Assignment 03

By:

Mahdi Farrokhi Maleki 30221603

Sina Ziaee 30216257

Data:

Our data consist of 8216 pictures of cars with 100 labels. Around 4080 of them are in the 'test' section which does not have a label.

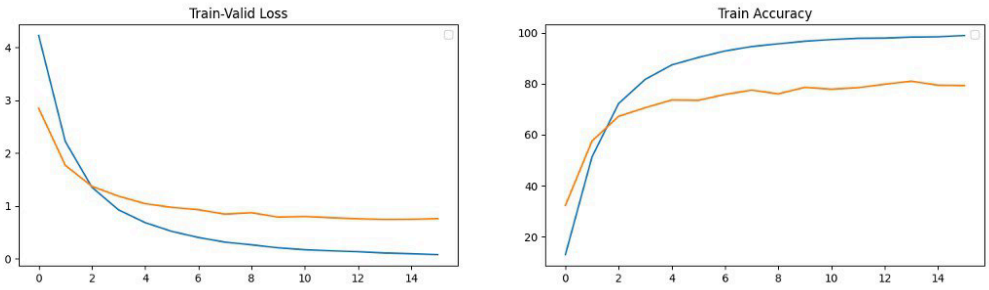
In our project, to have a validation section that can validate our trained model, we split the remaining images into 'train' and 'validation' sections from the training images in 80%-20% split. we have 827 images for validation and 3308 images for training.

Model:

For our project, we tried to use different pre-trained models in PyTorch library. We used several models such as GoogleNet, ResNet18, Resnet 34, Resnet50, Resnet151, and EfficientNet_v2_s, EfficientNet_v2_m, Mobilenet_V2. At the end we used and ensumble of all of these models in total.

Our approach to avoid overfitting:

Obviously, after a while the models tend to overfit:



Blue is the training and yellow is the validation.

We trained each of the above models individually and then used an ensumble of all of the to predict. In the process of training the models, we tracked the validation loss for each epoch and if the validation loss was smaller than the minimum loss, then we save the model, change the minimum loss to this new loss. So, the model may be trained for a while, but if the validation loss is not going to be decreased anymore, than the model is not going to be saved for testing purposes.

```
        if best_valid_loss > valid_loss:
            best_valid_loss = valid_loss
            torch.save(model.state_dict(), f'{results_folder}/best_model.pt')
            print('SAVED-MODEL')
```

In addition, to delay the occurrence of overfitting, we added augmentation to images so that the model can learn a different version of an image in each epoch. Below, you can see some of the transformations we used for our dataset. In each run, we adjusted parameters like the possibility of each transformation occurrence so that we could gain better results.

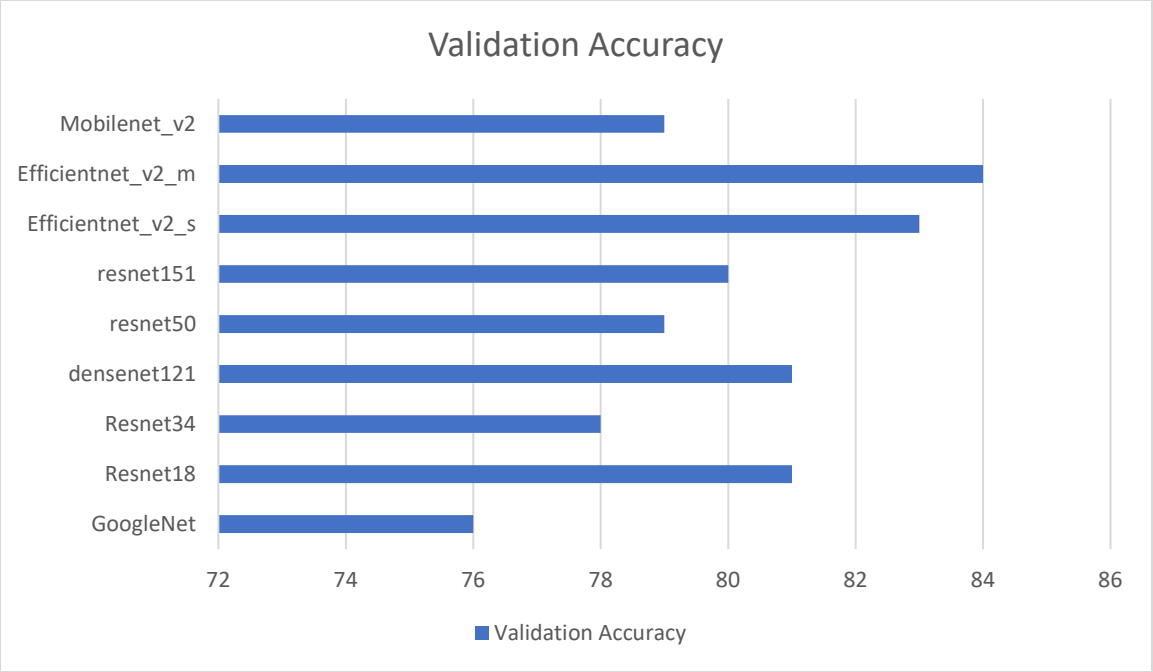
```
train_transformer = transforms.Compose([transforms.Resize((img_size, img_size)),
                                transforms.RandomHorizontalFlip(),
                                transforms.RandomHorizontalFlip(),
                                transforms.RandomRotation(360),
                                        #  transforms.ColorJitter(brightness=0.2,
contrast=0.2, saturation=0, hue=0),
                               transforms.ColorJitter(brightness=0.2, contrast=0.2),
                                transforms.RandomApply([transforms.GaussianBlur(k
ernel_size=3)], p=0.2),
                                transforms.ToTensor(),
                                transforms.Normalize(mean, std)])

random_transoform = transforms.Compose([
    transforms.ColorJitter(brightness=0.3),
    transforms.RandomHorizontalFlip(0.5),
     transforms.RandomApply([transforms.RandomAffine(degrees=30,  translate=(0.1,
0.1), scale=(0.8, 1.2))], p=0.2),
    transforms.RandomApply([transforms.GaussianBlur(kernel_size=23)], p=0.2),
    transforms.RandomAdjustSharpness(sharpness_factor = 10, p = 0.2),
             transforms.RandomApply([transforms.RandomResizedCrop(size=(img_size,
img_size), scale=(0.5, 0.9))], p=0.5),
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
    ])
```

We computed mean and std of the images and then used it for normalization in the transformers. Also, one other thing that we used for having more images in our training so that we can have less

overfitting was having **2 different augmentations** of the training set and then add them up. So, we have 3308 training images, then we add each of the augmentation from above to each copy of this datasets and then we have 9924 images in total for training.

The accuracy of the single models that we used are:



To increase the accuracy of the prediction, we used an ensumble of all of these models together, and then we used majority call to get the prediction from all of the models. So, in total the **ensumble** of all of these model **perform 89+%, which is a very good improvement and the result is available in the kaggle leaderboard as the 2nd team.**

In addition, we tried different sizes for images to feed into the network from 224*224 to 400*400. It seems that the second one is going to train the model better for generalization. 400 size pictures have 1% more accuracy on validation in comparison to 224 pictures.

For Hyper Parameter tuning we used these combinations:

Metric	Values
Learning Rate	0.1, 0.01, 0.001, 0.005
Optimizer	SGD, Adam

The best combination for hyper parameters was:

Adam optimizer with **0.005 learning rate** and we added L2 Regularization

Also, it is worth noting that, to have better estimations of the results, we added **test time augmentation** to our code, so that we can have 4 different predictions from our models and have a better estimation of what the actual predictions are.