DSC 672: Predictive Analytics Capstone

Milestone 3 - Intermediate Results

Stanford Car Dataset: Vehicle Recognition Project

Alexandre Girault (Team Manager)
Chris Shaffer
Sean Sungil Kim
Ryoh Shinohara
Yanxi Cai

Spring 2019

Introduction

In working towards Milestone 3, the team attempted to produce initial iterations of image classification models to classify car make and model after dedicating extensive amount of time setting up the necessary environment for analysis. The team is split up into two main groups, with two members working on CNNs and three members working on non-NN models. In working with CNNs, the CNN group utilized both custom CNN structures and pre-trained CNNs (ResNet34 and Xception). For the non-NN group, members used some of the most successful algorithms found in Caruna and Niculescu-Mizil's paper discussed in Paper Discussion 2 (SVM, Random Forest, and Boosted Trees).

Due to the size of the dataset, it was imperative that we find a work environment where we could perform model building in an adequate time frame. Memory was a major issue in both groups, requiring most members to utilize AWS services. While utilizing EC2 instances provided significant boost in performance, setting up the instances took a substantial amount of time for most users due to the lack of experience in using cloud services. This caused delays in generating our initial models especially for the non-NN members.

Boosted Trees and Random Forest

Boosted trees and random forests were one of the most successful algorithms in the paper from Paper Discussion 2, with both algorithms scoring well in image classification tasks. Therefore, we chose these two decision tree-based ensemble learners to compare performances with the CNNs. For boosted trees, we chose XGBoost due to its speed and its successes in image classification competitions prior to the arrival of CNNs. While time constraints prevented us from generating random forest models for this milestone,

As a default initial run of the models, we used the entire training set to see what the performance would be like using the models without years as the target labels (189 classes). Because the linear regression of the aspect ratio generated a parameter around 0.5, an aspect ratio of 2:1 was used. As a starting point, all images were resized to 100x50 pixel images. Images with vastly different aspect ratios were distorted, which may influence model performance. To reduce the dimensions of the image from 3D to 2D, two versions of dimensionality reductions were used: grayscaling and Canny edge detection. The Canny edge detection also provided feature extraction, which may aid in further model building.

Using these modified datasets, very basic XGBoost models were fit on the entire training set. Due to time limitations, the hyperparameters (Table 1) were set based on preexisting image classification code and modified to prioritize speed over performance, making these models have vastly inferior performance compared to the other models. Nevertheless, it will provide a sufficient starting point in comparing different feature extraction method before hyperparameter tuning.

Hyperpameters	Description	Value
colsample_bytree	Subsample ratio of attributes when constructing each tree	0.3
learning_rate	Rate at which weights are adjusted	0.1
max_depth	Max depth of trees	3
alpha	L1 regularization	10

Figure 1: Hyperparamters for XGBoost

The predictions using the models generated showed very low accuracy equivalent, although feature extraction appeared to improve model performance. While the accuracy and F1 scores of the model using grayscale images were equivalent to random guessing, the edge detected image models greatly improved performance metrics. This could further be seen in the heatmap of the confusion matrix; whereas the grayscale model generated a plot akin to white noise, the edge detected image model displayed a less random plot with a hint of a diagonal line where correct predictions are found.

Data used	Time (s)	Accuracy	F1 (macro)	F1 (micro)	F1 (weighted)
Grayscale	6539	0.0050	0.0050	0.0050	0.0053
Edge-detected	6181	0.0251	0.0229	0.0252	0.0222

Figure 2: Testing set performances for XGBoost

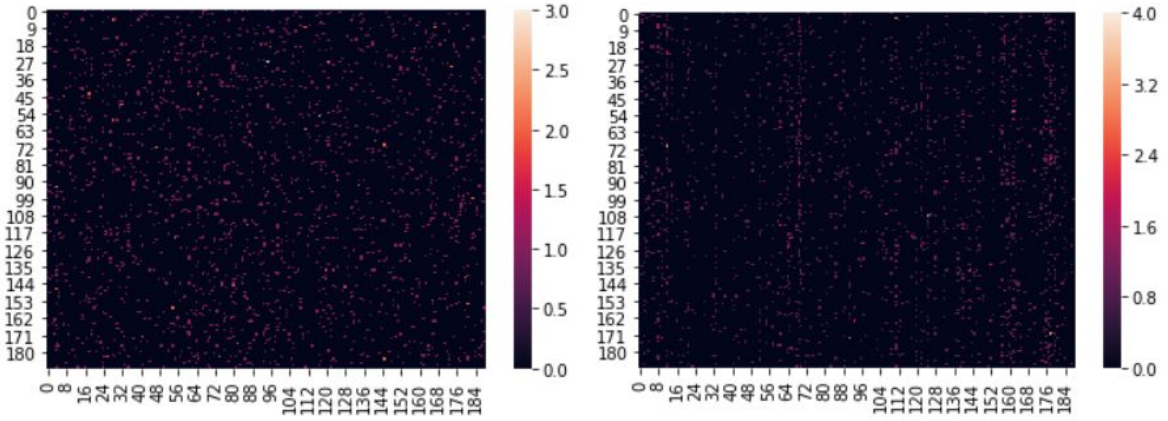

Figure 3: Confusion matrices of grayscale vs edge-detected XGB models

Support Vector Machines

Support vector machine is a supervised learning algorithm. It's a good candidate for image classification because of its capability to handle high dimensional data. SVM has several benefits over Deep Learning since sometimes SVM requires less data and it's less computationally heavy.

For the data preprocessing stage, we used the same algorithms as we did in boosted trees. Due to limited computation resources and time, we build used the default SVM model but set gamma equal to 0.001. The main hyperparameters are shown in the table below:

Hyperpameters	Description	Value
C	Penalty parameter C of the error term	1
kernel	Kernel type to be used in the algorithm	'rbf'
gamma	Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.	0.001

Figure 3: Hyperparamters for SVM

Then we used the model to fit on the test set. As a result, the highest accuracy is 0.0073, which is a bit better than the random guess, 0.0053.

Data used	Time (s)	Accuracy	F1 (macro)	F1 (micro)	F1 (weighted)
Grayscale	835.6	0.0073	0.0061	0.0073	0.0103
Edge-detected	835.8	0.0055	0.0006	0.0055	0.0010

Figure 4: Testing set performances for SVM

This initial model is definitely not ideal but it's far to draw a conclusion. We have much more future work to do to optimize the performance of SVM model. Tuning the parameters is one of the essential tasks. The parameter C allow us to control the tradeoff between accuracy and overfitting; gamma controls the stretching of data in the third dimension; we can try different kernels like 'linear' kernel. Using different feature extraction methods may also improve the performance, such as HOG.

Convolutional Neural Networks

Prior to model building, a concept called One-Cycle-Policy was researched in order to facilitate the training process, which can be found here: https://arxiv.org/abs/1803.09820. This policy essentially utilizes the learning rate and weight decay parameters to speed up the

learning phase of the convolutional neural network models. Learning rate is a parameter that controls how quickly the model learns the problem (a.k.a weight updates). It can be interpreted as the amount that the weights are updated during training (backpropagation). A small learning rate can result in slower training process but globally optimal set of weights and a large learning rate can result in faster training process but sub-optimal set of weights. Weight decay is a regularization term/parameter that penalizes big weights during the weight update phase. The best outcome is yielded when the learning rate just before the learning rate that results in the global loss minima is selected. This can be determined after examining the Loss vs. Log Learning Rate plot. Different weight decay parameter can be graphed in this plot, since the biggest weight decay results in the best outcome. Our team will experiment with this policy on our models in order to demonstrate its effectiveness.

The initial custom convolutional neural networks with 1~3 ConvNet layers (conv-conv-pool) with relu activations followed by dense(512), dropout and dense(189) fully connected layers performed poorly. All of these custom CNNs was trained after tuning the learning rate according to the One-Cycle-Policy by Leslie and fit on regular Stanford Cars images (under-sampled). The 3rd model (3 ConvNet layers) had the highest validation accuracy of about 5% post learning rate tuning. These initial set of models illustrated that simple CNN models will not be able to perform well on this complex and non-structured car image data. Since these initial models did not perform well, the outputs and graphs related to these are not documented here. From here, our team started building deeper custom CNNs and exploring other state-of-the-art architectures such as ResNet34 and Xception.

Custom convolutional neural networks (CNNs) were focused on predicting the Vehicle Make level. The following plot shows the training and validation accuracies (red and blue) along with the Top-5 accuracies (green and black). At present, the best custom CNN explored so far achieves an accuracy around 30% for determining the Make of a vehicle (49 unique classes). The top-5 accuracy scores show that the network has more success at proposing the five most-likely manufacturers instead of picking just one. It may be possible to leverage this model as a type of autoencoder for a separate downstream model to make better classifications.

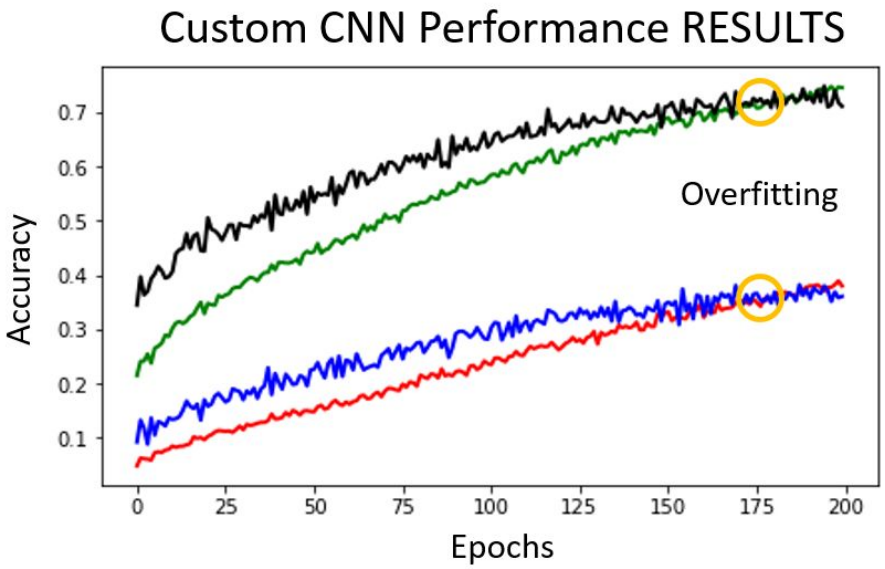

Figure 4 - Training and Validation accuracies for the best-performing custom CNN. Train/test accuracies are shown as red and blue, respectively, while the corresponding top-5 accuracies are shown in green and black.

This particular CNN uses an Adagrad optimizer (learning rate = 0.0015) and categorical cross-entropy as training parameters. A dropout rate of 65% is implemented between dense layers to help reduce overfitting. Batch normalization is applied after convolution layers in order to help speed up training and allow an infinite positive-negative range for weights to be evaluated in. Linear activation is used to prevent further degradation or loss of information.

Some model-specific image preprocessing is done prior to training. The model is trained in batches using augmented images since it is believed that there may not be enough data for the large number of target classes proposed. Image augmentation has greatly improved the rate at which models converge on a solution and the maximum accuracy overall. Overfitting still remains an issue, as can be seen in the previous plotted figure. The model begins losing utility after 175 epochs, at which point it is better able at classifying training data over novel inputs.

After the custom CNN model building phase, the Xception and ResNet34 architectures were explored on the Stanford Cars data. The state-of-the-art CNN models comes pre-trained on ImageNet data and come only with the weights for the Convolution layers. These models are trained on large visual databases, which are often used in object recognition. Since these pre-trained models are trained on large visual databases, the first step of implementing these models are adding custom fully connected layers. Once the top layers (FC layer) are added, the model has to be frozen (setting the convolution layers' trainable status to False) in order to only train the top layers. This process makes these pre-trained models suitable for any image data. After the top layers are trained for a small epochs, model has to be unfrozen to begin the actual learning. For both Xception and ResNet34 models, this initial setup was performed. In addition, Adam optimizer was used for both since it performs faster than other optimizers.

For the Xception model, the initial setup phase was done by hand in Python. The fully connected layers of the Xception model had GlobalAveragePooling2D, Dense(2048) with reul, Dropout(0.4) and Dense(189) with softmax layers. With a slight augmentation process performed on the training images using ImageDataGenerator in keras and the learning rate tuning, the Xception model was able to achieve a little above 30% validation accuracy. Due to the setup process, the learning rate was tuned twice, once before and after the training of the FC layers. There was a big RAM and kernel error issue, so the model was saved after 10 epochs of training and loaded again each time.

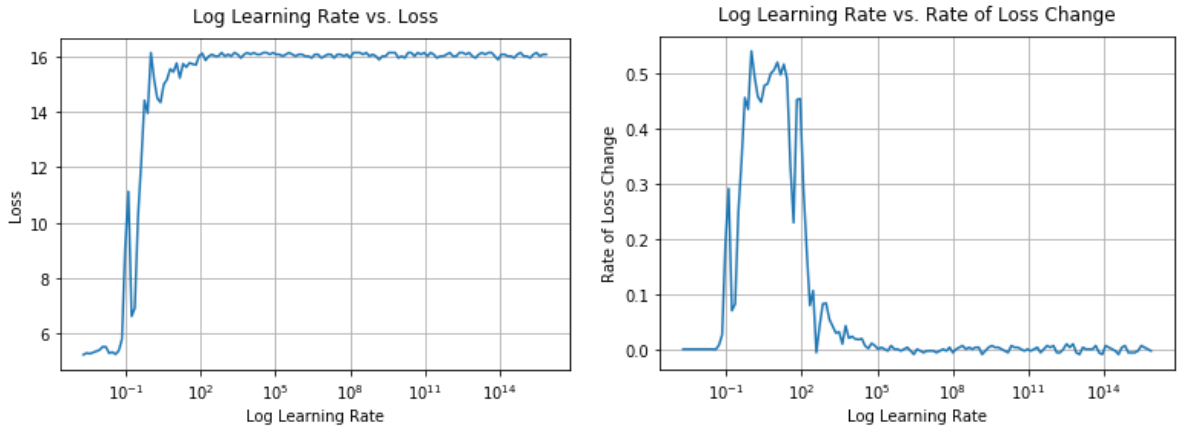

Figure 5 - Log Learning Rate vs Loss and Rate of Loss Graph Before Training the FC Layers for the Xception Model.

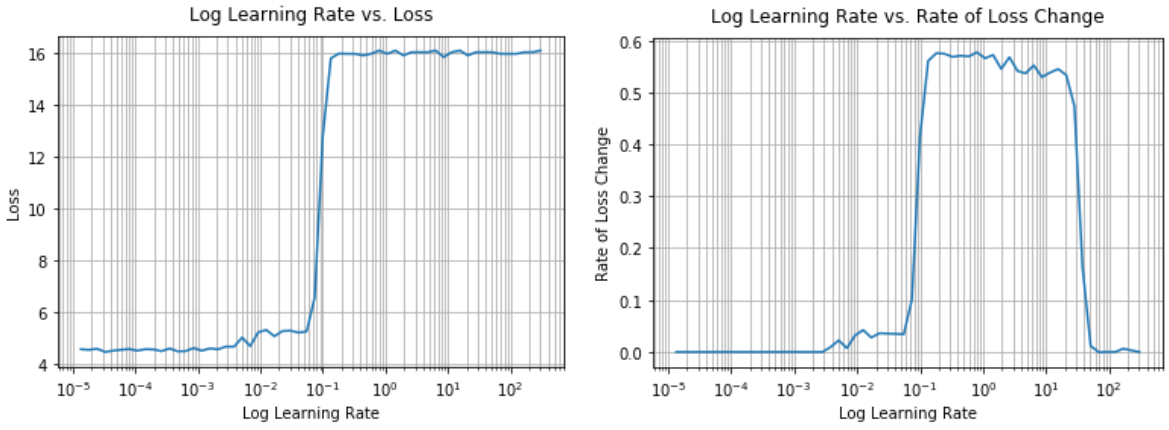

Figure 6 - Log Learning Rate vs Loss and Rate of Loss Graph After Training the FC Layers for the Xception Model.

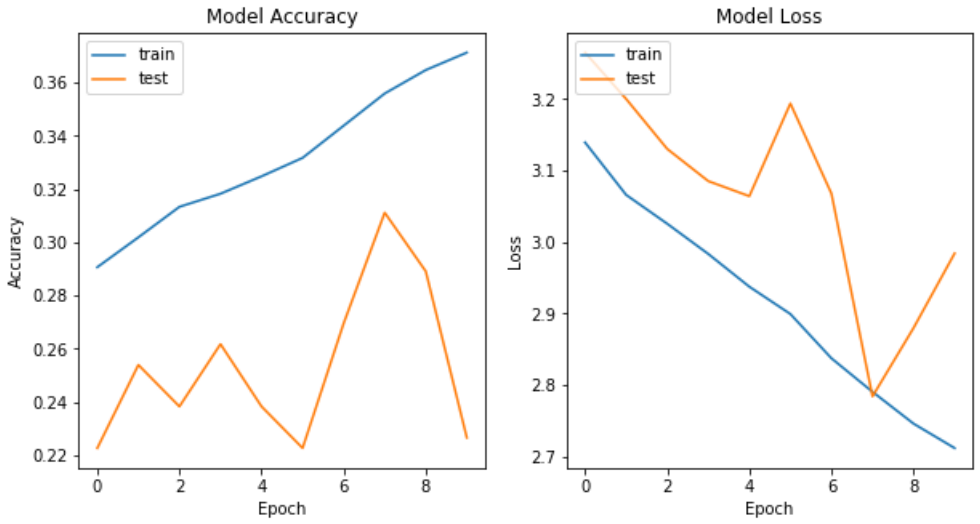

Figure 7 - Xception Model Performance Graph During 20~30 Epochs.

The initial setup for the ResNet34 model was done utilizing the fastai module. The fastai module does this freezing and adding of the FC layers automatically when loading the pre-trained models. These FC layers were larger than the one added to the Xception model, which had multiple BatchNorms, Dropouts and Dense layer with linear and relu activations. The same learning rate tuning process was performed, but different weight decay values were explored for the ResNet34 model. Utilizing the One-Cycle-Policy, the ResNet34 model was able to achieve 67.9% validation accuracy after just 7 epochs. This model building phase proved the effectiveness of the One-Cycle-Policy in terms of both performance and time costs.

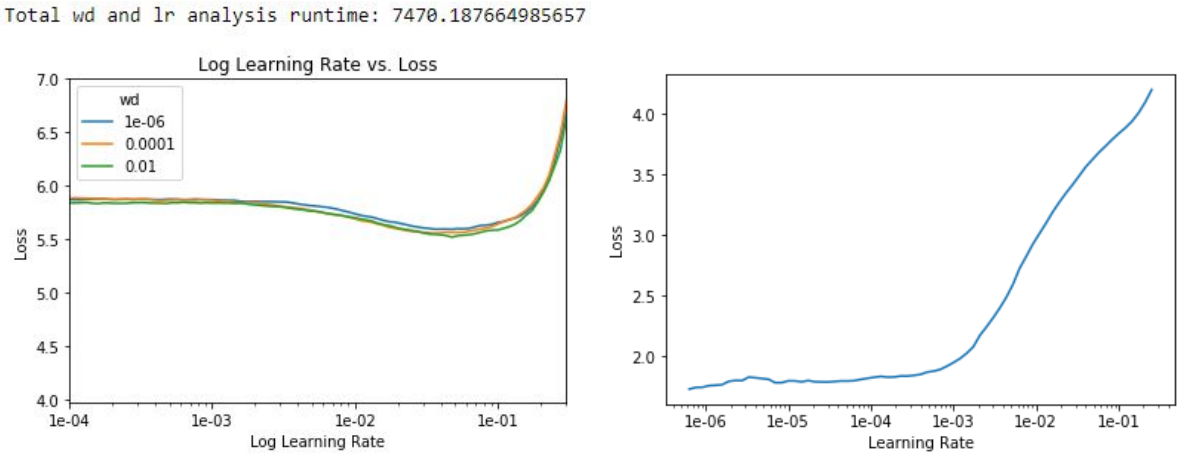

Figure 8 - Log Learning Rate vs Loss Graph Before and After Training the FC Layers for the ResNet34 Model.

epoch	train_loss	valid_loss	accuracy	time
0	1.808390	1.410663	0.632243	49:07
1	1.790068	1.385608	0.638939	50:36
2	1.781615	1.342545	0.654133	48:59
3	1.734212	1.309546	0.659284	49:18
4	1.619155	1.276190	0.664692	48:52
5	1.619910	1.250240	0.673191	49:28
6	1.594307	1.232061	0.673963	49:26
7	1.550314	1.222225	0.679372	49:52
8	1.552493	1.218435	0.675766	49:52
9	1.542795	1.218609	0.678342	49:56

```
The precision score is 0.705351568029419
The recall score is 0.6783414885397888
The fscore score is 0.6782396812708604
```

Figure 9 - ResNet34 Model Performance Table.

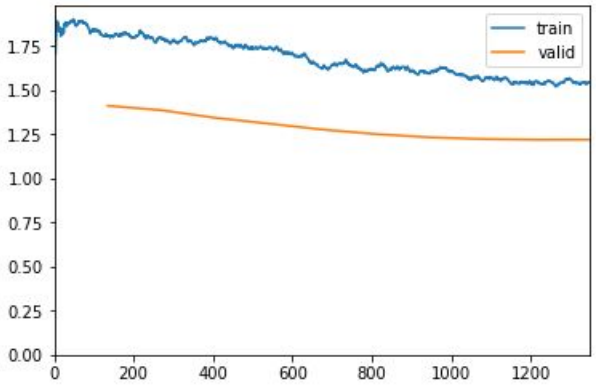

Figure 10 - Loss vs Iterations Graph for the ResNet34 Model.

The ResNet34 model is the best performing model at this moment. Further training of this model will be performed in the future, in addition to exploring with different learning rates at different phases. Some of the most confused (mis-classified) images and areas that the ResNet34 model focuses on during classification are portrayed below. Further analysis will be required in order to examine how to build better performing models.

```
[('Bentley Continental GT Coupe', 'Bentley Continental Flying Spur Sedan', 19),
 ('GMC Savana Van', 'Chevrolet Express Van', 13),
 ('Audi S5 Coupe', 'Audi A5 Coupe', 7),
 ('Volkswagen Golf Hatchback', 'Volvo 240 Sedan', 7),
 ('Dodge Sprinter Cargo Van', 'Mercedes-Benz Sprinter Van', 6),
 ('Aston Martin V8 Vantage Coupe', 'Aston Martin V8 Vantage Convertible', 5),
 ('Audi S4 Sedan', 'Audi A5 Coupe', 5),
 ('Audi S4 Sedan', 'Audi S6 Sedan', 5),
 ('Audi S5 Coupe', 'Audi S5 Convertible', 5),
 ('Chevrolet Avalanche Crew Cab', 'Chevrolet Tahoe Hybrid SUV', 5),
 ('Chevrolet Silverado 1500 Hybrid Crew Cab',
  'Chevrolet Silverado 1500 Extended Cab',
  5),
 ('Dodge Durango SUV', 'Chrysler Aspen SUV', 5),
 ('Dodge Ram Pickup 3500 Crew Cab', 'Dodge Ram Pickup 3500 Quad Cab', 5),
 ('Ford F-150 Regular Cab', 'Ford F-450 Super Duty Crew Cab', 5),
 ('GMC Savana Van', 'Chevrolet Express Cargo Van', 5),
 ('HUMMER H2 SUT Crew Cab', 'AM General Hummer SUV', 5)]
```

Figure 11 - Most Confused Images for the ResNet34 Model.

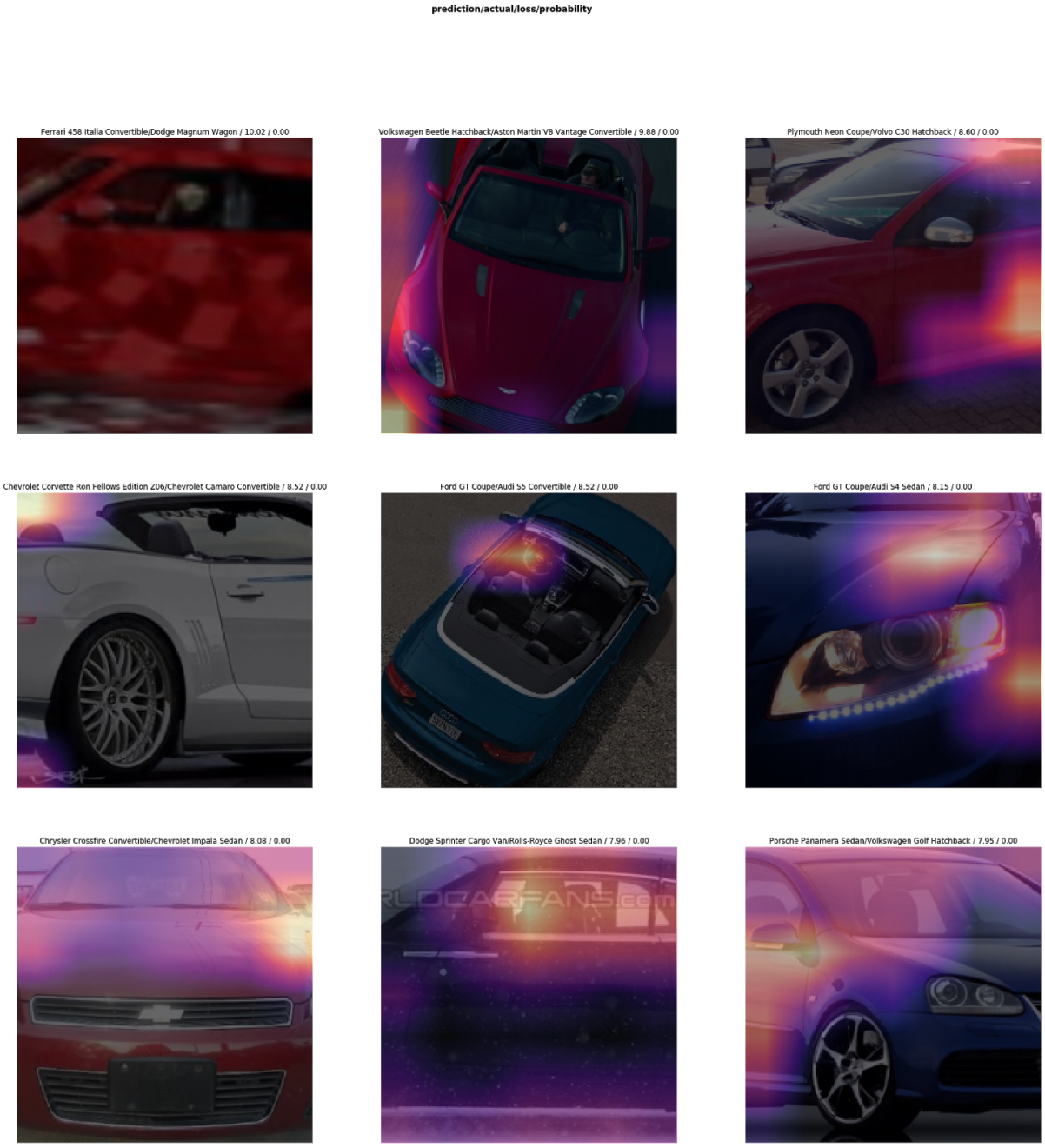

Figure 12 - Areas Focused for the Most Confused Images by the ResNet34 Model.

Future Plans

- Address overfitting
- Analyze and understand the state-of-the-art CNN model architectures
- Analyze and understand One-Cycle-Policy
- Reduce model training times
- Consider multi-model approaches
- Non-NN models: try out more feature extraction methods, start hyperparameter tuning
- Identify the most useful class information for vehicle identification (Make, Model, Make & Model, Vehicle Type, etc.)

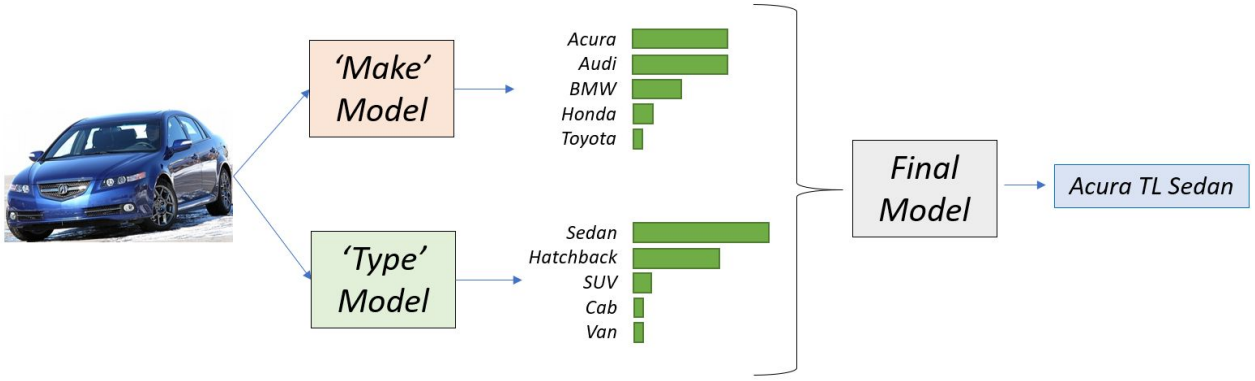

Figure 13 - Example of a proposed multi-model network for future work. Individual front-end models extract particular information which is then aggregated and processed by a final back-end model.