Deep Learning Methods for Classifying Images of Automobiles by Body Style

Raafi Rahman

1. Introduction

One of the main uses of Computer Vision is in the automotive industry. The car industry is undergoing a technological revolution with the rise of eclectic vehicles, and subsequently, self-driving vehicles. Car manufacturers are racing to make the best self-driving vehicles they can, constantly using them as buzzwords and showing off future technologies at conferences. A big part of self-driving vehicles is for the vehicle to see around it, see where objects and obstacles are, and what they are. These observations are necessary for vehicles to determine the size of the obstacle, the approximate weight of the vehicle, the speed it's moving, and how long it would take it to stop to prevent a collision. We will look through a dataset from Kaggle of car images organized by the car's body type. We will create many different types of classifying models and then compare them based on their accuracies.

2. Dataset

Our dataset consists of almost 2000 images of cars. These images are divided into 6 groups, each corresponding to the body type of the vehicle pictured. The 6 categories are as follows: Multi-Purpose Vehicles (MPV), Sports Utility Vehicles (SUV), Convertibles, Pickups, Sedans, and Station Wagons. The number of images in each category varies. To have a fair dataset, we only picked the first 250 images from each category. We chose this number since

every category had at least this many, or more, images. Another problem we had was each image itself was of a different size. We needed a consistent shape among all of the images. Each of our model types also requires different preprocessing of the images.

We preprocess the dataset 3 separate times and save them as separate datasets. The first time is for our Convolutional Neural Network. The second time is for our baseline models, which consist of non-network-based machine learning methods, and our normal Dense Neural Network. The last time we preprocess the data is for us to use Transfer Learning with ResNet50. Some steps are similar for all 3 times we preprocess. First, we would load in and read each image one by one from each category using for-loops. We use the "os" library to do this. The image files were in many different types of formats. We only want to use standard image file types in our models. These file types are ".png", ".jpg", and ".jpeg". Files such as ".gif" would not work so we would filter them out. We check this by looking at the last few characters of the image's path. Next, using "OpenCV", we read the file using the "cv2.imread()" function. This is where each of the 3 preprocessing methods starts to differ.

For our Convolutional Neural Network and Transfer Learning preprocessing we simply import all the images with their default color channels. Later on, I found a problem with this. One of the images used was in grayscale already. Thus when we required the shape of (256, 256, 3), we would receive an error for having the wrong shape. Writing a short for loop that finds the discrepancy and then reshaping the data solves our problem. For our non-network-based models and Dense Neural Network models, however, we import the images as grayscale values with only a single grayscale channel. This means the images in our CNN set of data and our Transfer Learning set of data have shape (256, 256, 3), while the images in our last set of data have shape (256, 256). This brings us to our next point. All the images in every copy of the data were scaled

down to a resolution of 256 x 256 pixels. Next, the Transfer Learning data set undergoes preprocessing specific to ResNet50. This is done through the "keras.applications.resnet50.preprocess_input()" function. Now the images in all the datasets are converted into NumPy arrays. For all the datasets except the Transfer Learning one, we divide the image array by 255 to normalize it and then append it into the respective processed dataset. For the Transfer Learning dataset, however, we don't normalize and append it straight into its processed dataset. Lastly, we take the name of the folder the image came from and append that into another list that keeps track of our targets. This is because the name of the folder each image came from corresponds to the body style of the vehicle in the image.

There is still some more setup for us to do. Currently, our targets are not in a format for us to efficiently use. We make a dictionary that makes an entry for each distinct value in the targets. We also make a helper function called "get_key()". This way we can easily reference the actual and predicted target by name. We partition the datasets into training, validation, and testing subsets. Finally, we convert them all into NumPy arrays are we're good to start modeling.

3. Analysis

We look through a variety of different machine learning classification models. We start by taking baselines with non-network-based machine learning models. Then we move on to a basic Dense Neural Network. After that, we construct a Convolutional Neural Network. Lastly, we use Transfer Learning to implement a pre-trained model into our model.

3.1 Logistic Regression & Random Forest Classification

Before we start experimenting with our neural networks, I wanted to create a baseline that we can compare our neural networks to. We use two non-network-based methods to classify our images. The first is Logistic Regression and the second is Random Forest.

We create a standard Logistic Regression model to fit our data. We also included L2 regularization to help prevent overfitting. We fit out data and receive the highest accuracy of 0.218. Not very impressive. Logistic Regression may perform well on MNIST, but this dataset is much more complex. Next, we try Random Forest. We set “n_estimators=100” and “max_features=1000”. Fitting our data returns as a model with an accuracy of 0.237. This is a very minimal increase over Logistic Regression. Overall, these models give us lackluster results. These accuracies can definitely be improved on.

3.2 Dense Neural Network

Our Dense Neural Network consisted of a flatten layer that takes 65,536 inputs, 1 for each pixel of our image, then 5 hidden layers all using the ReLU activation function, and then 1 output layer utilizing softmax. The model summary is as follows…

```
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten (Flatten)            (None, 65536)             0
 dense (Dense)                (None, 1024)              67109888
 dense_1 (Dense)              (None, 512)               524800
 dense_2 (Dense)              (None, 256)               131328
 dense_3 (Dense)              (None, 128)               32896
 dense_4 (Dense)              (None, 64)                8256
 dense_5 (Dense)              (None, 6)                 390
=================================================================
```

```
Total params: 67,807,558
Trainable params: 67,807,558
```

We train the model for 20 epochs using the “Adam” optimizer. Our loss function is sparse categorical cross-entropy since we are doing a classification problem with 6 categories. After our training, we receive an accuracy of 0.189. This model performed very poorly. Random Forest and even Logistic Regression both performed better than our Dense Neural Network. Regular dense networks, however, are not well equipped to handle images. Next, we will try a model more suited for the task.

3.3 Convolutional Neural Network

A major use of Convolutional Neural Networks is computer vision. We should expect much better performance from this model than any of the ones we spoke about previously. Our CNN consists of 3 convolutional layers. Each convolutional layer has a “kernel_size=3” and “strides=2”. We use the ReLU activation function for all of these layers with “padding=SAME” to preserve the size of the image. Max Pooling with “pool_size=2” is applied after each convolutional layer. We end with a flatten layer and then a dense layer with 6 outputs, one for each possible category and the softmax activation function. We again use the Adam optimizer and sparse categorical cross-entropy loss function. Our model looks like the following…

```
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 128, 128, 64)      1792
 max_pooling2d (MaxPooling2D) (None, 64, 64, 64)        0
 conv2d_1 (Conv2D)            (None, 32, 32, 128)       73856
 max_pooling2d_1 (MaxPooling2D) (None, 16, 16, 128)     0
 conv2d_2 (Conv2D)            (None, 8, 8, 256)         295168
 max_pooling2d_2 (MaxPooling2D) (None, 4, 4, 256)       0
```

```
 flatten_1 (Flatten)            (None, 4096)              0
 dense_6 (Dense)                (None, 6)                 24582
=================================================================
Total params: 395,398
Trainable params: 395,398
```

After training the model for 20 epochs, we end up with an accuracy of 0.309. Definitely an increase from our previous models. A major obstacle in making a convolutional neural network that performs well is the training time. I had to make many compromises in the model in order to have a half-decent model that trained in a reasonable amount of time. We will look at ways to circumnavigate this in the next section.

3.4 Transfer Learning with ResNet50

Transfer Learning is a powerful tool in the arsenal of a anyone using Machine Learning. Transfer Learning is a method where you use a pre-trained model as part of your model and apply it to your specific task. This greatly helps lower training time without losing model performance. Often times you will end up with a much better model. This was indeed the case for us.

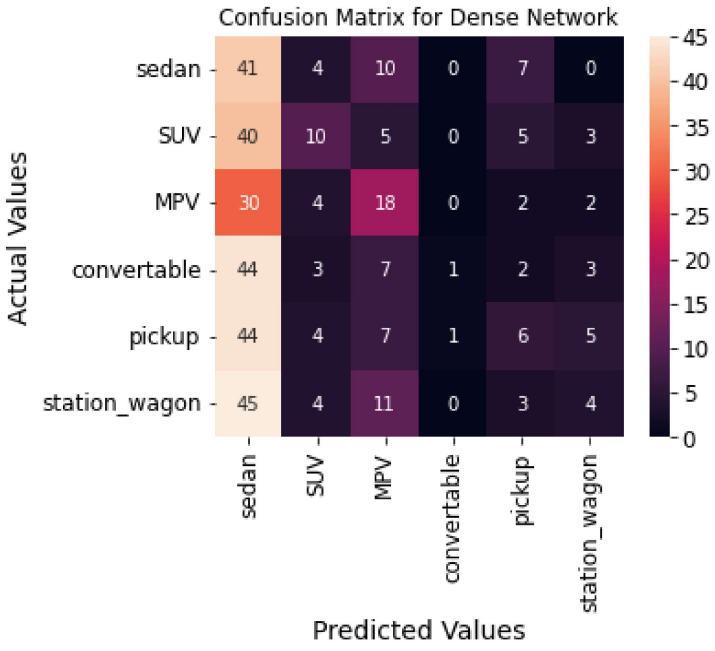
The model we chose to use is ResNet50. ResNet50 is a 50-layer Convolutional Residual Neural Network. It entered and won the Intel ImageNet competition in 2015. We load in the model and make sure to freeze the layers. We want to save the parameters so we can benefit from them in our model. Training all those layers would take a really long time and we would lose the benefit of using Transfer Learning. We add a dense output layer at the end with 6 nodes and the softmax activation function. The model is too long to paste the summary on here so I will only

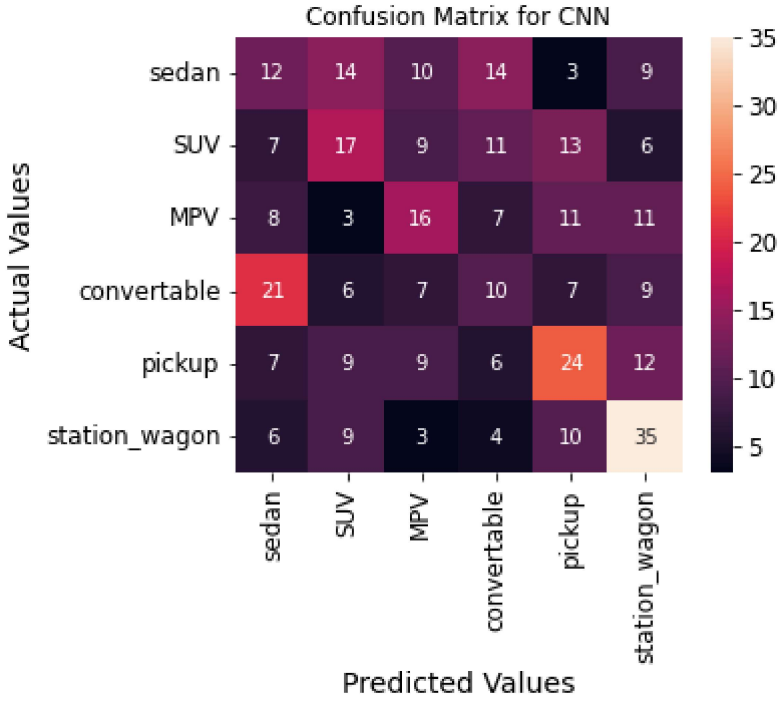
include important information about the model. ResNet50 itself includes 50 layers. Adding our output layer should make our model 51 layers deep. We note the parameters below…

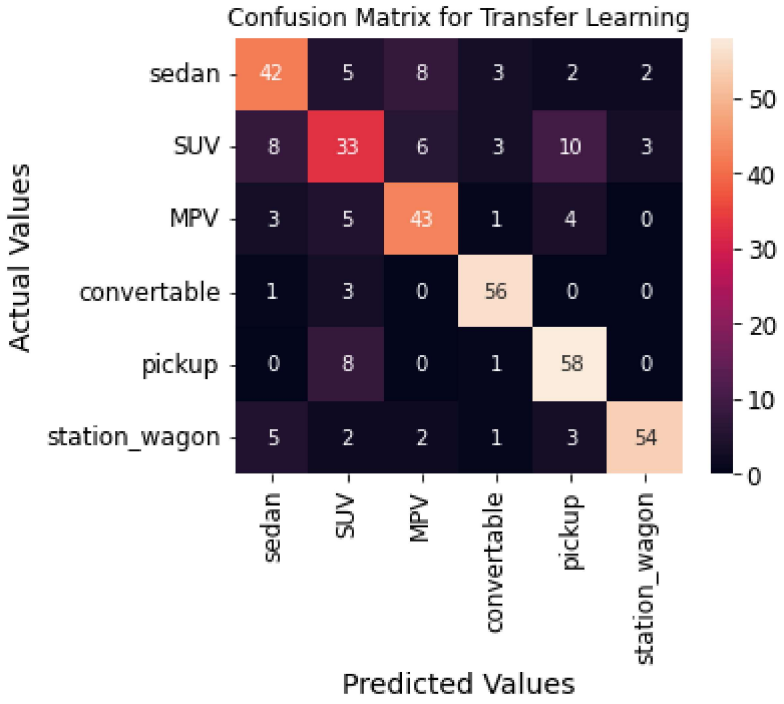
```
Total params: 23,600,006
Trainable params: 12,294
Non-trainable params: 23,587,712
```

We can see this is the only model on this list that has a non-zero amount of non-trainable parameters. This is due to us freezing all of the layers from the pre-trained network. Again we use the sparse categorical cross-entropy loss function along with the Adam optimizer. We train our model for 20 epochs and end up with a test accuracy of 0.763. This is the highest accuracy we achieved so far by a lot. That is more than double the accuracy we received from the Convolutional Neural Network we created.

4. Plots and Tables




Confusion Matrix for CNN
sedan
SUV
MPV
convertable
pickup
station_wagon
12 14 10 14 3 9
7 17 9 11 13 6
8 3 16 7 11 11
21 6 7 10 7 9
7 9 9 6 24 12
6 9 3 4 10 35
Actual Values
Predicted Values
35
30
25
20
15
10
5


Confusion Matrix for Transfer Learning
sedan
SUV
MPV
convertable
pickup
station_wagon
42 5 8 3 2 2
8 33 6 3 10 3
3 5 43 1 4 0
1 3 0 56 0 0
0 8 0 1 58 0
5 2 2 1 3 54
Actual Values
Predicted Values
50
40
30
20
10
0

5. Conclusion

Our task was to take images of cars and determine what their body style was using a machine learning model. We employed multiple types of models to tackle this issue. We can see the order of the models are as follows from worst to best: Dense Neural Network, Logistic Regression, Random Forest, Convolutional Neural Network, and Transfer Learning with ResNet50.

Out of the network-based models, it was expected that the Dense Neural Network would perform the worst, as it is not well equipped to handle image data. We can see the model often predicted “sedan” for most images. It was unable to distinguish what features made each car its body style. Our Convolutional Neural Network performed better. It still had a large number of mistakes, but with more layers and more training time, I believe it can be improved. Our ResNet50-based model performed very well. There were very few mistakes. Most of the mistakes made, I believe, are “justified” mistakes. After reviewing the data, I noticed something interesting. Hatchbacks are included under “sedans”. In the real world, I believe hatchbacks deserve, and are often placed in, their own category due to the major visual difference between them and sedans. Depending on the size of the hatchback, it can easily be confused for an SUV or station wagon, even by a human! There was one example where I had to look up what the car’s body type was because even I was confused, thinking a hatchback sedan was an SUV. It was an image of a hatchback version of a Mazda 3. Another common mistake was confusing pickups for SUVs and SUVs for pickups. Some SUVs such as the Cadillac Escalade come in both SUV and pickup (Cadillac Escalade EXT) forms. Cars like these, only viewed from the front, can easily be mistaken for each other.