DETECTION AND CLASSIFICATION OF AUTOMOBILES BY BRANDS

Solar Shao
37962937
longges@uci.edu

Kaiwen Hu
16072357
kaiwenh4@uci.edu

Shuoran Meng
14851539
shuoran@uci.edu

ABSTRACT

In this project, we wanted to explore the accuracies and efficiencies of different types of convolutional neural networks(CNN) on classifying car brands. We started from a basic 8-layers CNN, to VGGNets with different numbers of convolutional layers, and finally to the residual neural network. It gave us a direct comparison of effects between the models with different complexities. The dataset we used contained car model images with a variety of lighting conditions and angles. It helped us examine our CNN architectures based on real-world scenarios.

1 INTRODUCTION

As technology becomes more developed and as machines are trained to think more like a human, more and more tedious and difficult jobs can be done by machines such as face recognition, audio translation. Machine learning and deep learning, as the foundation, promote the development of many fields including medical, economics, biology and security with the help of various models and algorithms including natural language processing and computer vision. Though computer vision is a popular topic nowadays and has dived into many fields and successfully solved many tasks, there are still real life tasks that are rarely researched and to be experimented due to continuously improved algorithms and deep learning models.

One of the tasks that is rarely researched but is important to different fields is car make and model detection. As suggested by economists, the composition of car model segments was positively related to the social economics statusLansley (2016). Car make and brand detection done by image classification would be the key to segment cars into different groups as the data preparation process for this business analytic task.

Car make and model detection and classification is also important to traffic safety system, resolving tasks ranging from detection of forbidden cars on road with restrictions, to detection discrepancy between registered car information and real car information by comparing license plate and our classification result, which could be essential to reduce the car accident rate and help police to find illegal cars and stolen carsKUNDURACI & Kahramanli (2019).

The paper is outlined as follows. Section two covered the related work done by other researchers. In Section three, the dataset information will be provided. Detailed data preparation process will also be included in this section. Section four will cover different models we used and quantitative results and experiment process will be included in section five. Conclusion and discussion on future work will be included in section six.

2 RELATED WORK

As the original CNN shows a outstanding performance on the image classification task, we began our branch classification with it. However, as the testing accuracy achieve a potential threshold, we think a simple CNN structure is not enough to achieve our aim. As a result, we put our sight on the well-defined model - the Very Deep Convolutional Networks for Large-Scale Image Recognition(VGG)Simonyan & Zisserman (2014). The VGG model is a variant of the CNN, the well-defined and deep structure makes the classification task more accurate on multiple-classifying task. Beside the VGG model, we also explored and selected a model called TResNet modelRidnik et al. (2021), which is a variant of ResNet model. Comparing to the basic ResNet model, the model heavily used Squeeze-and-Excitation layers in the model to improve the accruacy of the model with relatively lower computational cost.

3 Data

3.1 Dataset Information

In this project, we used a dataset collected completely from real life scenarios. Compared to other dataset, including car images that were either collected randomly from the Internet, or was collected in limited angles and lightning situations which deviated to real life situations, we used data that was taken by different cameras in different angles with various lighting conditions. High resolution videos were first recorded and then images containing appropriate vehicles were cropped out. The data was then manually labeled and splitted into a train and test folder for research uses.

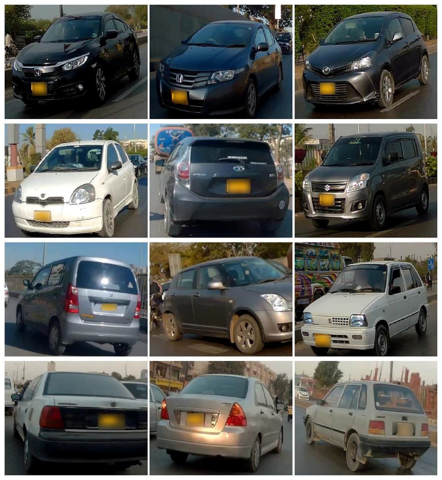


Figure 1: Sample image in dataset

We found this data that is best suitable for our task since it simulates real life scenarios well and was clearly labeled. The dataset contains 3847 images from 48 different classes that are common to see in real life. Each class represents a vehicle model with a label including brand name and model name(with year information in some classes).

After downloading the dataset, we visualized the train data by drawing the train data distribution plot over all 48 classes. The train data distribution is shown(Fig.2). As shown in the distribution, we found there is an issue of class imbalance in the dataset, with the largest group containing 270 images(Toyota corolla 2016) and the smallest group containing only 13 images(SUZUKI- Carry). The test data folder followed a similar pattern(Fig.3).

3.2 Data preparation using data augmentation

Though the dataset was already splitted into a train folder and a test folder, it is still meaningful to recombine the data and split it into three groups- train, validation and test so we would have greater confidence about our model result. However, given that there are only 3847 images in the full dataset, splitted it into three groups would lead to limited data size of each group. Thus, we faced the problem of a small data size.

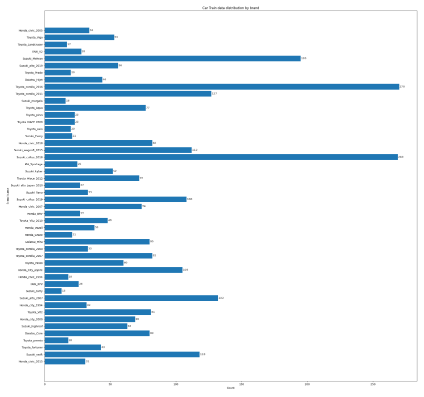


Figure 2: Train dataset distribution over 48 groups

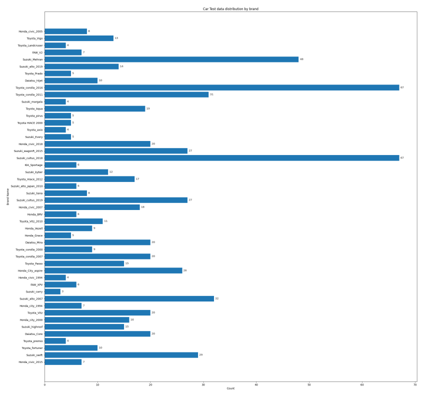


Figure 3: Test dataset distribution

In order to solve the dataset issues, we searched a lot of different methods including oversample, downsample etc. We eventually settled down with the solution using data augmentation since it helps solve data imbalance and expand the data at the same time. Data augmentation is a method that generates slightly modified copies of existing data in order to increase the amount of data. Various data augmentation methods of image modification include color jitters, center crop, rotation, blur, resize, and sharpen etc. We used the 'transform' module from 'torchvision' in PyTorch library. The transform module includes lots of useful functions for image modification. Few transform functions are shown as an example(Fig. 4).

We first recombined the dataset and spiltted it again into three groups- train, validation and test followed by the ratio 6: 2: 2. We treated each group independently and solved the data imbalance problem. The data augmentation processes are the same for all three groups, so we would take the training dataset as an example to illustrate the methodology. Training dataset data distribution figure was first drawn in order to see the skewness. Then, we selected a number as a threshold that is the closest to most class sizes while not too small for model training purposes. Classes in the training dataset were sorted and categorized according to the number of data in the class.

Different types and numbers of transform functions were applied to generate copies from the original classes in order to reach the data size threshold. In this process, we would apply more and more

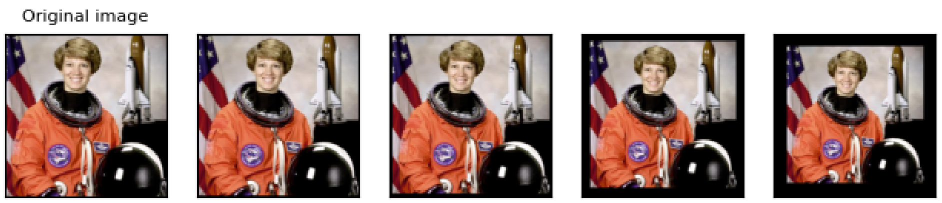


Figure 4: Illustration of transforms

diverse data augmentation to classes that are small. For classes that already contain a lot of data but still not reach the threshold, we would apply less data augmentation. For those that contain data exceeding the threshold, we do nothing about them. After expanding each group, some group size may exceed the threshold since it's difficult to find the exact multiplier of the original data size to threshold. We then randomly removed data from the modified dataset to the chopped out exceeded part.

Speaking about the transform functions, the frequency of being applied follows: color jitter, blur, sharpen and rotation, with color jitters being the first to be selected if we only perform one type of augmentation to the data. We followed this order because our project cares mainly about the car model and made. The color of the car doesn't change the car make but introduces new data. We used rotation less often because rotation of 90 degree would give a vertical car image, which we would definitely not appear in real life.

The example of data augmentation on our dataset is shown in the figure(Fig.5). After applying data augmentation, we have an exact balanced dataset for all three groups- train, validation and test. For the training group, each class has 200 images. For the validation group, each class has 75 images. For the testing group, each class has 80 images. New data distributions for three groups were drawn(Fig.6). We now have a total 16,960 images, with 9,600 training, 3,600 validation and 3,760 testing.

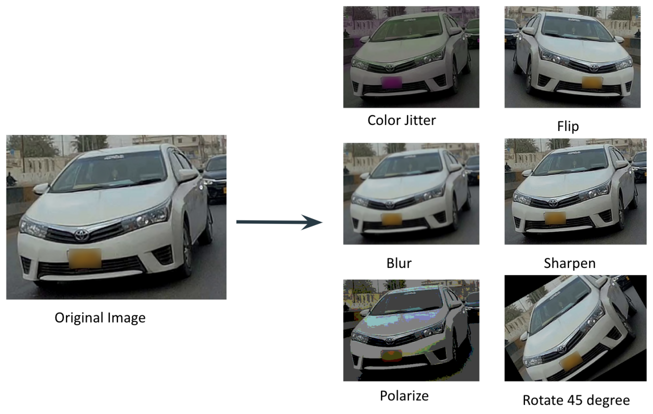


Figure 5: Data augmentation sample

4 Project Approaches

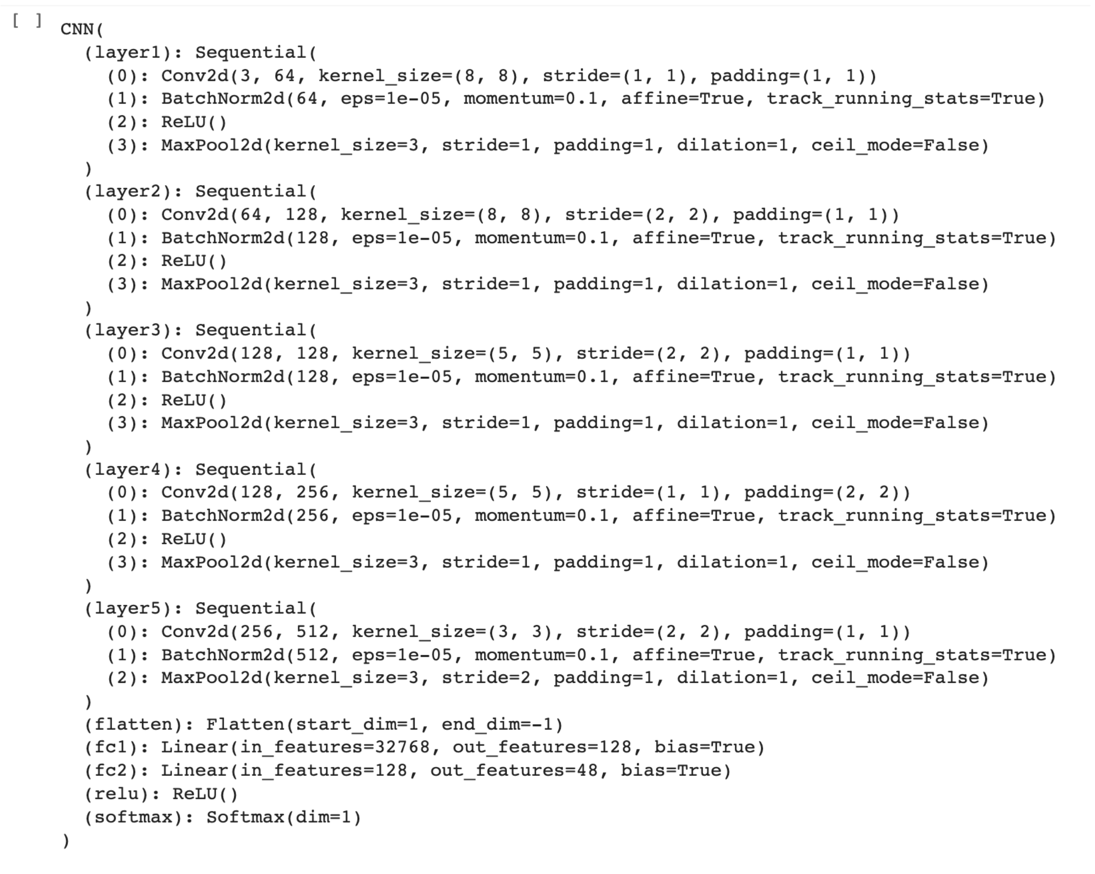
We trained our data on three different image classification models and compared the experiment result.

4.1 CNN

As the foundation of the most complex computer vision model, the traditional convolutional neural network architecture(CNN) is the first we trained and tested. We built an eight-layers CNN model using the 'PyTorch' library. The model contains five convolutional layers, each followed by a batch normalization layer to normalize the layer. It then goes into a Relu layer to filter out values that are less than 0. Maxpooling was then performed to reduce the dimension of the layer and downsampling. After the convolutional layers, a flatten layer and two fully connected layers were followed to do classification(Fig.6).

For CNN, we reshaped the image as a size 128 x 128 which would still contain important image features while not taking too long to train. We used an 8x8 kernel for the first convolutional layers with padding = 1 and stride = 1 to introduce parameters to attract image features while not reducing dimension. The second layer has the same kernel size and padding but a stride of 2 to downsample. The 5x5 kernel and 3 x 3 kernel was then used in order to extract a small pattern of the images.

Stochastic gradient descent and cross entropy loss were used for CNN training since this is a classification problem.



```
[ ]  CNN(
       (layer1): Sequential(
         (0): Conv2d(3, 64, kernel_size=(8, 8), stride=(1, 1), padding=(1, 1))
         (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (2): ReLU()
         (3): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
       )
       (layer2): Sequential(
         (0): Conv2d(64, 128, kernel_size=(8, 8), stride=(2, 2), padding=(1, 1))
         (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (2): ReLU()
         (3): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
       )
       (layer3): Sequential(
         (0): Conv2d(128, 128, kernel_size=(5, 5), stride=(2, 2), padding=(1, 1))
         (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (2): ReLU()
         (3): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
       )
       (layer4): Sequential(
         (0): Conv2d(128, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
         (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (2): ReLU()
         (3): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
       )
       (layer5): Sequential(
         (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
         (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (2): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
       )
       (flatten): Flatten(start_dim=1, end_dim=-1)
       (fc1): Linear(in_features=32768, out_features=128, bias=True)
       (fc2): Linear(in_features=128, out_features=48, bias=True)
       (relu): ReLU()
       (softmax): Softmax(dim=1)
     )
```

Figure 6: CNN architecture

4.2 VGG

Since a 8-layer CNN didn't give us an expected result, we target at the depth of the convolutional network. To examine whether a convolutional neural network with an increased number of simple layers instead of complex layers can give us optimal results, we introduce the VGG network. The VGGNet(or VGG model) refers to the paper "Very deep convolutional networks for large-scale image recognition" and has developed into a classic architecture of deep convolutional networks. It explores the effect of the depth of the convolutional network on its accuracy.

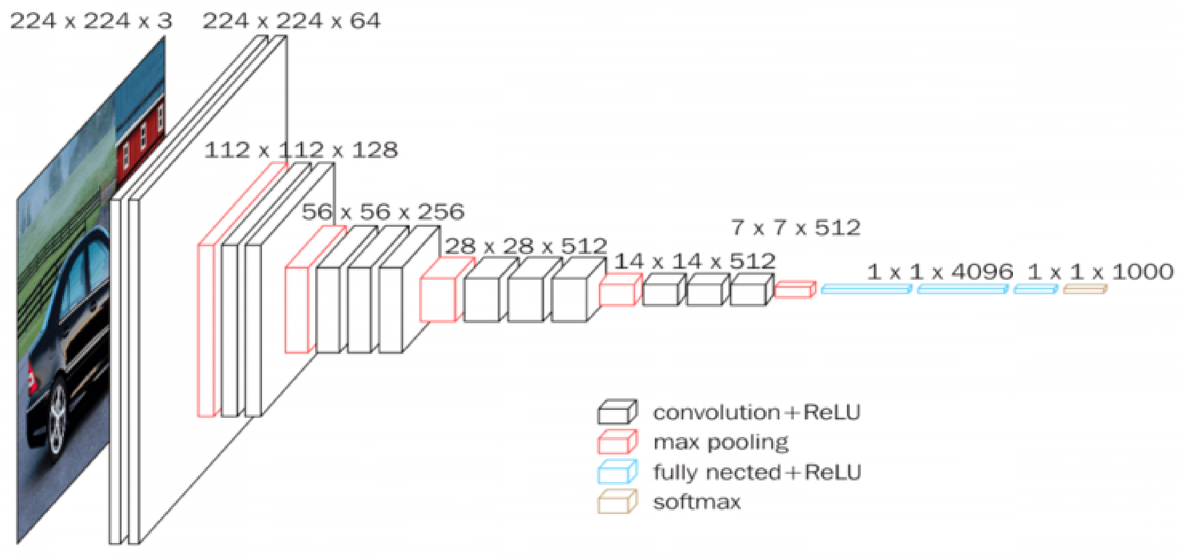


Figure 7: VGGNet architecture with different number of layer from the paper "Very deep convolutional networks for large-scale image recognition"

The figure 7 shows a 16-layers architecture of VGG(The number of the layers is typically vary from 11 to 19). It contains a stack of convolutional layers with Rectified Linear Unit activation function. Each filter is used with a small receptive field3 x 3.

The convolution stacks are followed by three fully connected layers, two with size 4,096 and the last one with size 1,000. The last one is the output layer with Softmax activation. Spatial pooling is carried out by five max-pooling layers which help effectively downsample the output of the convolutional layer. Furthermore, there are 3 Fully-Connected layers which follow a stack of convolutional layers (can be different according to the numbers of layers). First two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). then the output of 3rd fully connected layer is passed to softmax layer in order to normalize the classification vector. The configuration of the fully connected layers is the same in all networks of VGG network

4.3 TResNet

As the top model in classifying the Standford Car Dataset, we try to train the TResnet model as our final model. The basic block structure of the model are shown in the Figure 8. This model uses many Squeeze-and-Excitation layer(SE layer) to improve the performance of the model with relatively low computational cost.

We are using the TResNet-M in our project. The "M" means the model size or the amount of the parameters in the model. The "M" TResNet contains 24 layers in total; the model contains 1 Space to Depth layer, 1 1x1 Convolution layer, 3 56x56 basic blocks with SE layer blocks, 4 28x28 basic blocks with SE layer blocks, 11 Bottleneck with SE layer blocks, 3 bottleneck blocks followed by a Global Average Pool layer, and finally the soft-max block for outputting classification result.

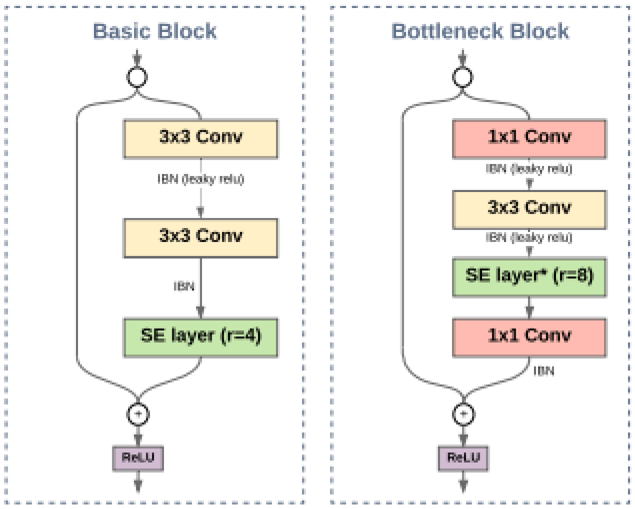


Figure 8: TResNet architecture

5 Quantitative Result

5.1 CNN Result

The CNN model ran 200 epochs with batch size equals to 50 images and learning rate 0.005. The model was trained on Google Colab using GPU for four hours. Hyperparameters were modified and we found that the current hyperparameters gave the best validation accuracy. Train accuracy after 100 epochs was 76.46% with a training loss 3.15. Validation accuracy after 100 epochs was 36.28% with a validation loss 3.58. The final train accuracy after 200 epochs reaches to 87.30% with a loss of 3.03. The final validation accuracy after 200 epochs reaches to 49.05% with a validation loss of 3.46. The training and validation processes are recorded and shown in Fig 9 and Fig 10. After training and validation, the model was tested on the test dataset. The testing accuracy was 47%.

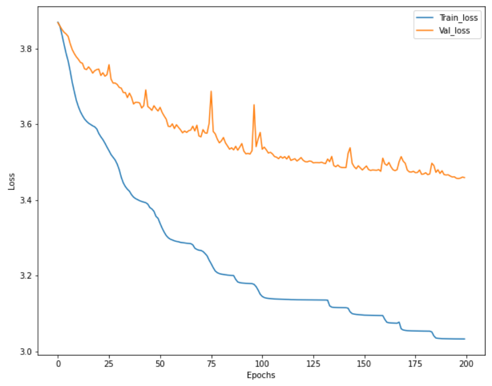


Figure 9: CNN train validation loss

5.2 VGG Result

We started our classification from a 11 weighted layers VGG with 150 epochs and set learning rate as 10e-3. We can see that after about 85 epochs the model become overfitted, but the test accuracy

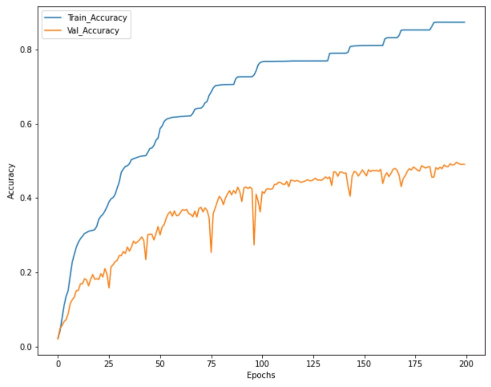


Figure 10: CNN train validation accuracy

can reach to 70.83%. Then we increased the numbers of layer to 16 and trained the model with the same parameters. The model became overfitted earlier than the 11-layer VGG but the test accuracy increased to 71.13%. Based on the results, we think the test accuracy of 19-layer can be even higher. However, the test accuracy dropped to 64.13% since the model became overfitted soon.

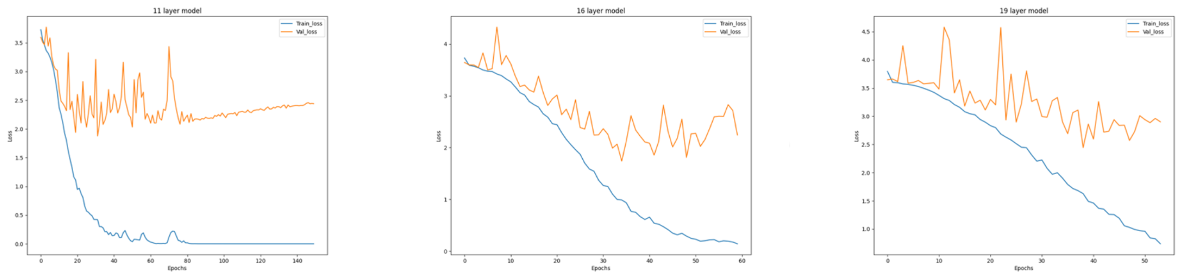


Figure 11: VGG loss by epoch.(11-layers, 16-layers, and 19 layers)

5.3 TResNet Result

We trained the TResNet Result many times by changing the hyper-parameters of the model. In the beginning of the modeling, we tried the smallest TResNet model - TResNet-M with learning rate equals in [0.1,0.05,0.02,0.01,0.002]. We finally found that 0.02 is a fair learning rate for this model. The performance of the model is shown at Figure 12. By setting the batch size to 64, learning rate to 0.02, and early stopping when loss is less than 0.01, the model achieved the 83.5% of the accuracy on testing data in 58 epochs. After that, we trained the TResNet-L model, which has double the parameter than the TResNet-M. During the training process, we found that testing data output a relatively low accuracy. We tried to drop the test data and use the validation data as the testing data, which means there is no validation data in the later training. By doing so, we achieved a testing accuracy we stated previously.

6 Conclusion and discussion

Among all three models we tried, the TResNet shows the highest performance with a low computational cost. As a result, our final model reaches 83.5% of the accuracy in testing meaning our model should be able to identify most of the car branches correctly. However, we still have room for us to improve. Many of the images in the dataset don't show the characteristic of certain car branches,

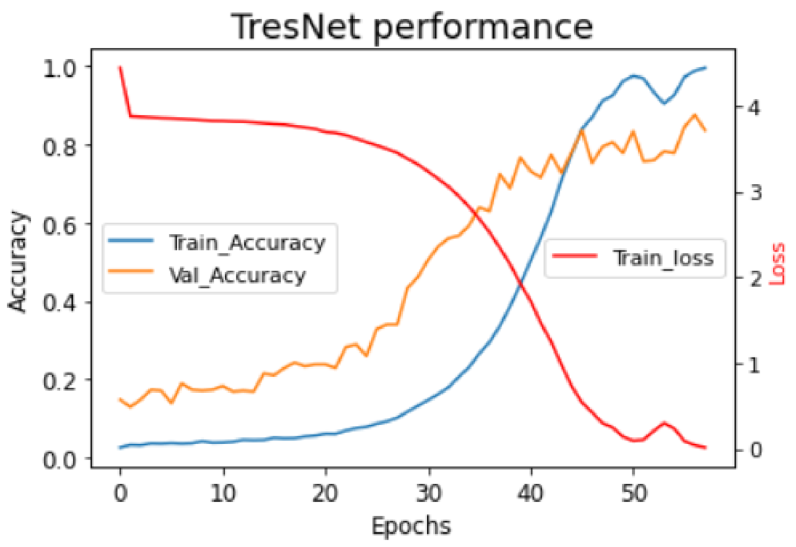


Figure 12: TResNet train validation accuracy

which might be a potential reason that the testing data gives a lower accuracy while the training accuracy is significantly high. Also, we face a limitation on the computational power, because the Google Colab limits the GPU usage for each of the users. The among of GPU resource isn't able to support us to adjust parameter and training many models. We only have one GTX 1070 as the local GPU. Training model with large batch(32+) size will cause the memory error. This makes the training process slow. If we get more GPU resources, we might be able to try more models (like TResNet - XL) and make the accuracy higher.

7 Acknowledging section

We splitted tasks equally in each of the phases of our project. All of us complete the tasks on time. There is not uneven contribution happened in our group.

References

Mehmet KUNDURACI and Humar Kahramanli. Vehicle brand detection using deep learning algorithms. *International Journal of Applied Mathematics Electronics and Computers*, 7:70–74, 09 2019. doi: 10.18100/ijamec.578497.

Guy Lansley. Cars and socio-economics: understanding neighbourhood variations in car characteristics from administrative data. *Regional Studies, Regional Science*, 3(1):264–285, 2016. doi: 10.1080/21681376.2016.1177466. URL `https://doi.org/10.1080/21681376.2016.1177466`.

Tal Ridnik, Hussam Lawen, Asaf Noy, Emanuel Ben Baruch, Gilad Sharir, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1400–1409, 2021.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.