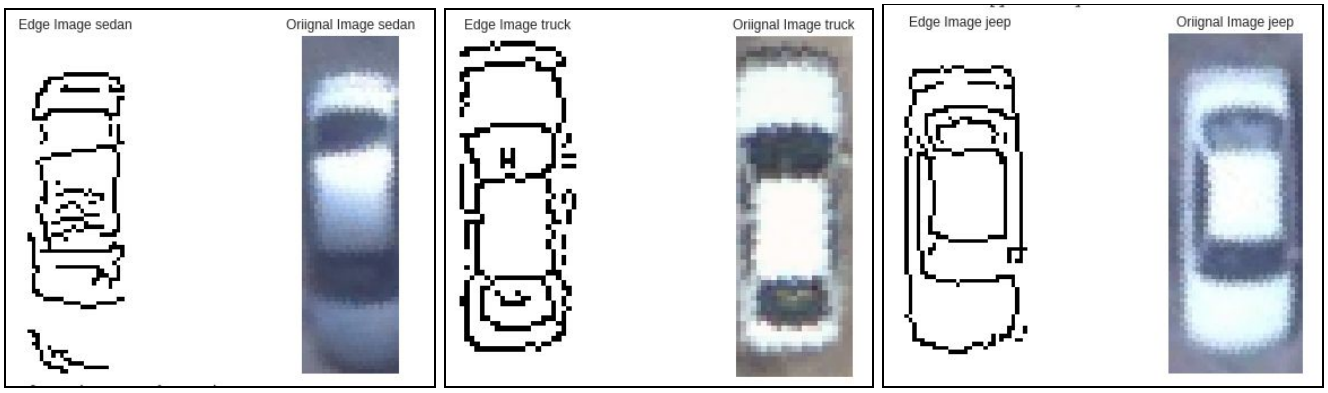
The data transformation was pretty convenient as pytroch has ready to use transformation methods for different types of transformation. Although a few transformation were done considering the formation of images, others were done just for experimental purposes. Following were the transformation applied to the images:

- **Resize**:
    - All the images were resized to a size of 72 height x 30 width
    - In order to find this width and height a mean of all the heights and widths was taken across the training images. The output was rounded off to the nearest integer value (72 and 30)
    - However, for restnet18 the input size had to be 224 for using a pretrained model
- **RandomHorizontalFlip**
    - It was observed in the training data that most of the images were vertical. Therefore, it was important to have some diversity in the orientation of the images, for which this augmentation was used.
- **RandomVerticalFlip**
    - It was used for the same reason as RandomHorizontalFlip
- **RandomRotation**
    - Some of the images in the training data were at an angle and for the model to learn better, the images were randomly rotated by an angle of 10 degrees. This transformation was not applied for the initial few runs. However, addition of this transformation didn't increase the accuracy score significantly. However, it is assume that it helps in better generalization of data
- **ToTensor**
    - For network to interpret the images, it has to be converted to tensors. This transformation takes care of it
- **Normalize**
    - It normalizes the images using the mean and standard deviation that was calculated across all the training data

- Normalization transform was only applied when the find_edges option was set to false. The description of find_edges will in discussed in the next subsection

Find Edges:

When going through the images, it was found that one of the major differences between all the different car types if their structure. The sedan had a longer hood compared to other car types and so on. Following are a few sample images with edges with their original counterpart. **The edges detection was done with canny module in python. The training and validation was done using both edged images and original images**. The only difference was that, **when using edged images, no normalization was applied**, as the image after finding edges was in itself a black and white image and there was hardly any need for further normalization.



(Fig 1.1) Images from left to right. Sedan, Truck and Jeep.

6. Creating a network

The initial idea was to create a simple network with two convolution layer with one fully connected layer. **The first conv layer had 3 input channels and 16 output channels. The second conv layer had 16 input and 32 output channels**. The fully connected layer had input depending upon the output of the final conv layer. The conv layer had a sequential, Conv2d, ReLU, MaxPool2d. Later in the experiment BatchNorm2d was added after Conv2d. The reason to use ReLU is because it is said to work better with CNN compared to sigmoid activation.

The kernel size was initially set to 3, stride to 1 and padding was set to 2. These parameters were changed to observe their effect on the accuracy results.

After the first basic iteration. **The number of conv layers were increased upto 10 and fully connected layers were increased upto 5**. **With every increase there was a bump in accuracy by 2% for the initial few epochs**. Although the accuracy become stagnant after a certain number of epochs.

Also, just to benchmark the scores with existing architecture, **resnet18 was used for a couple of experiments** with varying batch size and transformations.

7. Training and validating the network

The training dataset was divided into batches using dataloader and the data was set to run for 300 epochs. However, the number of **epochs were manually cut short if the accuracy or the losses became stagnant**. At least for the surety of results they were run for around 80-100 epochs with a initial batch size of 10. The batch size was intuition based. The very basic experiments were done with batch size of 50 and 100. However, due to low in those experiments results it was chosen to experiment with lower batch size.

Just to make sure that there was improvement in the results, two things were observed. First was the losses, it was expected that with every iteration and adjustments in weights, the model with learn something and the losses would reduce. Second, for more substantial results, the model was validated on the validation set for every 5 epochs. An increase in accuracy over validation set (the set of images that the model haven't seen or trained upon) and a slight reduction in loss built the confidence that it was worth giving more iterations to the model to learn.

In order to improve the process of optimizing the gradient descent, adam optimizer was used. Adam optimizer helps to converge to the best results in the quickest time. This could explain why the results were stagnant after a certain number of iteration for all the experiments, as the learning rate would have became so small that the model was reached the best possible results it could.

8. Experiments and Results

The experiments conducted are shown in the table below (Table 1.1), along with plots displays key differences in losses when varying batch size (Fig 1.2) and the training vs validation accuracy curves (Fig 1.3).

The ipynb (notebook) files for all the experiments can be found in the notebook folder. Sometimes the google colab used to disconnect half way through the execution and all the variables were lost. However, the models were saved with the best validation accuracy for that experiments in the models folder along with the project.

The find edges experiment was successful as the results were lower on find edges compared to original images.

The losses reduced very gradually in case of small batch size compared to larger batch size. The losses plots for a few of those experiments are shown in the report and remaining can be found in the losses folders.

The resnet performed similar to the best model from our own build network. The validation accuracy was 72% and test accuracy was 67% analogous to our experiment 4 where the validation accuracy was 71% but similar test accuracy was achieved.

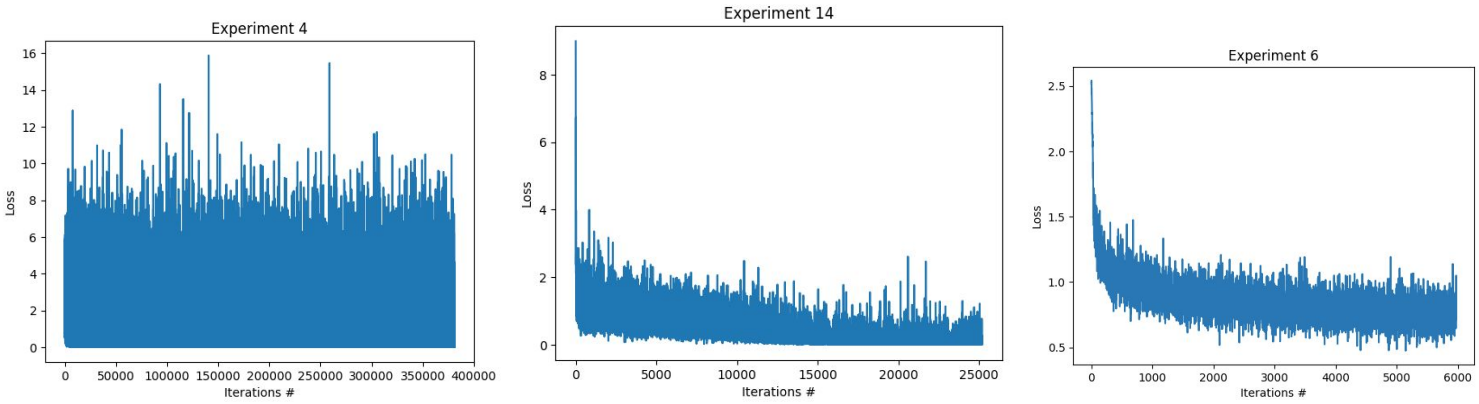
The best models were from experiment 4 and experiment 15. The experiment four used our own network and gave the best validation accuracy at 30 epochs beyond that it was observed that the model was not learning anything new but rather it was overfitting the training data. Similar problem was observed with resnet18 as the best learning was observed at 2 epochs and beyond that the training accuracy kept increasing without increase in the validation accuracy.

*(**Find_edges** = True means that the edges were found from the original images before classification)*

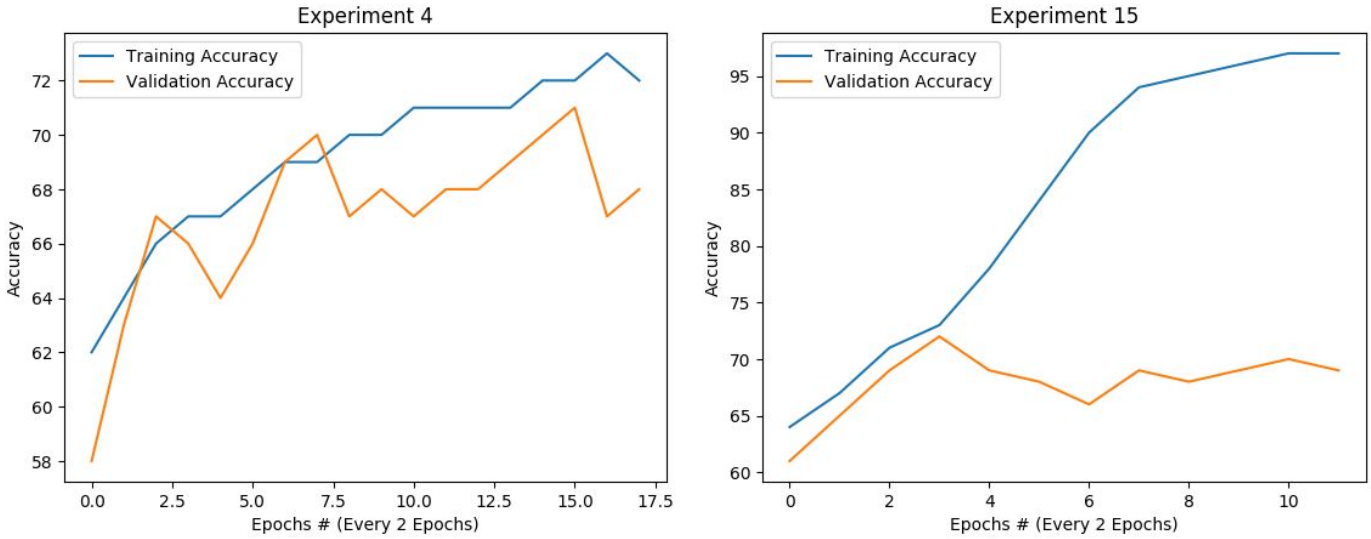
Exp #	Epoch	Batch Size	Learning Rate	*Find Edges*	Kernel Size	Neuron Count	Conv layer count	Fully connected count	Max Validation Accuracy	Test Accuracy
1	54	10	0.001	FALSE	3,3	32	10	8	69%	68%
2	59	10	0.001	**TRUE**	3,3	32	10	8	66%	63%
3	49	**1**	0.001	FALSE	3,3	32	5	3	68%	66%
4	**89**	**1**	**0.001**	**FALSE**	**5,5**	**32**	**5**	**3**	**71%**	**67%**
5	54	**10**	0.001	**TRUE**	5,5	32	5	3	67%	66%
6	*54*	100	0.001	FALSE	5,5	16	5	3	68%	67%
7	*89*	100	**0.0001**	FALSE	5,5	32	10	8	67%	66%
8	54	10	0.001	FALSE	10,10	32	2	2	68%	64%
~~9~~	~~29~~	~~30~~	~~0.001~~	~~**TRUE**~~	~~15,15~~	~~16 (maxpool = neuron_count)~~	~~10~~	~~5~~	~~58%~~	~~NA~~
~~10~~	~~14~~	~~1~~	~~0.001~~	~~FALSE~~	~~10,10~~	~~64(maxpool = neuron_count)~~	~~10~~	~~5~~	~~49%~~	~~NA~~
~~11~~	~~24~~	~~10~~	~~0.001~~	~~FALSE~~	~~15,15~~	~~32(maxpool=neuron_count/4)~~	~~10~~	~~5~~	~~60%~~	~~NA~~
12	94	10	0.001	FALSE	3,2	32	15	10	70%	67%
13	115	1	0.001	FALSE	5,5	64	2	2	68%	64%
14	13	1	0.001	FALSE	5,5	Resnet18			69%	65%
15	**25**	**10**	**0.001**	**FALSE**	**5,5**	**Resnet18**			**72%**	**67%**

(Table 1.1) All the combination of experiments conducted.

Experiment 9, 10, 11 produced the worst performing models. Green highlighted were the best performing models



(Fig 1.2) Plots showing the transition of losses when the batch size increases from 1 in the leftmost to 100 in the rightmost plot.



(Fig 1.3) Training VS Validation accuracy plots for the best performing models.

The best performance was observed at epoch 30 for experiment 4 (custom network) and 6 for experiment 15 (resnet18).

References

[1] https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

[2] https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks

[3] https://medium.com/ml2vec/intro-to-pytorch-with-image-classification-on-a-fashion-clothes-dataset-e589682df0c5