Building image recognition models on the Stanford car data set

In this report, I will analyse the Stanford car data set and build two CNN models and evaluate their performance. The aim is to build a reliable CNN model that classifies car images correctly. This may be useful for training autonomous driving models where one needs to distinguish between cars and other objects. Another use case might be in police chase situations, where advanced city camera systems can be able to distinguish the specific make, model and year of the car being chased. First, let's perform some Exploratory Data Analyses.

First, we will select 50 random classes from the 196 classes of the original train data. Then, we do a 80-20 split of the data into a train and validation data sets. Let's examine how many car images we have for each of the 50 selected classes.

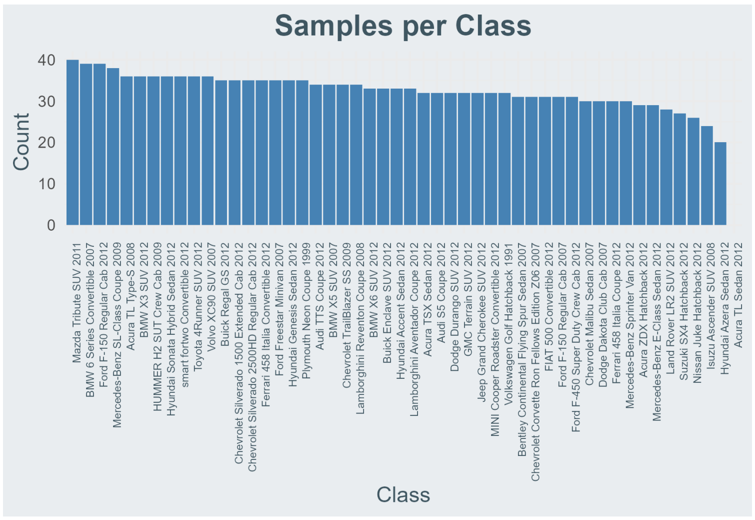

Figure 1: Number of samples for each class

We see that each class is represented relatively equally, i.e., we do not have a class that is overly presented nor under presented, which might create problems with the prediction abilities of our model. Now, let's examine the above behavior for the validation data set in comparison to our train data.

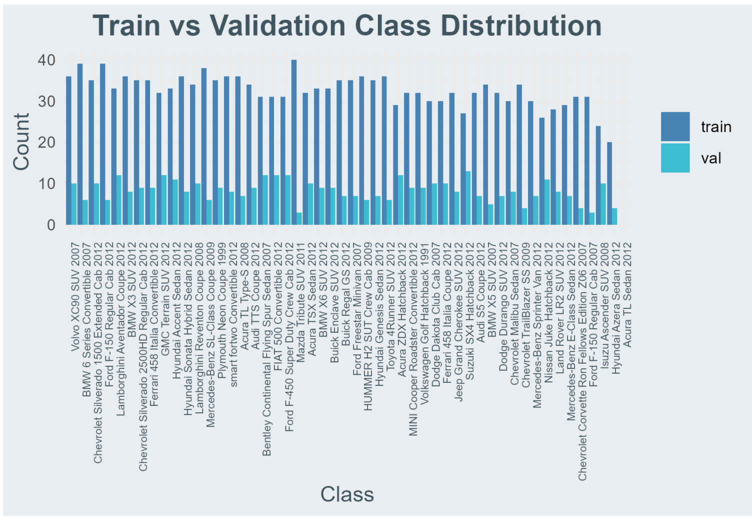

Figure 2: Number of samples for each class for both data sets

We can see that we observe the expected 80-20 split for each of the 50 classes, which is important to be able to evaluate the performance of our models.

For this data set, we will examine the performance of two CNN models - one we will build from scratch, and the other one is the pre-trained RESNET18 model. Our CNN model has the following structure

(conv1): Conv2d(3, 16, kernel_size = 3, padding = 1)
(maxpool): MaxPool2d(kernel_size = 2)
(conv2): Conv2d(16, 32, kernel_size = 3, padding = 1)
(fc1): Linear(32 * 56 * 56, 128)
(fc2): Linear(128, length(dataset$classes))

and the RESNET18 has the following structure

(conv1): Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
(bn1): BatchNorm2d(64)
(relu): ReLU(inplace=True)
(maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1)
(layer1): Sequential(
(0): BasicBlock(
(conv1): Conv2d(64, 64, 3, stride=1, padding=1, bias=False)
(bn1): BatchNorm2d(64)
(relu): ReLU(inplace=True)
(conv2): Conv2d(64, 64, 3, stride=1, padding=1, bias=False)
(bn2): BatchNorm2d(64)
)
(1): BasicBlock(. . .) - Same as above
)

(layer2): Sequential(
(0): BasicBlock(
(conv1): Conv2d(64, 128, 3, stride=2, padding=1, bias=False)
(bn1): BatchNorm2d(128)

```
(conv2): Conv2d(128, 128, 3, stride=1, padding=1, bias=False)
(bn2): BatchNorm2d(128)
(downsample): Sequential(
(0): Conv2d(64, 128, 1, stride=2, bias=False)
(1): BatchNorm2d(128)
)
)
(1): BasicBlock(...)
)

(layer3): Sequential(...) - With 256 filters

(layer4): Sequential(...) - With 512 filters

(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=512, out_features=1000, bias=True)
```

We will train each model for 10 epochs using an Adam optimizer with learning rate of 0.001. Below you can find the plots of the losses and accuracies of the CNN model

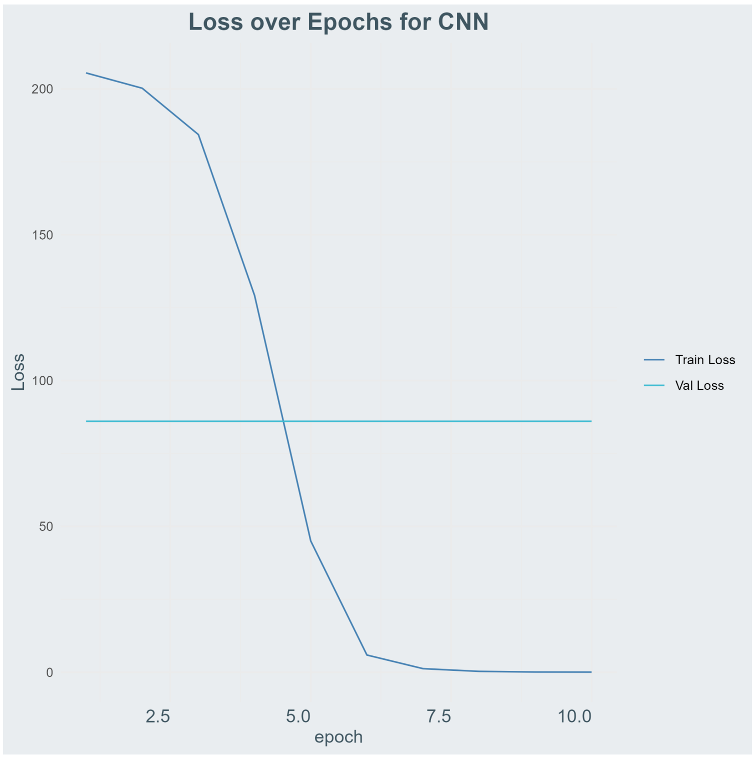

Figure 3: Values of loss

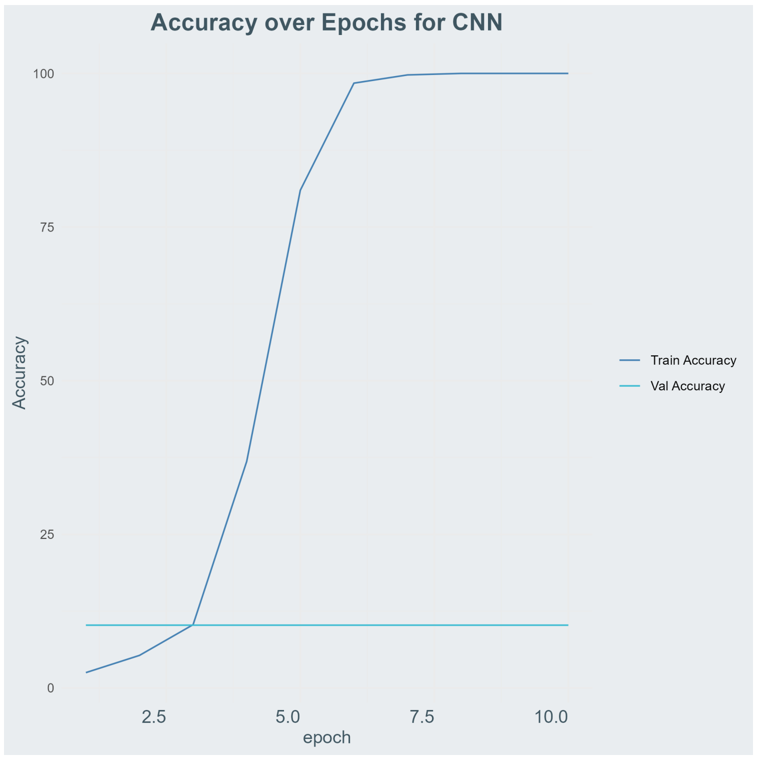

Figure 4: Values of accuracy

From the plots above, we can see that our CNN model performs really poorly at classifying the car images correctly. We see a good performance of the train loss. However, the validation cost is relatively high. Moreover, a validation accuracy of around 10% suggests a very poor performance which cannot be relied on. Still, this model is performing better than random guessing (accuracy should be 1/50 * 100, so approximately 2%). Therefore, some learning is happening.

Now let's examine the losses and accuracies of the RESNET18 model

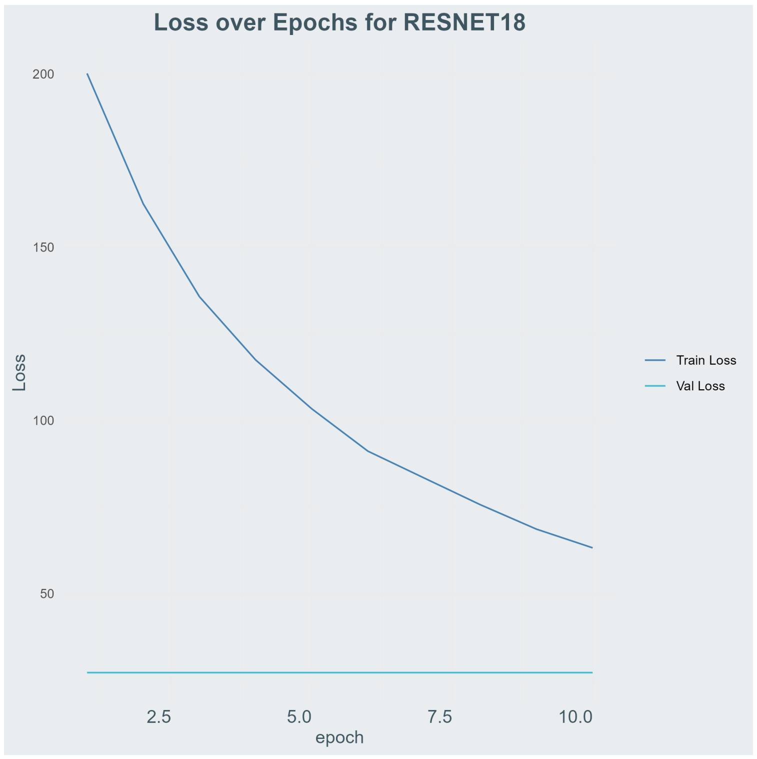

Figure 5: Values of loss

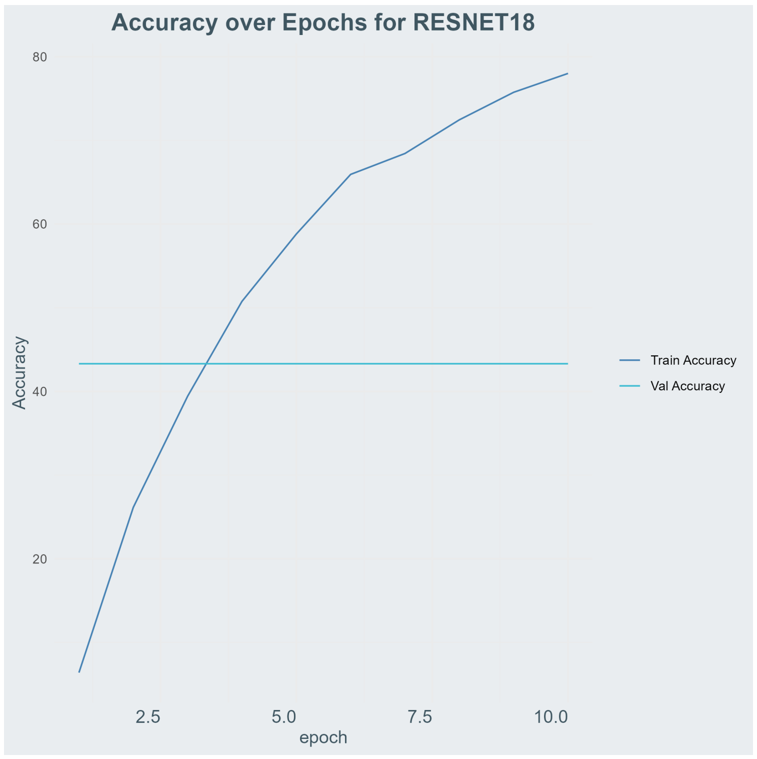

Figure 6: Values of accuracy

From the plots above, we can see that the RESNET18 model performs much better than our simple CNN model - we achieve a validation accuracy of around 43%, which is much better than before. However, this is still not high enough to have confidence that the model will perform well. Finally, let's compare the values of other performance metrics like precision, recall and F1 score.

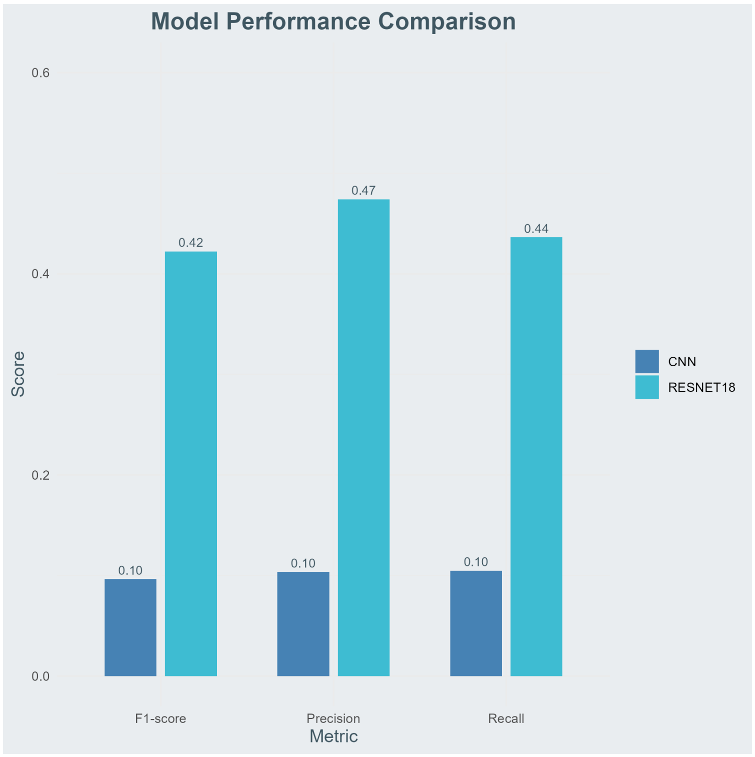

Figure 7: Values of precision, recall, and F1 score

We see again that the RESNET18 model performs much better than our CNN model. In conclusion, the Stanford car data set is a very complex data set that requires CNN models with very high capacity. This is why the RESNET18 model performs much better than the simple CNN model we built. Moreover, because of the complexity of the data, more training epochs would be recommended when someone builds a high capacity model and training on all 196 classes would be recommended.