DSC 672: Predictive Analytics Capstone

Final Report

Stanford Car Dataset: Vehicle Recognition Project

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

DePaul University, DSC 672, Spring 2019

Executive Summary

With the advancement in image recognition algorithms, applications of image recognition have become increasingly common in our lives (Captcha, government, surveillance, etc.). One such application is for image classification tasks, which common uses included digit recognition and organism classification. In this paper, the authors present an empirical comparison of different machine learning techniques in order to build a vehicle model classification model, which could be used in various business applications such as a mobile app to identify car models based on pictures taken by users.

For this study, the team used the Stanford Car dataset - a collection of 16,185 images of cars resulting in 196 unique classes. In the exploratory phase, class labels were split to explore the characteristics of make, model, vehicle type, and model year. After analysing the statistics of the attributes, we decided to merge some classes, yielding 189 classes to use as the target for our models. Due to the success of convolution neural networks (CNNs) in image classification, we aimed to use different CNNs (custom structure, XCeption, and ResNets) in order to create a strong vehicle classification algorithm. Non-CNN classification algorithms (random forests and support vector machine) were also built in order to compare performances.

While we had expected a gap in performance between CNN and non-CNN models, the remarkable performances achieved by the ResNets were considerably better than both custom CNN architecture and non-CNN models. Compared to the sub-0.1 accuracies found in non-CNN models and the custom CNN's accuracy of 0.29, the ResNets generated accuracies as high as 0.88 (validation accuracy of 0.73 for ResNet50). The successes in the ResNet models despite the limited dataset and timeframe show great potential in further improving performance. Given more time for model tuning and by adding more images to train the models as well as adding more classes (car models), the models could eventually be utilized in various applications that may prove useful in business applications.

Keywords: Image Recognition; Machine Learning; Deep Learning; CNN;

Technical Report

Introduction

With technology development and image recognition methods being increasingly accurate, many business and government applications arise. Among them, recognition technologies are often used for security and/or tracking purposes. The Stanford Car Dataset will be utilized to build a vehicle recognition predictive model. The ultimate goal of the model is to classify a car's make and model given an input image. This model could be further developed to be used in creating a mobile application that assists users in identifying cars of interest. The users would simply take a picture of the vehicle of interest and the application would return information (Make and Model) regarding the recognized vehicle. Users could also input a picture found on the Internet or elsewhere. This application is in fact currently being used by vehicle-renting/sharing companies such as Bird and Lime. Partnerships with other car dealership websites could be beneficial in enhancing the application quality, since the recognized vehicle name would be used in searching the partners' database to obtain valuable information such as availability, price and so on. An improved model would result in direct reviews/subscription profit. This application could help people who are not familiar with cars or who simply want quick information without searching the Internet themselves. Another potential development idea of this project would be for traffic law enforcement. Traffic AI is a huge market globally. One example of this would be the China Transinfo Technology Corp. They focus on extracting features of vehicles the moment they appear in security cameras, which can help the police to track the targeted cars. Different classification algorithms such as RandomForest, SVM, some of the boosting approaches, in addition to Convolution Neural Networks will be explored. Some of the feature extraction and feature selection methods will be explored as well, for the non-deep learning classifiers. The values of these models will be quantified in terms of performance and cost. Furthermore, in order to evaluate the true performance of the model, 30 images or more will be added to the validation set to gauge the true real-world predictive power.

The Stanford Cars Dataset is a large collection of vehicle images produced by Dr. Jonathan Krause and his team at Stanford University. To generate an initial list of car labels, the authors crawled an unspecified popular car website to create a list of all cars from 1990 to 2012. Because many car models do not change their appearances across model years, the authors merged car classes with similar visual features using a technique called perceptual hashing. This technique compares two media objects such as images to see if they are different from each other. In this study, the authors used Hamming distance, the difference between two strings of numbers that represent the pictures, as a measure to determine the dissimilarities between car classes. After the initial round of perceptual hashing, the authors generated 197 classes of cars. To expand on the pool of car images, Dr. Krause and his team collected car images from Flickr, Google, and Bing. While these search engines allowed the

authors to collect many images, they needed to verify that the collected car images were from the correct car classes. To verify the identities of these images, the authors utilized Amazon Mechanical Turk (AMT) workers to annotate the car images with the correct car labels. The car identification task contained an image of the car that needed to have its identity verified, an image of the actual car from the class of interest, and an image of a car from a different class that could easily be mixed up as an image from the target class. Based on the two images with confirmed classes, the workers must decide whether the unverified car image was from the class of interest. If not, the workers annotated the image with the correct class label. For the workers to qualify for this task, they needed to pass a series of tests that contained some of the most difficult cars to identify.

To determine the quality of annotations, the authors used a technique called Get Another Label (GAL), which is an algorithm using expectation-maximum, a type of maximum-likelihood algorithm that estimates values for model parameters for incomplete data, that estimates the probability that an image is from a certain class while also determining the quality of a worker based on their correct annotations. The criteria for GAL for the car annotation task were: 1) an agreement of workers on the correct car class of an image and 2) the ability of workers to identify "gold standard" images, which were images that the authors knew the correct labels. After the GAL probabilities of a candidate image exceeded a certain threshold, the image was put into the target class. GAL was also used to further weed out poor quality workers by assigning more and more images to users that have low scores, discouraging them to continue the task. After obtaining the set of images with assigned classes, the authors utilized a different group of AMT workers to assign bounding boxes, the section of an image that contains the target object, using a technique presented by Fei-Fei et. al. To further remove duplicate images, the authors used another round of perceptual hashing on the images based on the bounding boxes, yielding a total of 16,185 images with 196 classes of cars.

Data Preprocessing & Exploratory Data Analysis

The original dataset defined a 'class' as the combination of make, model, and year. This yields 196 individual and unique classes. An example of one of these classes is shown below (Type is a subdivision in the Model level). The class levels were parsed into the components.

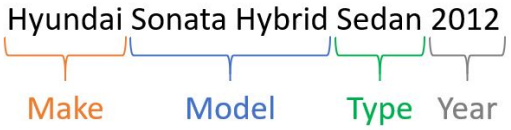


Figure 1 - Example class and separable characteristics.

The following table provides specific descriptive statistics from the entire original dataset. Due to the way that the image of this dataset was created, a thorough Exploratory Data Analysis of the original class distributions was highly desired.

	ClassNo	Class	Make	Model	Type	Year	Image Height	Image Width	Color Channels
Type	Integer	String	String	String	String	Integer	Integer	Integer	Integer
Uniques	196	196	49	177	13	16	Several	Several	3
Mean	-	-	-	-	-	2009.56	308	573	-
Std Dev	-	-	-	-	-	4.43	214	375	-

Table 1 - Descriptive Statistics for the Stanford Cars Dataset.

The dataset contained no missing values, so no imputations or data removal was required. Class labels were split to explore the unique labels in Make, Model, Type and Year. The string-formatted labels were split by a space, then categorized into Make, Model, Type and Year levels. Automating this was challenging, since vehicle make and model strings vary widely in length and character type. Due to the lack of domain knowledge, there may exist miscategorized class information; however, this extraction of class label levels were performed to the best of our abilities. Because of the high number of classes, the levels of class labels were analyzed with the hopes of reducing the total class number. Initially the class labels were analyzed by human eyes. Our group believed that there were no multiple Make+Model yearly labels after the first examination. However, this was in fact not correct, proven by the Model vs Year scatterplot per Make graph demonstrated in this link: https://rshinoha.shinyapps.io/jitter_plot_of_car_makes_across_years_1990-2012/.

While the initial expectations were to create models that classifies 196 classes of car models found in the dataset, this appeared to be not feasible in terms of computational power of most team members' devices. In addition, the authors stated in their research that vehicles with

similar visual features were merged into the same Year level. In order to reduce cost and achieve more accurate class labels, the Year level was dropped. There were a few duplicate Make+Model class labels due to this class engineering. These duplicate classes were simply fused together. This Year removal only resulted in a slight decrease in total class number, from 196 to 189.

Although the source data provided pre-split training and test sets, the 50/50 partitioning was not ideal for the purposes of this study. Therefore, the two sets of data were concatenated in order to re-generate a custom training-testing split. Due to the class imbalance issue shown below, random undersampling without replacement was used to create the final training-testing split. A random 80/20 split was performed on the concatenated Make+Model data, where the 80 was the under-sampled training set and the 20 was the imbalanced testing set (for the state-of-the-art CNN models, 70/30 was used). The testing set was kept imbalanced in order to preserve the original data integrity. This would also provide accurate true testing performance due to the maintained data integrity. Undersampling rather than oversampling (or SMOTE) was utilized because of the restrictions in terms of computing power and time costs.

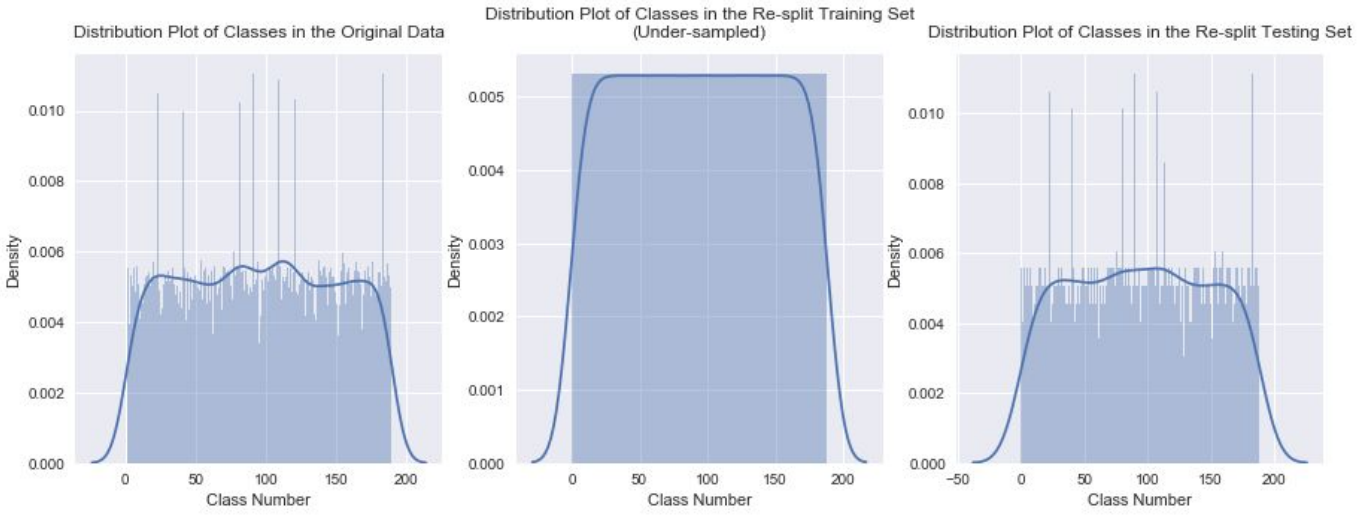


Figure 2 - Class distribution graphs comparing the original and under-sampled training/test data using the Make+Model class labels.

The Stanford Cars Dataset had images that had no apparent order or structure. There is high variability in how the photographs were taken. The angles, elevations, and rotations of the target vehicle vary widely between images, resulting in a more complex classification problem.

Preprocessing steps were dependent both on the modeling method and the available resources. The original dataset contained a collection of images with varying heights, widths, and color channels. In order to simplify problem and model complexity, images were normalized and resized to a particular height and width deemed suitable for model construction. For non-deep-learning models, images were downsized to 100x50x3. Grayscale

was also applied to the images, producing size of 100x50. For deep-learning models, an image size of 224x224x3 was selected as the standard input size. Due to the computation heavy nature of image data, all building and analysis of models were executed on AWS EC2. Depending on the computation need, the instance size was determined. From the exploratory phase, many class imbalances were revealed. Having identify those imbalances allowed to better understand the dataset and the obtained results and errors.

Methods

XGBoost

Decision tree based ensemble learners are some of the most generalizable algorithms for classification tasks. Methods like Boosted Trees and Random Forests consistently perform well in many applications including image classification, of which boosted trees such as XGBoost performing better in Caruna and Niculescu-Mizil's paper. Based on these observations, we attempted to use XGBoost as our decision tree-based ensemble learner model for this particular task. All model inputs were converted to 100x50 pixel grayscale images. While initial runs of XGBoost showed promise, performing hyperparameter tuning proved to be too computationally expensive despite using large EC2 instance (m5.2xlarge instance: vCPU: 8, memory: 32 GiB, dedicated EBS bandwidth: up to 3,500 Mbps, network performance: up to 10 Gbps); frequent memory issues and weak internet connection prevented any hyperparameter tuning iterations to run to completion. The issues arose due to the nature of boosting ensemble methods. Boosting utilizes weight systems in order to focus on difficult instances during its sequential training. Certain weak learners are prioritized during the prediction using weight systems as well. This ultimately caused memory errors, since the EC2 instance could only handle so much variables during individual runs. On the other hand, bagging simply utilizes parallel training and averaging during the prediction phase, which reduces a lot of computing cost. Therefore, we focused on using random forests as our decision tree-based ensemble learner using the same 100x50 grayscale images.

Random Forests

While better than XGBoost, the sklearn Random Forest Classifier struggled with memory issues. In order to minimize data and time loss, we used sklearn's RandomizedSearchCV() function to run a wide range of hyperparameters using 5-fold cross-validations. RandomizedSearchCV() chooses a random combination of hyperparameters based on a given set of ranges of hyperparameters. *Table 2* shows the initial hyperparameters for RandomizedSearchCV(). To further prevent data and time loss, we ran single RandomizedSearchCV() per run using different seed values.

With four running m5.2xlarge instances, we were able to run approximately 70 sets of hyperparameters. However, a subset of hyperparameters failed to run to completion due to memory issues, which suggests certain hyperparameter combinations were excluded from further analysis. *Figure 13 (Appendix)* shows a scatterplot matrix comparing the different hyperparameters with respect to the mean test scores of each 5-fold cross-validation. Based on this scatterplot matrix, we narrowed the hyperparameter ranges to use in a grid search.

Like RandomizedSearchCV(), we ran each hyperparameter combination of GridSearchCV (5-folds cross-validation) in separate lines of code in an attempt to minimize data loss. *Figure*

14 (Appendix) shows a scatterplot matrix comparing the mean test scores of 5-fold cross-validations with respect to the different max_depth and min_samples_split hyperparameters. Based on the observations of the scatterplot matrix, we chose the best hyperparameters for the final model as shown in the "Best Value" column of *Table 2*.

Parameters	RandomizedSearchCV()	GridSearchCV	Best Value
n_estimators	200, 400, 600, 800, 1000, 1200, 1400	800	800
max_features	'sqrt'	'sqrt'	'sqrt'
max_depth	10, 20, 30, 40, 50, 60, 70, 80, 90, 100	30, 40, 50, 60, 70	60
min_samples_split	2, 5, 10	2, 3, 4, 5	3
min_samples_leaf	1, 2, 4	1	1
bootstrap	True, False	False	False

Table 2 - Random Forest Hyperparameters

Support Vector Machines

Support Vector Machines (SVM) is a supervised learning technique and was popular in image classification before the adoption of deep learning. In this project, we used it as one of our traditional (non-deep-learning) algorithms. Several different feature extraction methods were applied during modeling. Before tuning the parameters, the default SVM model was used, where C = 1, gamma = 'auto', and kernel = 'rbf' to fit with different feature extraction algorithms to find an optimal combination. The testing set performance is shown below, where 'edge' means Canny edge detection, 'Gaussian' means adaptive gaussian thresholding and 'Mean' indicates adaptive mean thresholding.

Preconditioning	Accuracy	Recall	Precision	F1 Score	Time (s)
Grayscale+edge	0.0050	0.0050	0.0000	0.0001	9.03
Grayscale+edge+PCA	0.0030	0.0030	0.0023	0.0024	11.77
RGB+PCA	0.0675	0.0675	0.0735	0.0971	279.82
GrayScale+PCA	0.0599	0.0599	0.0727	0.0490	241.62
GrayScale+Gaussian+PCA	0.0358	0.0358	0.0454	0.0293	566.93
GrayScale+Mean+PCA	0.0363	0.0363	0.0468	0.0294	536.77

Table 3 - SVM Test Set Performances

As shown in *Table 3*, PCA on RGB images had the best performance. However, according to our experiment during modeling, sometimes model performance of PCA on grayscale images

could surpass that of PCA on RGB images. To eliminate the randomness, we did parameter tuning on both models, run training and testing splits 30 times, took the mean of the performance results and made comparisons. After parameter tuning, C = 10, gamma = 0.0001, kernel = 'rbf' for 'GrayScale+PCA', and C = 100, gamma = 0.00001, kernel = 'rbf' for 'RGB+PCA'. The mean of performance results is shown below:

Tuned Parameters	Accuracy	Recall	Precision	F1 Score
GrayScale + PCA	0.0657	0.0657	0.0777	0.0581
RGB + PCA	0.0856	0.0856	0.0849	0.0768

Table 4 - Effect of Image Color Scheme on Model Performance

As *Table 4* shows, color input images have a slight advantage over grayscale versions. Using this approach, experiments were performed on different sizes of image data. The graph of performance results is shown below. Since accuracy and recall are the same, only accuracy was plotted.

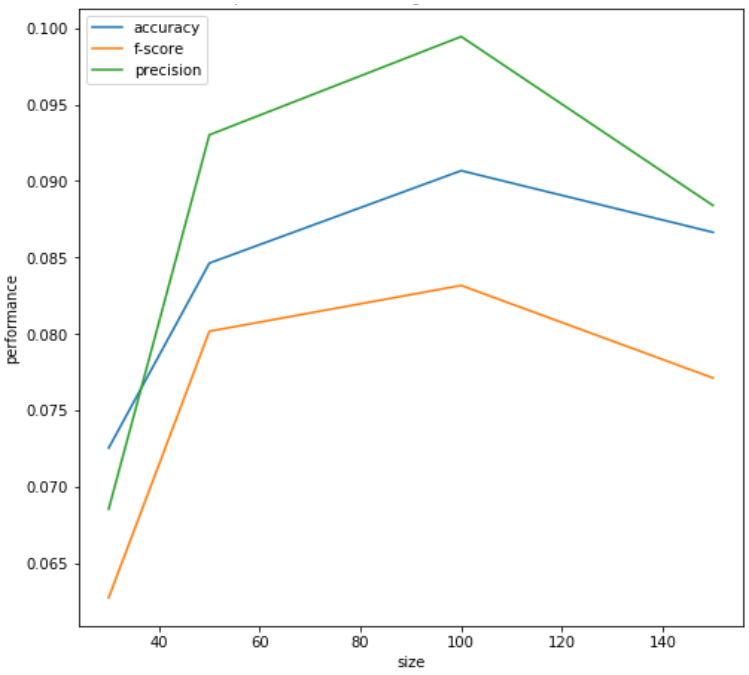


Figure 3 - Model performance on image data of different sizes.

Since downsizing is not a lossless process, it is not surprising that inputs as small as 50x25 and 30x15 resulted in worse performance. However, when the image size was decreased from 150x75 to 100x50, the performance was improved. This abnormal pattern reveals the weak capability of SVM in our non-structured image data.

Convolutional Neural Networks

Convolutional neural networks (CNNs) are an extension of neural nets adapted for image classification. The model is composed of layers of filters and compressions in sequence. Filter weights are adjusted like those in the neural net for similar purposes. With an appropriate amount of training samples and adequately-tuned parameters, a CNN will automate the feature selection process and accurately classify the input.

Prior to model building, a concept called One-Cycle-Policy was researched in order to facilitate the training process (as discussed in https://arxiv.org/abs/1803.09820). This policy essentially utilizes the cyclic approach of learning rate and weight decay parameters to speed up the learning phase of the convolutional neural network models. Learning rate 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 that results in just before 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 for analysis. Our team experimented 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 exploratory custom CNNs were trained after tuning the learning rate 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 are not able to perform well on this complex and non-structured car image data. Due to this, 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 Xception, ResNet34 and ResNet50.

The following custom architecture was inspired by the VGG16 CNN model, though it is more compact due to hardware limitations. Using several layers of filters allows the model to consider a large space of possible weights to explore. *Figure 4* shows the training history of the model. Validation accuracy (blue) approaches 30% while the top-5 validation accuracy (black) levels off near 50%. These metrics, while not wholly descriptive, provide valuable feedback during an iterative model development process.

In addition to a homegrown CNN architecture, state-of-the-art CNN models such as Xception, ResNet34 and ResNet50 were explored and analyzed. The state-of-the-art CNN models come 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. Due to this fact, the first step of implementing these models are adding custom fully connected (FC) layers. Once the top FC layers are added, the model weights are frozen in order to train only the top layers. This process makes these pre-trained models suitable for any image data of interest. After the top layers are trained for a small number of epochs, model is unfrozen to begin full learning. For all state-of-the-art models, this initial setup was performed. In addition, an Adam optimizer was used since it performs faster than other optimizers.

Image Augmentation

Due to the curse of dimensionality, a large number of images are required to produce satisfactory classifiers. In lieu of collecting more samples manually, the CNN models were trained with synthetic images. These images are slightly distorted (mirrored, rotated, and sheared to some extent) in order to help the model better generalize the input space. Consequently, this helps combat overfitting since the model is exposed to more than just the 'real' training images. Test images were left unmodified in order to evaluate models with actual image data.

Custom CNN

Filter Layers

Convolution layers contain trainable kernels (sets of weights) which transform patches of the input image. These are essentially different kinds of image filters that the model adjusts during training in order to optimize the chosen loss function. A well-trained model contains an array of filters that effectively discriminate between different classes of images. In the Keras package, convolutional layers have several user-defined inputs such as stride length and kernel dimensions. Altering these two parameters in particular can influence model learning and performance, but were not explored in depth here. The default kernel size of 3x3 was used for each models' filter layers. Max-pooling is used after each block of convolution layers. The pooling process summarizes a window of pixels into a single output value. In the case of max-pooling, the window will be condensed into the highest value. This helps emphasize and maintain areas of large contrast from layer to layer.

Deep Neural Network

During model development, larger and deeper neural networks tended to have an edge over smaller ones. The neural network model selected attempts to make use of the available resources by using a relatively large number of nodes in the hidden layers. The best performing combination of optimization algorithm and loss function proved to be Adagrad and categorical cross-entropy, respectively. The Adagrad algorithm is a gradient-descent method which has an adaptive learning rate and can compensate for class imbalances.

Dropout is utilized to increase problem-space generalization by forcing the model to consider more than the output of just a few nodes. A rate of 0.25 provided a good balance between

training time and validation accuracy. Linear activations are used between layers. This identity mapping preserves the variance of the data. The final layer applies a Softmax transformation to the final summations. This transforms the output into a set of likelihood values, the highest of which belongs to the best guess.

Development

The Keras package readily allows for two metrics to be monitored during training- accuracy and top-5 accuracy. The top-5 accuracy metric provides another level of feedback during training, since accuracy alone does not give credit to close guesses.

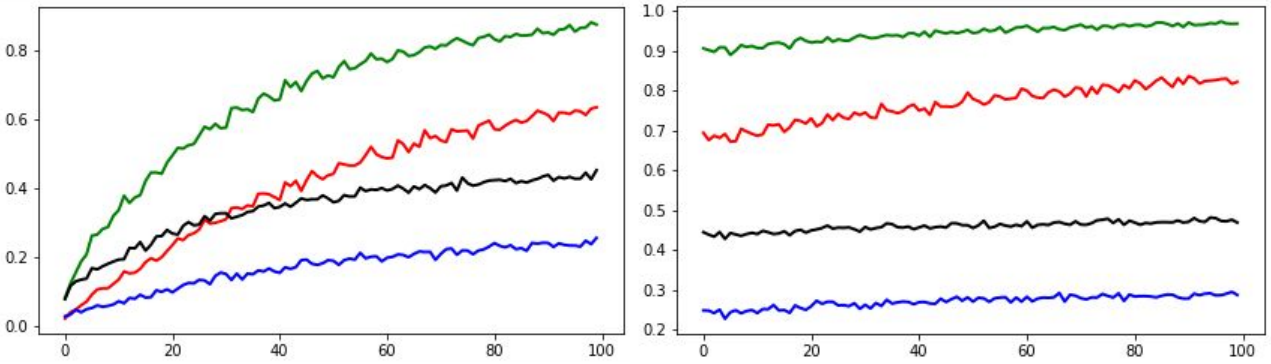


Figure 4 - Custom CNN training history. The blue and black lines show, respectively, the validation and top-5 validation accuracies as the model trains. Eventually, a cumulative penalty parameter in the Adagrad algorithm makes performance plateau by dampening the learning rate.

Xception

The fully connected layers of the Xception model had GlobalAveragePooling2D, Dense(2048) with relu, Dropout(0.4) and Dense(189) with a softmax layer. 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 once after the training of the top-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. This model resulted in extremely long computing time. Although the Xception model could have improved with further training, this model was discontinued because of the computation limitations. The outputs from the Xception model can be found in the Miscellaneous Graphs section. To combat the issue of lack of resources, models that are available in the fastai module were the highest priority, since the fastai module offers the One-Cycle-Policy. This led to the ResNet34 model.

ResNet34

The initial setup for the ResNet34 model was done utilizing the fastai module. The fastai module does the 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. They had multiple BatchNorms, Dropouts and Dense layers with linear and relu activations. The One-Cycle-Policy learning rate tuning process was performed, in addition to implementing the weight decay parameter for the ResNet34 model. This model was able to achieve 0.68 validation accuracy after just 7 epochs during the top-layer training. This proved the effectiveness of the One-Cycle-Policy in terms of time costs. Excluding the top layer training phase, this model was trained for a total of 100 epochs. The loss vs. learning rate graphs used in determining the optimal learning rate for the One-Cycle-Policy is demonstrated below. Both the top layer training and full training used the One-Cycle-Policy.

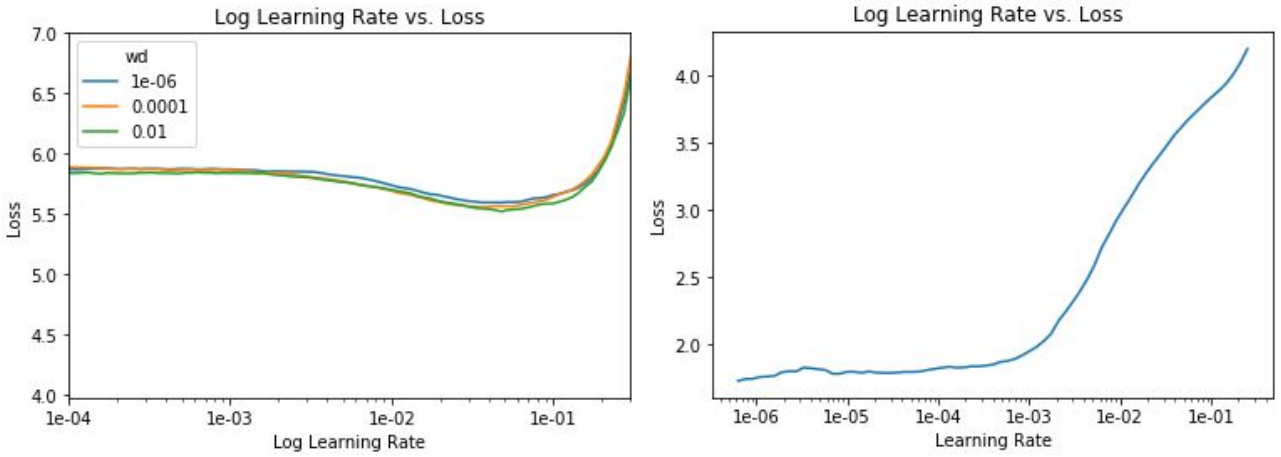
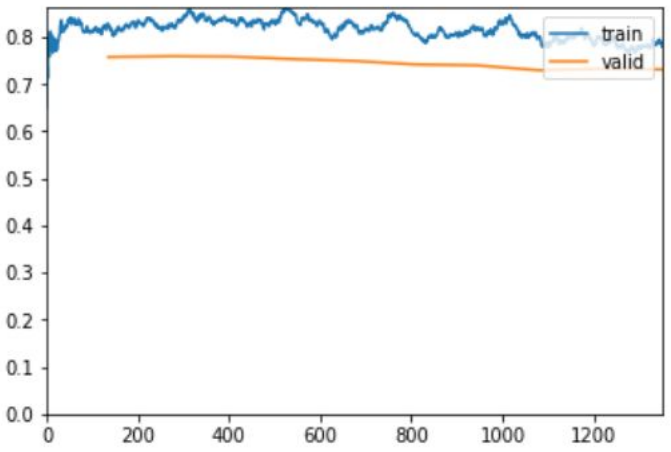


Figure 5 - ResNet34 Log Learning Rate vs Loss graph before and after training the FC layers.

Some augmentations and transformations were applied on the image data for the ResNet34 model. This helps prevent overfitting, since this process creates variability in the inputs. The parameters used for the augmentations are do_flip = True, flip_vert, False, max_rotate = 90. A preview of some of the images are illustrated in *Figure 6*.

epoch	train_loss	valid_loss	accuracy	time
0	0.828971	0.757601	0.789338	47:57
1	0.823475	0.759355	0.789338	47:17
2	0.834208	0.758517	0.786248	47:51
3	0.851897	0.753168	0.785733	47:02
4	0.821245	0.749253	0.790883	47:52
5	0.804125	0.741682	0.790111	46:54
6	0.818743	0.739885	0.791141	48:05
7	0.801630	0.729439	0.791398	47:00
8	0.798832	0.732448	0.792944	47:57
9	0.780111	0.731119	0.793974	46:53




```
The precision score is 0.8100481150660742
The recall score is 0.7939737316507854
The fscore score is 0.7940082428478917
```

Figure 6 - ResNet34 final (10th) model performance.

After 100 epochs of training, the ResNet34 model was able to obtain 0.78 training loss and 0.73 testing loss, as illustrated in the table and graph above. Compared to the previous loss vs. iteration graphs that are attached under the Miscellaneous Graphs section at the end of this paper, the gap between the training loss and the testing loss was decreasing during the last 10 epochs of training. The large gap between training and testing loss could be interpreted as the model not being able to perform similarly on the training and the testing set. Small gap between the training and the testing loss could be interpreted as the model's ability to perform similarly regardless of training/testing set it uses. In ResNet34's case, the training loss was much higher than the testing loss during the beginning stages of the learning phase. This meant that the model had room for improvement by learning more from the training data. After 100 epochs of fitting, both the training and testing loss were improved (decreased). In addition, the gap between the two loss lines also narrowed down. The training of ResNet34 was successful, since it was able to bring the training and testing loss down and increased the model accuracy. The cross and/or the divergence of the training and testing loss lines indicates overfitting. This did not occur during the 100 epochs of model fitting. There is potential room for improvement in terms of loss and accuracy scores. However, since 10 epochs take about 8 hours and there was less than 0.005 increase in model validation accuracy during the last 10 epochs, the training process was stopped after 100 epochs to save computation and time costs.

```
[('Bentley Continental GT Coupe', 'Bentley Continental Flying Spur Sedan', 19),
 ('GMC Savana Van', 'Chevrolet Express Van', 15),
 ('Dodge Sprinter Cargo Van', 'Mercedes-Benz Sprinter Van', 6)]
```

Figure 7 - ResNet34 most confused images.

Although the confusion matrix in the Appendix is not very readable, it is clear that the model performance is high based on the diagonal actual vs. predicted line. There are two spots that are not a part of this diagonal line, which are the first two lines of the most confused images summary above. According to the most confused images, the misclassified labels are in fact very similar in shape. The most confused (19 counts) image labels both belonged to the Bentley Make family. The second most confused (15 counts) image labels were both vans. This indicates that the misclassified images are not by chance, but actually due to the similarities in shape. The heatmaps in the Appendix portray images with high loss scores and highlight areas that the ResNet34 model believes that are important. This could be used to analyze why the model is classifying incorrectly more in depth. Although direct interpretation is not obtainable, it could be concluded that more variety of training images and further training will be beneficial by the heatmap images that has no important areas. The same could be said by the heatmap images that do not focus on the Make logo.

ResNet50

Since ResNet34 using One-Cycle-Policy was extremely successful in classifying the Make+Model level, ResNet50 was explored after development of the ResNet34 model. There

are a few differences between the ResNet34 and ResNet50 architectures. The main differences are the numbers and sizes of the convolution layers. As the name suggests, ResNet50 has a deeper architecture than ResNet34. As shown in the ResNet family comparison graph below, only the conv2.x ~ conv5.x are different. These layers have smaller window sizes and more Conv filters and layers, which can be interpreted as focusing on smaller pixel windows more heavily.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 5 - ResNet-family architectures.

Some augmentations and transformations were applied on the image data for the ResNet50 model. Same augmentation parameters were used as the ResNet34 model, which are do_flip = True, flip_vert, False, max_rotate = 90. However, the resize method was changed to 'squish' rather than the default 'crop'. The default 'crop' method crops the images to obtain the predefined image shape, but the 'squish' method squishes the images to the predefined image shape. This could include some important information but also lose some due to the loss of the original proportions. A preview of some of the images are portrayed in *Figure 16 (Appendix)*.

Similar to the ResNet34 model, the initial setup for the ResNet50 model was done utilizing the fastai module. The FC layers had multiple BatchNorms, Dropouts and Dense layers with linear and relu activations. Both learning rate and weight decay were used for the ResNet50 model as well. The loss vs. learning rate graphs used in determining the optimal learning rate for the One-Cycle-Policy is demonstrated below. Both the top layer training and full training used the One-Cycle-Policy. During the top-layer training using the One-Cycle-Policy, the ResNet50 model was able to achieve 0.57 validation loss after 10 epochs of top-layer training. This is extremely significant, since ResNet34 was only able to obtain 0.73 after 10 epochs of top-layer and 100 epochs of full training. Although the top-layer training of ResNet50 illustrated powerful performance increase in small number of epochs, the full training did not result in as much improvement. During the full training of 10 epochs, the validation loss ultimately decreased to 0.42. The interesting part about ResNet50 was that the

validation loss and the training loss already crossed each other during the top-layer training. Then the 10 epochs of full training resulted in the validation loss and the training loss diverging from each other. By the end of the full training, the validation loss seemed to be not improving even though the training loss was going down. This is an apparent sign of overfitting, hence the training was ended. The short full training means that the pre-trained set of weights already work well on the Stanford Cars data, so further training (or weight updates) is not necessary.

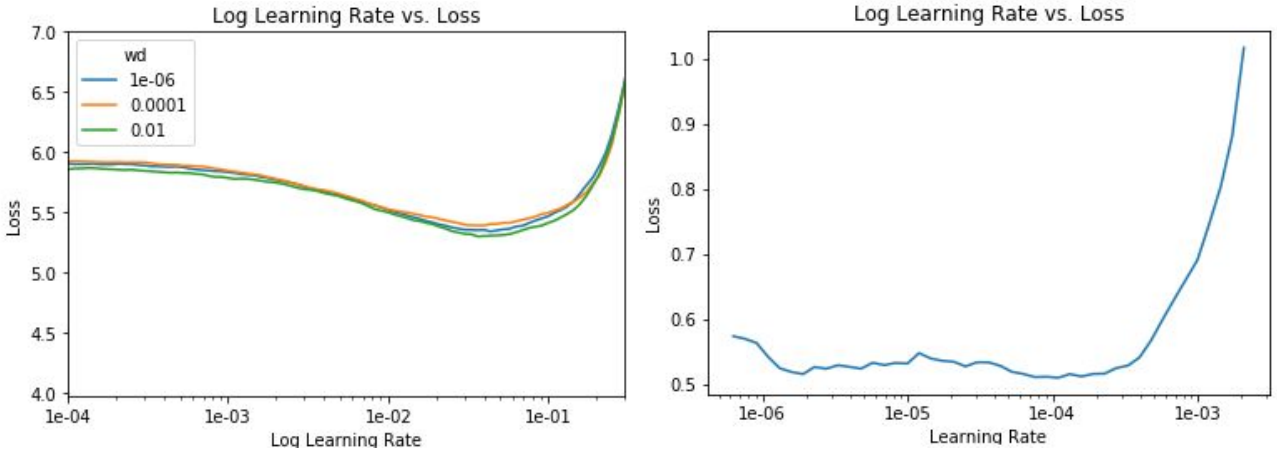
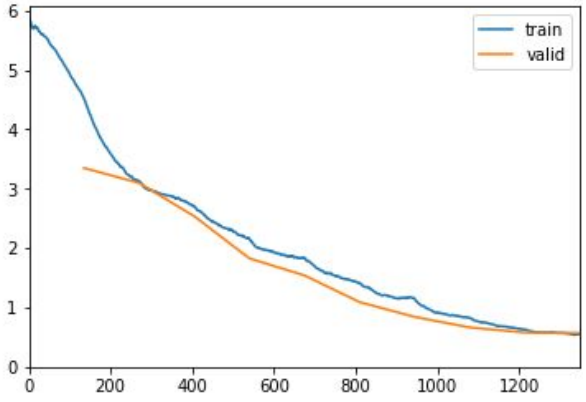
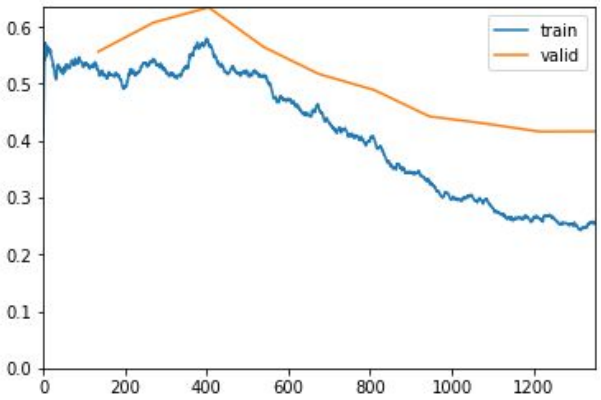


Figure 8 - ResNet50 Log Learning Rate vs. Loss graph before and after training the FC layers.

epoch	train_loss	valid_loss	accuracy	time
0	4.534760	3.349324	0.257533	1:04:04
1	3.134347	3.097307	0.278651	1:10:56
2	2.708412	2.537358	0.355138	1:10:16
3	2.165936	1.827119	0.507597	1:10:22
4	1.841726	1.539408	0.581252	1:11:16
5	1.419916	1.089868	0.696369	1:10:57
6	1.149049	0.841191	0.757147	1:11:00
7	0.824681	0.663156	0.814319	1:11:03
8	0.629282	0.578202	0.843678	1:10:17
9	0.547708	0.570551	0.843420	1:10:21

epoch	train_loss	valid_loss	accuracy	time
0	0.525315	0.556713	0.849086	57:36
1	0.543948	0.607696	0.831316	1:28:48
2	0.572520	0.634493	0.819212	1:28:36
3	0.514778	0.564616	0.837754	1:27:35
4	0.460670	0.517179	0.857842	1:28:57
5	0.404159	0.489222	0.858099	1:28:48
6	0.327171	0.442721	0.873036	1:27:26
7	0.294380	0.430297	0.879990	1:28:53
8	0.262758	0.415979	0.881020	1:29:26
9	0.254132	0.416259	0.880247	1:28:58





```
The precision score is 0.891808046663706
The recall score is 0.880247231522019
The fscore score is 0.8815954450168534
```

Figure 9 - ResNet50 top-layer and full training model performance.

ResNet50 resulted in significant improvement in precision, recall and F1-score. All three of these performance metrics were above 0.88. As shown in the confusion matrix in the Appendix, the diagonal actual vs. predicted line is slightly more apparent than the ResNet34 model. According to the most confused images summary below, the overall misclassified count has decreased. In fact, the Bentley family misclassification count went down to 8 from 19. All the misclassified labels are still very similar in shape, which indicates that the errors made by the model is due to the similarities of vehicle shapes. In *Figure 20*, the images that have no important areas disappeared. This means that the ResNet50 model is picking up important features from vehicles, due to the deeper architecture and the learning. However, there are still heatmap images that do not focus on the Make logo. It could be appropriate to conclude that even though ResNet50 model performs better, more variety of training images and further training will still be beneficial.

```
[('GMC Savana Van', 'Chevrolet Express Van', 16),
 ('Bentley Continental GT Coupe', 'Bentley Continental Flying Spur Sedan', 8),
 ('Audi S5 Coupe', 'Audi A5 Coupe', 6),
 ('Chevrolet Express Van', 'Chevrolet Express Cargo Van', 6)]
```

Figure 10: ResNet50 most confused images.

Analysis of Results

Several metrics were used to compare model performance. In addition to overall accuracy, weighted precision, recall, and F1-score were all calculated. The F1-score describes a balance of precision and recall. The weighted F1-score for a multi-class problem takes into account the class imbalance, and generates a fairer performance score. These metrics are provided in *Table 6*.

Model	F1 Score	Accuracy	Precision	Recall	Image Size
SVM	0.08	0.09	0.08	0.09	50 x 100
Random Forest	0.06	0.06	0.07	0.06	50 x 100
CNN (Custom)	0.29	0.29	0.33	0.27	224 x 224
CNN (ResNet34)	0.79	0.79	0.81	0.79	224 x 224
CNN (ResNet50)	0.88	0.88	0.89	0.88	224 x 224

Table 6 - Summary of model performances on Make & Model classification

SVM and Random Forest models yielded poor performance despite rigorous hyperparameter tuning. Both models produced similar performances that were better than random guessing but were considerably inferior to all CNN models. It should be noted that different models had access to different amounts of resources. This is reflected in the input image size column in *Table 6*. ResNet50 was by far the best performing model that had the highest weighted precision, recall and F1-score. Moreover, it took way less time cost than the ResNet34 model, since the pre-trained weights were extremely suitable for the Stanford Cars dataset. Considering both the cost and performance, it is obvious that ResNet50 is the top performing model.

Validation & Testing

To further investigate model performance on the real world data, we collected thirty additional images that were not included in the Stanford Cars dataset to gauge the real-world performance of the ResNet50 model. It is clear that more images in this ultimate testing phase would be ideal; however, these images were gathered and reviewed by the team manually. Due to the significant cost of the gathering of real-world data requires, only thirty images were used. In the original Stanford Cars dataset, the images were collected mainly from Google and Flickr, and went through a vehicle detection phase, which provided the coordinates of bounding boxes of the object of interest. This preprocessing phase removed a lot of the background noise. However, the final testing images were images that included the background, which explains the decrease in performance measures.

Model	F1 Score	Accuracy	Precision	Recall	Image Size
CNN (ResNet50)	0.73	0.73	0.73	0.73	224 x 224

Table 7 - ResNet50 model validation performance.

In addition to the background noise which is the key suspect, there were other factors that played a role in this decrease. The final testing set was collected from Google, which resulted in having access to newer yearly model images. The ResNet50 model was trained on the Stanford Cars data that was developed years ago. This means that the model was learning from older models of cars. These slight differences between the old and the new generations of vehicles definitely had a negative impact on the model performance. The performance gap may also have been led by the small-sample bias and the unstructured nature of these additional images.

Actual	Predicted
Audi A5 Coupe	Chevrolet TrailBlazer SS
Audi S6 Sedan	Mitsubishi Lancer Sedan
Dodge Challenger SRT8	Chevrolet Camaro Convertible
Ferrari 458 Italia Coupe	Geo Metro Convertible
Hyundai Sonata Sedan	Ford Fiesta Sedan
Land Rover Range Rover SUV	Cadillac SRX SUV
Mercedes-Benz SL-Class Coupe	BMW 1 Series Convertible
Rolls-Royce Phantom Drophead Coupe Convertible	Rolls-Royce Phantom Sedan

Table 8 - Misclassified Validation Images

The ResNet50 model performed fairly well on the validation set. In the misclassified validation images table above, most of the misclassified images are in fact very similar in overall shape. As discussed in the ResNet50 section, this means that the model is not making mistakes by random guess but due to the presence of certain objects in each image.

Discussion & Conclusions

Results

The team had expected CNNs would perform better than non-deep-learning models. However, we were surprised with the gap in performance between the two groups of algorithms, with state-of-the-art CNN architectures achieving great success despite the difficult task of classifying vehicles. The performances achieved in the ResNets provide strong evidence in promoting further research in creating a more complex car classification model to deploy to be used in a car identification app in the future.

The SVM and Random Forest non-CNN models performed considerably worse than CNN models. This was likely due to the nature of the models, inherently being unable to retain positional information about the pixels in a 3D recognition task. While these algorithms have shown success in simpler 2D image classification tasks such as the MNIST digits classification, the images found in the Stanford Cars dataset proved to be too complex for the two algorithms. Comparing the two non-CNN models, SVMs had slightly higher performance metrics (F1 score = 0.08, etc). This may be due to SVM's flexibility in the shape of their decision boundaries in higher dimensions, especially with complex kernels such as RBF. Given a more rigorous round of feature extractions more applicable to image classifications, we may have been able to improve the performance for SVMs. However, it would not reach the performances of CNN models, making further research on improving performance unnecessary. Similar problems arose for Random Forests as well, but with even larger consequences (F1 score = 0.06, etc). This was likely due to how decision tree decision boundaries are straight lines perpendicular to the axis of the feature, which limits the flexibility of decision boundaries. Furthermore, the combination of the large feature set and how Random Forests take subsets of features most likely exacerbated the situation. Considering how the edges found on cars are likely the most important features in classifying the different makes and models, the relevant classification features most likely occupy a very small portion of the feature set. Therefore, it is likely that many of the decision trees in the RF did not contain the important pixels of the images, making it difficult for Random Forest to create a relevant model.

Ethics and Social Acceptance

As image-recognition becomes more ubiquitous, it raises many questions about the ethical use of such capability. As recently employed in China, image-recognition methods can be used to spy on the population, giving rise to many questioning regarding privacy and it being violated. By not being regulated and not protecting personal information, such use of data lies on the border of ethics, the ownership of the data and what is socially acceptable. Regarding the dataset the authors used, one solution would have been to blur out license plates and faces during data collection for instance.

Conclusion

Among all machine learning methods used, CNNs are the models providing the best vehicle recognition model with precision, recall and f-score all above 0.88 on the validation set and 0.73 on additional testing images. This approach should be further explored by adding a substantial number of images to train the models as well as adding more classes (car model) in order to make this model more relevant from the application development business perspective.

Appendix

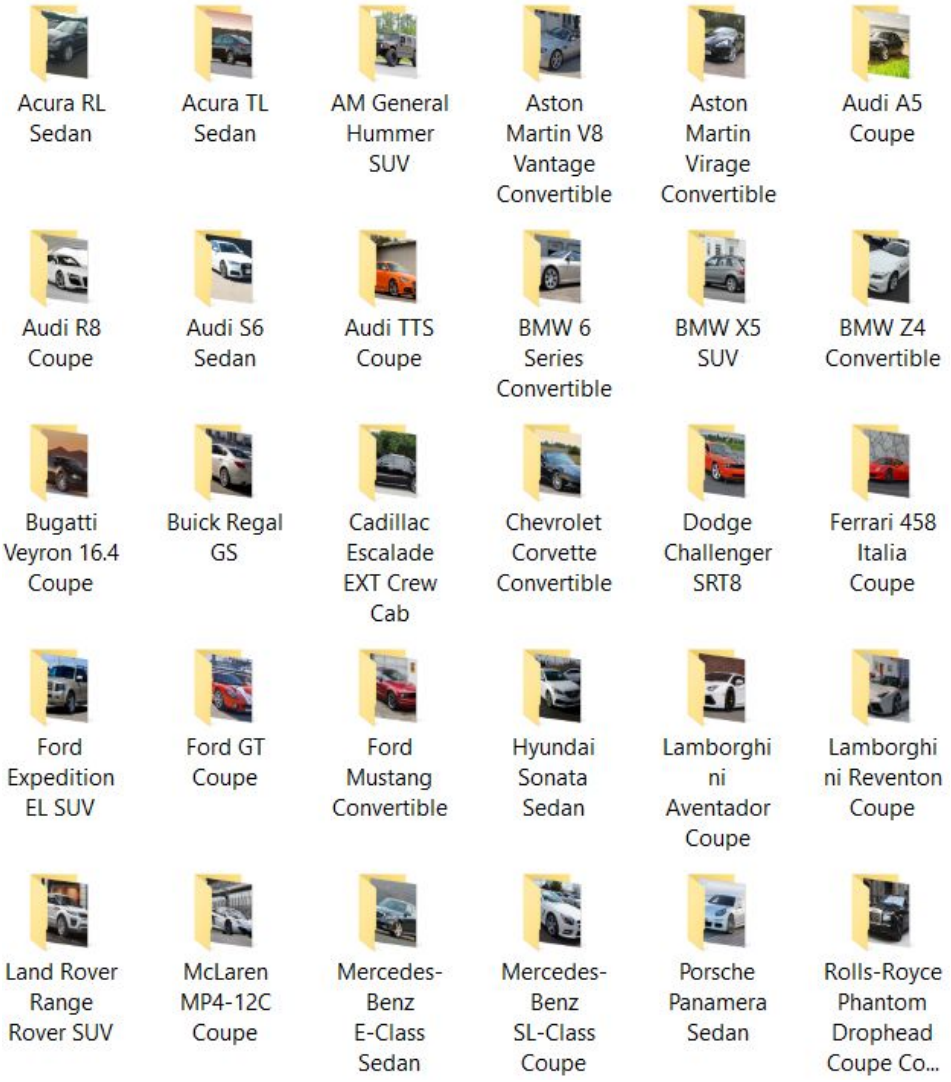


Figure 11 - Additional 30 images withheld for final model validation.

Figure 12 - Sample images from each Make+Model class.

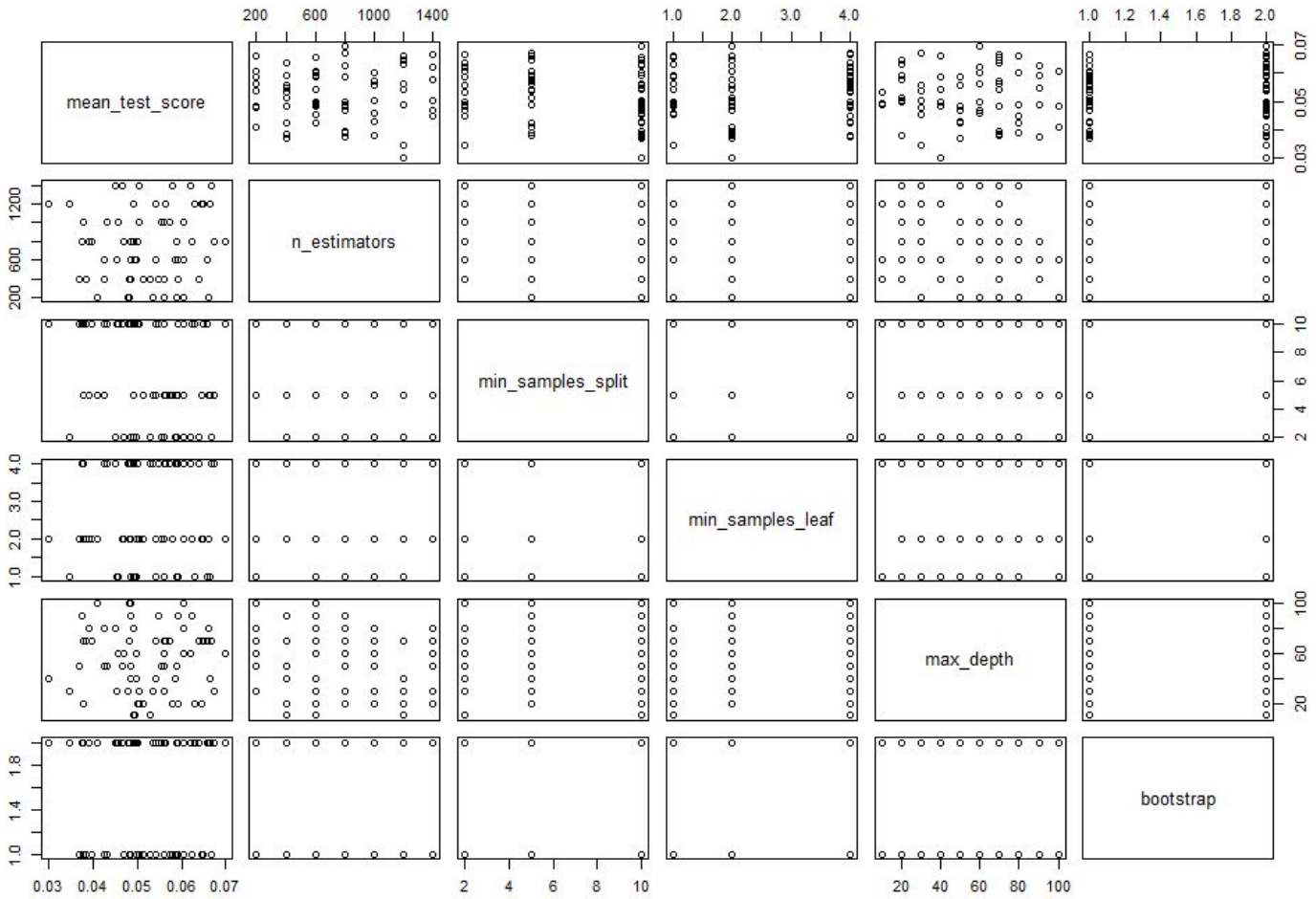


Figure 13 - Scatterplot matrix of mean test scores of 5-fold cross-validations with respect to RandomizedSearchCV() hyperparameters.

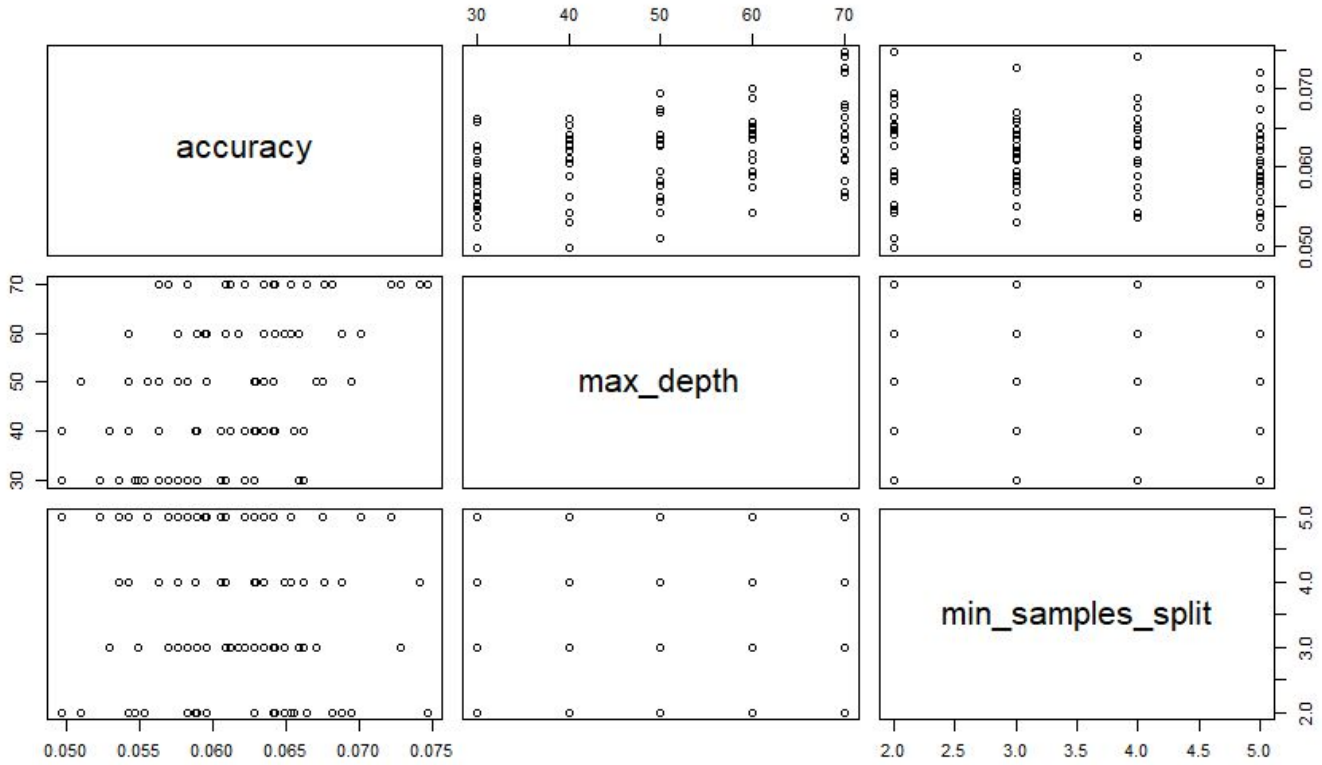


Figure 14 - Scatterplot matrix of mean test scores (accuracy) of 5-fold cross-validations with respect to the two grid search hyperparameters.

	Layer	Output Shape
1	Input Image	224 x 224
64	2D Convolutions (3x3) MaxPooling (2x2)	111 x 111 x 64
128	2D Convolutions (3x3) MaxPooling (2x2)	54 x 54 x 128
256	2D Convolutions (3x3) MaxPooling (2x2)	26 x 26 x 256
512	2D Convolutions (3x3) MaxPooling (2x2)	12 x 12 x 512
64	2D Convolutions (3x3) MaxPooling (2x2)	5 x 5 x 64
1	Flatten	1600
4608	Dense (Linear) Dropout (0.25)	4608
4608	Dense (Linear) Dropout (0.25)	4608
189	Dense (Softmax)	189

Figure 15 - Custom CNN model structure.

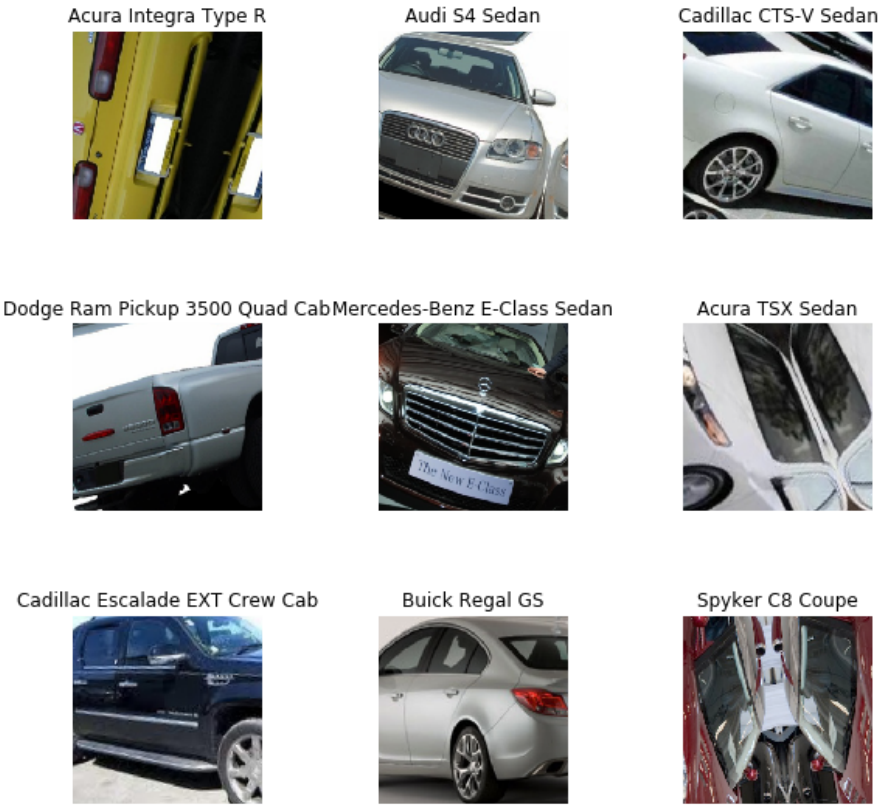
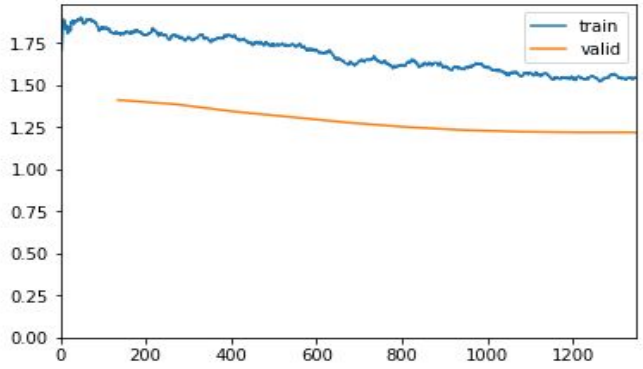


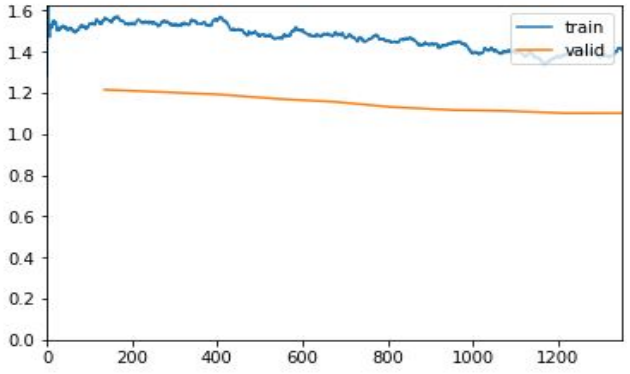
Figure 16 - Preview of the ResNet34 training images with augmentation/transformation.

Figure 17 - Preview of the ResNet50 training images with augmentation/transformation.

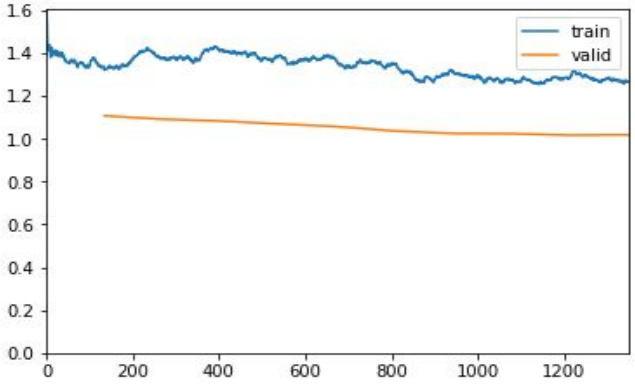
epoch	train_loss	valid_loss	accuracy	time
0	1.808390	1.410663	0.632243	49:07
1	1.790068	1.385608	0.638939	50:36
2	1.781615	1.342545	0.654133	48:59
3	1.734212	1.309546	0.659284	49:18
4	1.619155	1.276190	0.664692	48:52
5	1.619910	1.250240	0.673191	49:28
6	1.594307	1.232061	0.673963	49:26
7	1.550314	1.222225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



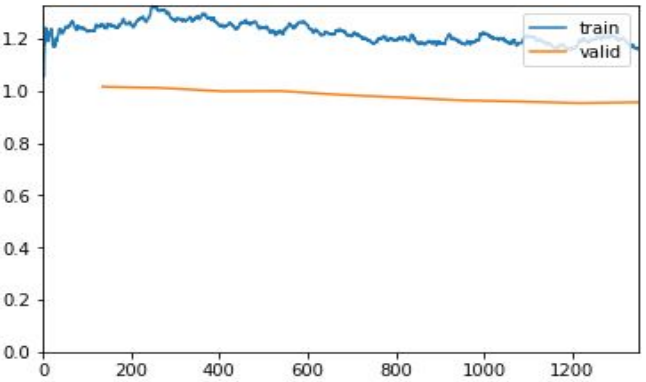
epoch	train_loss	valid_loss	accuracy	time
0	1.557081	1.216002	0.675251	14:58
1	1.545179	1.204483	0.681689	14:48
2	1.564081	1.192634	0.684522	14:51
3	1.465772	1.171620	0.688643	26:22
4	1.483997	1.156955	0.693021	47:19
5	1.449640	1.131565	0.704095	48:21
6	1.440592	1.117790	0.703580	46:59
7	1.400102	1.112542	0.704095	49:10
8	1.384001	1.101945	0.708215	47:01
9	1.410389	1.102118	0.707443	48:00



epoch	train_loss	valid_loss	accuracy	time
0	1.321524	1.107562	0.706155	37:22
1	1.378396	1.091762	0.708215	46:38
2	1.416011	1.083155	0.712078	48:37
3	1.387095	1.069836	0.714654	47:32
4	1.357621	1.057059	0.720577	47:41
5	1.350469	1.036730	0.725728	48:44
6	1.307412	1.025207	0.729591	46:43
7	1.274621	1.024214	0.730363	48:47
8	1.290659	1.016932	0.732423	46:05
9	1.266117	1.018915	0.731393	48:49

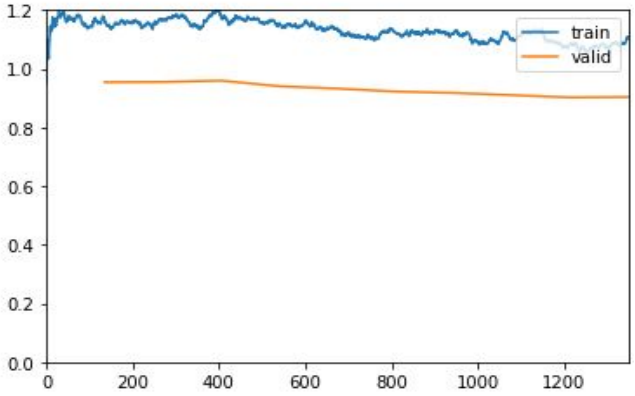


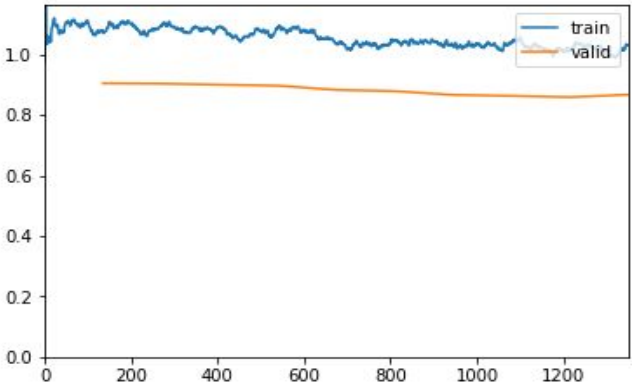
epoch	train_loss	valid_loss	accuracy	time
0	1.250747	1.015860	0.732938	47:46
1	1.307564	1.010971	0.730621	47:41
2	1.249875	0.998471	0.736029	47:18
3	1.230348	0.999026	0.734484	48:15
4	1.232139	0.984812	0.734226	46:46
5	1.207032	0.974829	0.742982	48:10
6	1.189793	0.963933	0.742725	46:56
7	1.181206	0.958936	0.743497	47:41
8	1.183900	0.953023	0.745815	47:05
9	1.159541	0.956321	0.744785	47:41



epoch	train_loss	valid_loss	accuracy	time
0	1.168369	0.954173	0.746073	47:08
1	1.166362	0.954944	0.741952	47:26
2	1.184143	0.959483	0.745043	47:08
3	1.153834	0.940191	0.748648	47:51
4	1.137994	0.932312	0.747360	47:12
5	1.123653	0.922264	0.751481	47:53
6	1.118429	0.917528	0.754829	47:37
7	1.097875	0.909929	0.753799	47:31
8	1.086187	0.902132	0.755086	48:07
9	1.109474	0.903756	0.754314	46:52

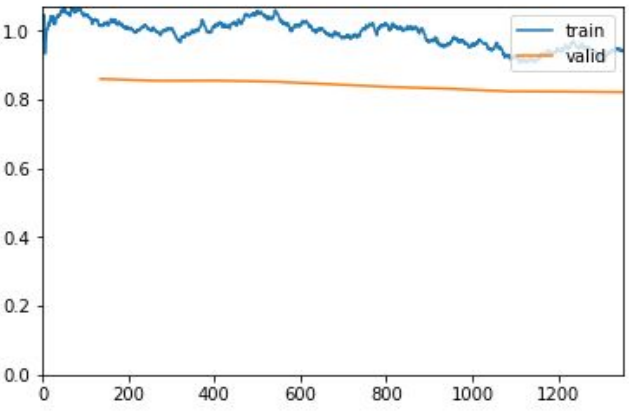
epoch	train_loss	valid_loss	accuracy	time
0	1.074703	0.904523	0.757404	14:52
1	1.097129	0.903359	0.757404	14:50
2	1.092081	0.899894	0.756374	14:58
3	1.088593	0.896544	0.756631	24:34
4	1.035675	0.882772	0.761267	47:48
5	1.048380	0.878450	0.761267	47:56
6	1.029200	0.866074	0.764873	49:15
7	1.038603	0.862865	0.762297	47:42
8	1.012528	0.858699	0.764873	47:58
9	1.029502	0.866741	0.763842	49:20

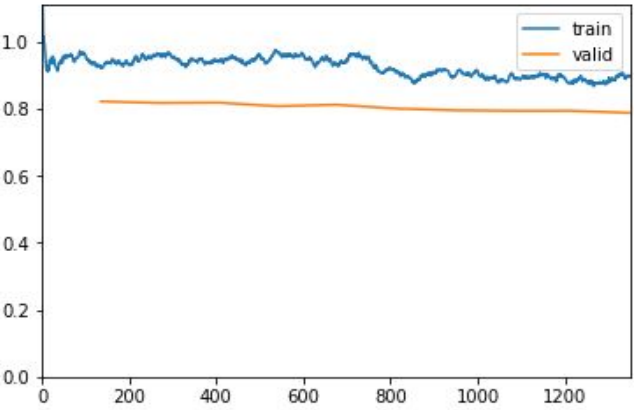




epoch	train_loss	valid_loss	accuracy	time
0	1.014698	0.859986	0.764873	43:32
1	0.995521	0.854635	0.768736	48:41
2	1.018612	0.855251	0.766675	46:24
3	1.044091	0.851986	0.768478	48:38
4	0.986598	0.844310	0.773114	46:36
5	1.006567	0.836668	0.772341	48:20
6	0.961997	0.831541	0.773371	46:45
7	0.927437	0.823986	0.776719	48:14
8	0.946067	0.823031	0.777749	47:13
9	0.940417	0.821491	0.776204	48:06

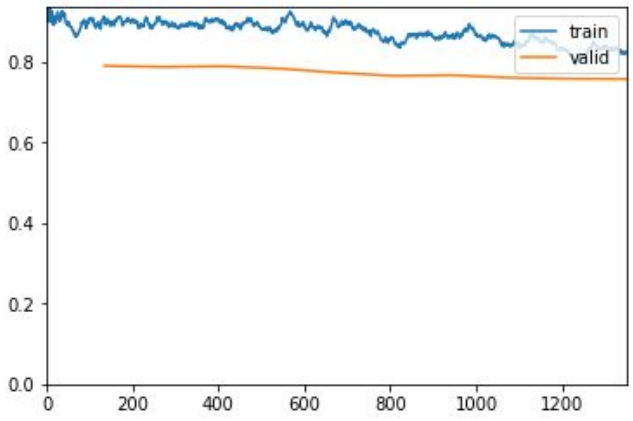
epoch	train_loss	valid_loss	accuracy	time
0	0.925195	0.822583	0.777492	47:00
1	0.960630	0.818020	0.777749	48:39
2	0.940481	0.818945	0.778779	46:50
3	0.971666	0.808305	0.777234	48:22
4	0.932030	0.812829	0.778264	46:55
5	0.917777	0.801429	0.779294	48:02
6	0.895212	0.796258	0.783672	47:22
7	0.905308	0.794433	0.785733	47:41
8	0.881446	0.794623	0.783157	48:14
9	0.898755	0.788267	0.786505	47:41





epoch	train_loss	valid_loss	accuracy	time
0	0.907949	0.790865	0.783672	42:28
1	0.899129	0.787927	0.781097	47:36
2	0.896608	0.789776	0.780325	48:17
3	0.888053	0.784256	0.786763	46:54
4	0.900395	0.773797	0.784960	48:36
5	0.851556	0.765870	0.787020	46:53
6	0.869491	0.767473	0.788308	48:45
7	0.841771	0.761102	0.789853	46:46
8	0.824650	0.758777	0.788566	48:22
9	0.825978	0.757898	0.789596	47:00

epoch	train_loss	valid_loss	accuracy	time
0	0.828971	0.757601	0.789338	47:57
1	0.823475	0.759355	0.789338	47:17
2	0.834208	0.758517	0.786248	47:51
3	0.851897	0.753168	0.785733	47:02
4	0.821245	0.749253	0.790883	47:52
5	0.804125	0.741682	0.790111	46:54
6	0.818743	0.739885	0.791141	48:05
7	0.801630	0.729439	0.791398	47:00
8	0.798832	0.732448	0.792944	47:57
9	0.780111	0.731119	0.793974	46:53



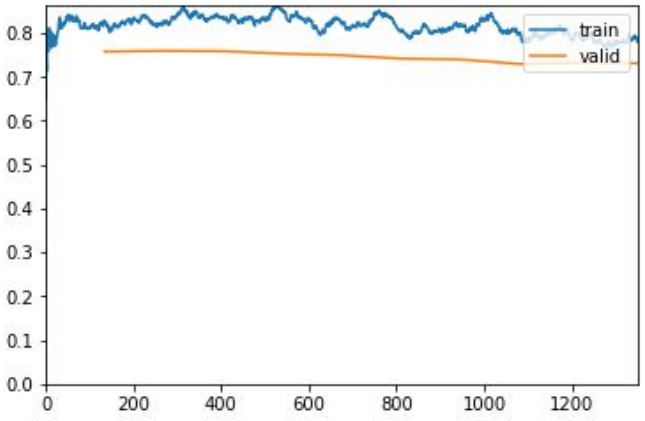


Figure 18 - ResNet34 Training Outputs (10 phases of full training, 10 epochs each, excluding the top layer training).

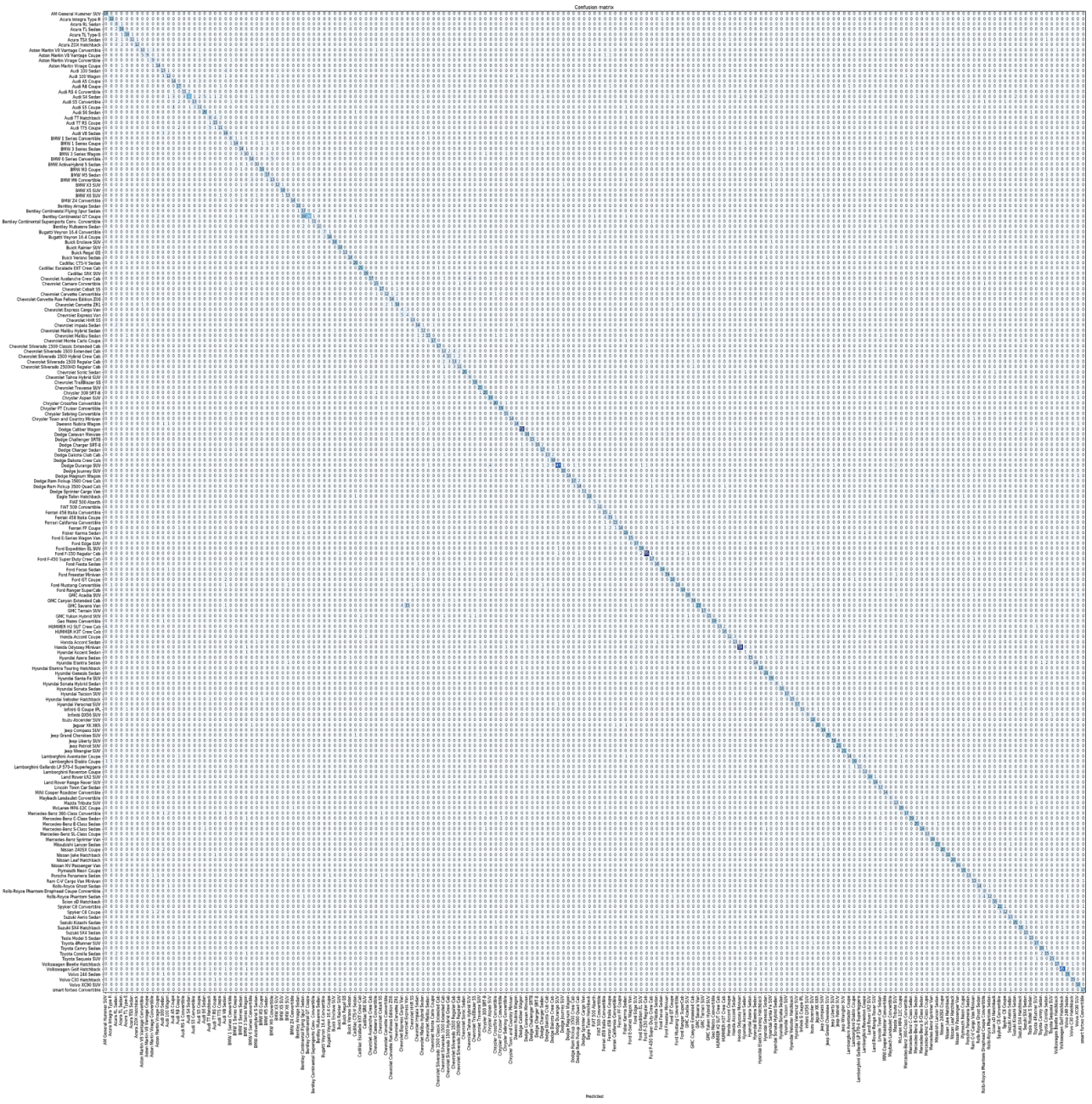


Figure 19 - ResNet34 Confusion Matrix.

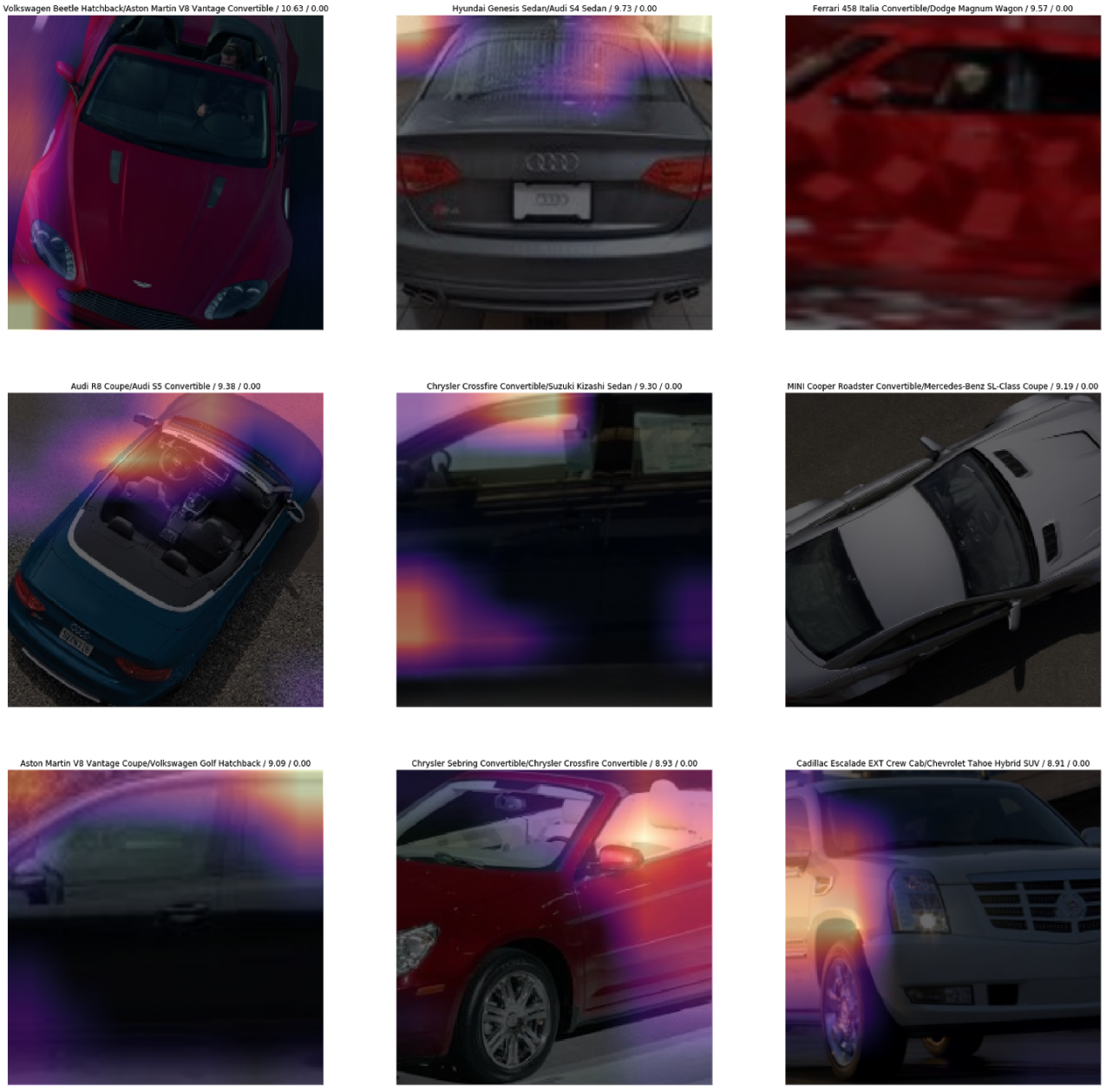


Figure 20 - Heatmap of areas of focus for the top-loss images by the ResNet34 model.

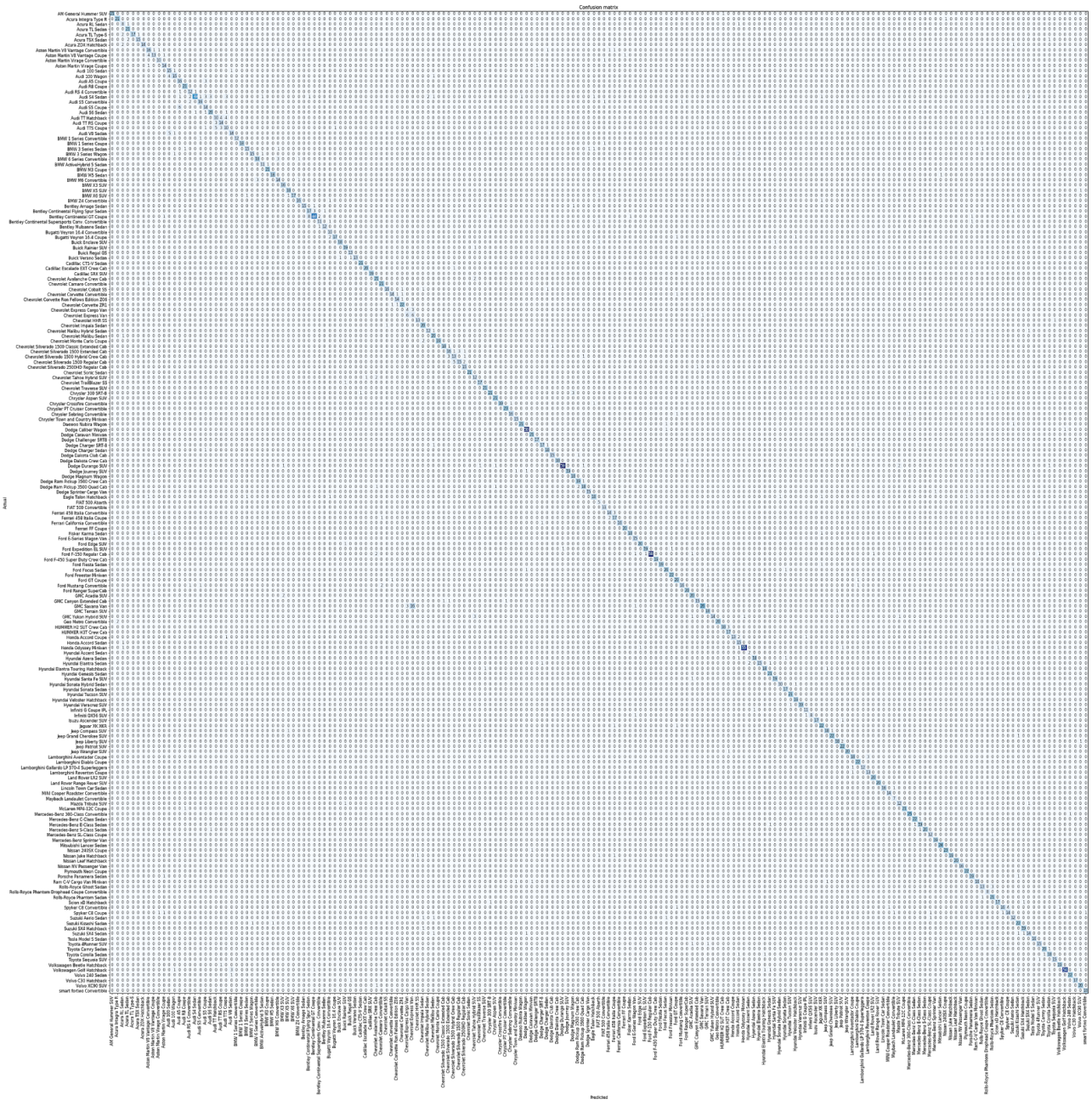


Figure 21 - ResNet50 Confusion Matrix.

Figure 22 - Heatmap of areas of focus for the top-loss images by the ResNet50 model.