Samuel Bernsen

Assignment 1 Report: Car Body Style Classification

Introduction

This project focuses on classifying images of different car body styles (i.e. Sedans, Hatchbacks, Convertibles, etc.) using transfer learning. A practical application of this computer vision classifier is in policing, more specifically automatic suspicious vehicle identification. This work could be extended to classify vehicles by make and model as well, although that would probably be less successful because body styles are generally more consistent than manufacture styles. I worked on this task because I am interested in cars and want to see how a computer would classify oddly shaped vehicles. To classify 7 different kinds of cars, I implemented a deep convolutional neural network modified from the VGG16 architecture. Deep learning is required because image classification is contemporarily achieved by finding feature maps that correspond to different kinds of images. The simple alternative would be to use each individual pixel in each RGB channel as a predictor for the image type and use a multi-class classification algorithm (such as decision trees or densely connected neural networks). These techniques discard important spatial relationships present in images and are thus less suitable for computer vision tasks.

Data and Methods

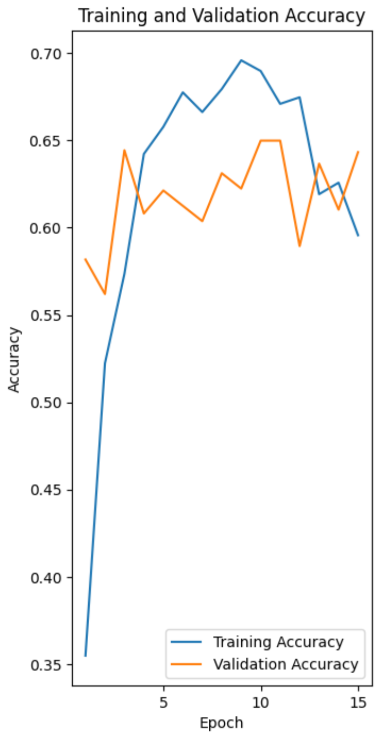
The data used for training and validation are from a Kaggle dataset with a total of 3343 images, divided mostly evenly between the classes. 305 of these images were corrupt or unusable for other reasons with TensorFlow, so I was left with 3038 images. A quick check through the folders revealed no obvious misclassifications by the dataset creator. Buggies and limousines had the fewest observations, and convertibles had the most observations. Every image was in JPEG format, and the images ranged in size dramatically.

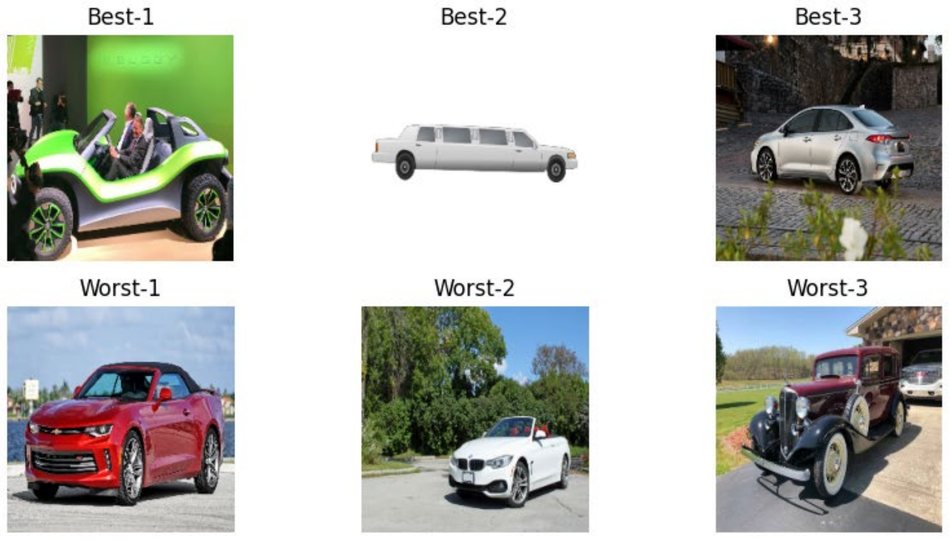
I down-sampled and standardized each image to 224x224 pixels, which may have caused underfitting issues. Since there was more information available for most images, it would have been advantageous to down-sample less, but VGG prefers images of this size. The images were preprocessed by TensorFlow for compatibility with VGG16 before performing any augmentation. For data augmentation, I added up to 6 degrees of random rotation and 5% random zoom for each image during training. Initially, I rescaled each pixel, rotated images up to 20 degrees, and zoomed up to 20%, but these efforts were counterproductive to the task. Thinking logically, zooming too much on already compressed images may leave out crucial signals on the outer parts of the images.

The model itself consists of VGG16's convolution and max pooling layers with my custom feed-forward layers at the end of the network. There are 4 densely connected FF layers, starting with 135 nodes in the first layer and ending with 7 nodes in the output layer (corresponding with 7 possible classes). Changing the number of nodes per layer and the number of layers did not significantly alter the model's performance. I added more regularization with 2 dropout layers, coefficient penalization on all but one dense layer, and implemented early stopping during training.

Results

After multiple attempts to achieve higher accuracy and lower the categorical cross-entropy, the best validation accuracy was around 58%. I am not satisfied with this metric, and as demonstrated in the attached Python notebook, 4 out of 7 test images (one for each class) were predicted correctly. When the model predicted the class from the test set, it struggled most with the coupe and hatchback while being the most confident in the limousine and buggy. The convertible was predicted correctly but with only 23.54% confidence. When I trained for less than 10 epochs, the predicted probabilities for most of the images were 31.18% regardless of the final predicted class. I investigated further and empirically found that the confidence scores deviated from 31.18% when I trained more. I am guessing that with fewer than 10 epochs, the model converged on a local minimum of the objective function and never found a better set of parameters regardless of the random initialization. I am satisfied that the model is not significantly overfit like it was without augmentation. There was about a 25% delta between the training and testing accuracy when I did not use any augmentation, which is a success. I think I found the proper regularization for my model architecture because the accuracy scores are within 5% of each other. I know that accuracy is not the only useful metric. However, it is the most important here because there are no significant consequences for predicting certain kinds of vehicles. False positives and false negatives are consequently weighed the same as both equally undesirable. For better interpretability, I also extracted the images from the validation set with the smallest and largest errors. Below are the plots of the training and validation set accuracy over 15 epochs, and the best and worst images from the validation set respectively:


Training and Validation Accuracy
Accuracy
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
5
10
15
Training Accuracy
Validation Accuracy


Best-1
Best-2
Best-3
Worst-1
Worst-2
Worst-3

Discussion

Overall, this attempt at classifying cars based on their body style was somewhat successful. I expected a higher accuracy score because I used a pre-trained, competition-winning architecture as the base. It took a lot of work to achieve above 60% validation accuracy. Without data augmentation, the model performed poorly on unseen data, never reaching above 40% accuracy on the validation set. My other regularization methods also seemed to help overall performance and generalizability, with dropout making the second largest impact on performance. Future work would include a larger dataset with more images in each class and, ideally, a model whose image height and width parameters are larger than 224x224. With more data, this architecture might work well for individual car make and model identification. The only required significant change to my current architecture would be to change the number of output nodes in the final dense FF layer.