Car Make and Model Prediction Using Xception

Team 06 — Big Data Analytics
Ariannah Black, Ummea Salma, Sai Kiran Belana, Md Ishtyaq Mahmud
Department of Computer Science and Engineering
University of Connecticut — Spring 2024
Storrs, CT

***Abstract*—Vehicle identification and classification models can serve a vital role in a variety of aspects for traffic safety and driver enforcing accountability. Developing a machine learning model that can, from a glance, quickly and accurately determine the make and model of a car, while difficult, has the potential to benefit victims of vehicle-related incidents, authorities, and other parties alike.**

I. Problem Statement

A. Motivation, Importance, and Goals

The overall goal for this project was to generate a cohesive piece of software which is able to identify the make and model of a car from an arbitrary image. Vehicle identification can play a significant role in a variety of disciplines: for authorities, such software could help identify stolen vehicles or parties involved in traffic violations; for drivers, it could verify other cars involved in collisions when incorporated into dashboard cameras; and for hobbyists, it may provide an easy way to identify rare vehicles.

While vehicles are certainly distinct enough to the human eye to quickly and easily decide the make and model (should the individual be generally knowledgeable about cars), the value for these kinds of models lies in their efficiency, versatility, and ease of use. While the model in this project has been trained on a considerably small dataset of images alone, should the team expand upon the software in the future, it would be rather similar to the same technologies that allow authorities to identify offenders of speeding or similar incidents automatically, provided video footage. Instead of requiring a human aspect for predicting each vehicle that was captured, the same task can be performed more efficiently using a model similar to the one in this project.

B. Project Overview

The model developed in this project is for classification; the input is an image of a vehicle, alongside the coordinates of a bounding box that encapsulates the vehicle within the image, and the output is the vehicle's predicted make, model, and year.

The primary metric for evaluating the performance of this model is considering the percentage of accurate make and model classifications for the pre-determined testing set. Additionally, the team considers the relative proximity of the model's predictions for each class by evaluating precision, recall, and the F1 score. Considering the model's performance with respect to each individual class will yield a better understanding of exactly where the model's strengths lie. While not ideal, for instance, the team defines it more acceptable for the model to be more successful at identifying the make or manufacturer of the vehicles and less successful at predicting the model and year, rather than vice versa. The year of a car does not necessarily correlate to distinct trends in car manufacturing style (as a 2016 Honda Civic may look very similar to a 2017), and different models belonging to the same manufacturer may also have very similar characteristics overall (like a 2016 Honda Civic and a 2016 Honda Accord).

II. The Dataset

The dataset chosen for this project was the Stanford Cars Dataset, which contains 16,185 images of 196 different classes or types of vehicles. The original data was uploaded by PhD student at Stanford Jonathan Krause in his online archive, but was removed a few years prior to the start of this project. The team was able to piece the original dataset back together by locating a few sites where parts of it were secondarily uploaded.

The dataset can be divided into two major components: the images and the annotations.

A. Images

The images are small JPG files with slight deviations in size. Most were relatively closely cropped in order to capture less extraneous information prior to any data preparation, though some were not. There were a few images of slightly lower quality than others (that were more pixelated or were not taken under ideal conditions), though the difference was not severe enough to warrant any instances of image removal.

B. Annotations

The annotations portion of the dataset contains all metadata about the images, divided into two types of matrices: a smaller matrix to associate each car class name (a number between 1 and 196) with its correct make and model, and a larger matrix to connect each image's file name with its bounding box coordinates and class number. Overall, there are six major features in the dataset (bounding box coordinates (x_1, x_2, y_1, y_2), class name or number, and file name or path) and 16,185 instances.

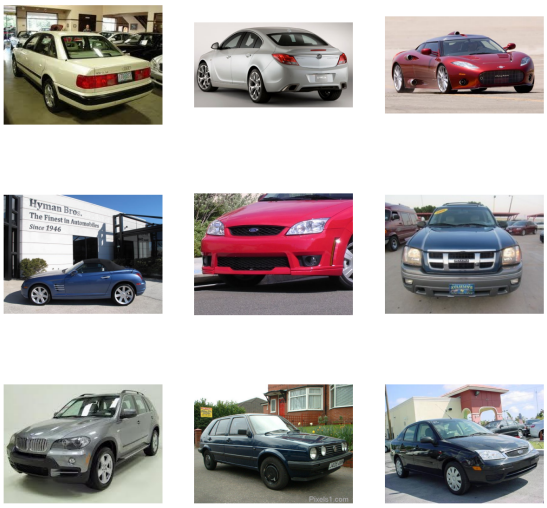

Fig. 1. An assortment of sample images from the Stanford Cars Dataset.

III. Data Preparation

Since there were no instances of missing or incomplete data within either the images or annotations portions of the dataset, the group did not need to take any steps for repair. The only text attributes within the matrices belonged to the feature denoting the class label names, which was replaced by the numeric (categorical) and equivalent attribute of class name (denoted by a number 1 through 196). A more thorough explanation of the model's handling of categorical attributes is below.

A. Dividing the Dataset

Although the dataset was already split into train and test sets, the team generated a new random split for the subsets to ensure that the instances were divided randomly and in accordance to the expected ratio (80% of the instances would be used for training, and the remaining 20% for testing or evaluation). In order to do this, the team wrote a script that performs the following:

1) Combines the instances within both the training and testing directories.
2) Divides the larger set into training and testing sets, according to the decided ratio of 80/20.
3) Splits the resulting training set into training and validation subsets, according to a decided ratio of 75/25.

B. Cropping Images

Since there was an array of images within the dataset which were not quite as closely cropped as they could have been, the team wrote a script to crop the images according to their specified bounding boxes. This assisted the model such that it would receive less extraneous information and thus less noise during training and validation.

C. Resizing Images

In order to better prepare the images for integration into the model, the team regularized them by ensuring that the sizes were consistent for all instances. Rather than resizing to a more common value of 180x180, the team chose 299x299 as to include more details of each car for classification. Since there are few instances per class (with an average of around 50), small details are essential. Note that square resolutions are ideal here, since convolutional neural networks train best on square images.

Fig. 2. An image after being cropped to its bounding box and resizing (not to scale).

D. Random Eraser

The team wrote a script to randomly obscure a portion of the image with a rectangle of a random size, aspecr ratio, and color. This adds some aspect of randomization to the training set, allowing for a greater variety of training data and less chance of overfitting.

E. Data Augmentation

In order to reduce overfitting and improve model generalization, the team used an ImageDataGenerator from the Keras library to increase dataset size and balance it for training. This way, the image diversity could be artificially inflated for better performance on the validation and test sets. The images experienced a random horizontal flip (since photos may be taken of a vehicle from either side), random rotation of 0.1 (since photos may not be perfectly level, but they will not typically be taken upside down or at very steep angles), and random zoom at 0.2 (as images may not encapsulate the vehicle perfectly).

IV. Model and Technique Selection

A. Initial Models and Techniques

The technique previously leveraged for developing the model for this project was to utilize feature extraction alongside data augmentation with a pre-trained model. Using the Keras library, the team instantiated the VGG16 convolutional base using imagenet. After extracting the VGG16 features and corresponding labels, the team added some extra layers to the model to ensure that the images were augmented and preprocessed (using Keras' built in image preprocessing method) and to polish the model (to add a dense layer using softmax, for instance). The model was trained with 50 epochs and approximately 300 instances per batch.

This technique was selected over others since similar models have been selected for similar datasets that the team had found. VGG16 was the chosen CNN method since it supports processing large-scale datasets with deeper network layers and

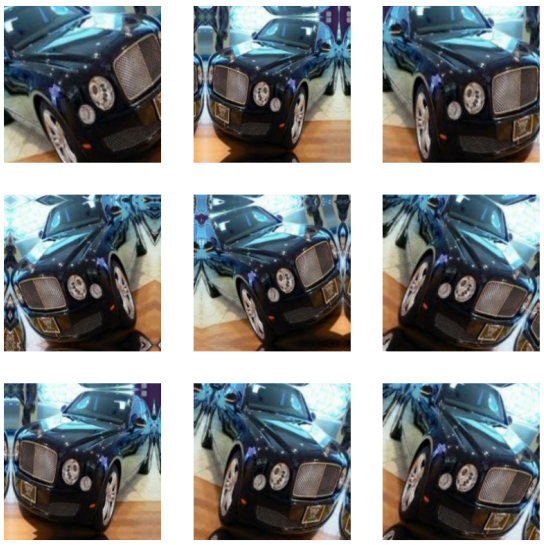

Fig. 3. A number of images produced by a single instance after data augmentation, random eraser not visualized.

smaller filters; while the project's dataset may be considered rather small for the number of classes it contains, its overall size makes training and compiling it (especially in a timely manner) significantly more difficult. Selecting a pre-trained model is generally a better choice for projects like this, as it results in less difficulties and complications for the team during the process.

There were other similar models that the team had worked with and scrapped, as their projected accuracies or their efficiencies were much too low to consider an option for the classification task.

B. The Selected Model, Before Fine Tuning

After working with the VGG16 model amongst others, the team had very little luck receiving good results during preliminary evaluations until incorporating Xception. Developed by Google, the model makes two minor differences in comparison to original depthwise separable convolutions: a slightly modified order of operations (performing 1x1 convolutions first, before channel-wise spatial convolution) and the lack of intermediate ReLU non-linearity. For incorporating Xception into the project, the team selected a pre-loaded model with the same justifications as selecting the pre-loaded VGG16 model above: for efficiency and to reduce the risk of complications during the process. Using imagenet for weights, the model included an array of additional layers in order to ensure regularized, non-overfit results and included multiple different kernel initializers and activation functions to extract the right types of information from the data.

This model, as well as the previously trained ones, are explained in more detail in the *Model Evaluation* and *Methods of Improving the Results* portions of this paper.

V. Initial Model Evaluation

A. Evaluation of Previous Models

The team developed a regularized convnet with a rescaling layer to limit pixel intensities between 0 and 1, convolution layers to compress 9 pixels into 1 using a ReLU activation function, and an increasing number of filters from 32 to 256. Max pooling layers were used for feature extraction and dimensionality reduction, and layers were flattened. The softmax activation was used to convert vectors into a probability distribution. Despite the model showing promise during training with a higher validation accuracy than previous models, it didn't show enough improvement to proceed, as the validation didn't improve significantly even though the training loss decreased over time.

The team then utilized the VGG16 model, a pre-trained model on the ImageNet dataset, for feature extraction. For the specific task at hand, the team used a technique called "Fast Feature Extraction without Data Augmentation". The model was configured to exclude the top layers, which are fully connected layers, and to accept input images of shape 224x224x3. The total parameters of this model were approximately 14.7 million, all of which were trainable. For fast feature extraction without data augmentation, the team designed a new model. The input to this model was of shape 7x7x512, which is the output shape of the VGG16 model. The team used Global Average Pooling to convert the 3D outputs to 1D vectors. Then, they flattened the output and connected it to a dense layer with 256 neurons. A dropout layer was added for regularization, followed by the final output layer with 196 neurons (one for each class) using the softmax activation function.The model was compiled with the loss function set to `sparse_categorical_crossentropy`, the optimizer set to `rmsprop`, and the metrics set to `accuracy`. The `sparse_categorical_crossentropy` loss function is used when the classes are mutually exclusive, i.e., each sample belongs to exactly one class. The `rmsprop` optimizer helps to adjust the learning rate dynamically, making it a good choice for many machine learning models. The `accuracy` metric is used to measure the model's performance.

During training, the model was set to save the best weights based on validation loss. The model was trained for 50 epochs. At the end of training, the model achieved a training accuracy of 97.02% and a validation accuracy of 66.02%. The training loss was 4.8667, while the validation loss was significantly higher at 115.5264. This discrepancy between training and validation metrics suggests that the model might be overfitting on the training data, and therefore, might not generalize well to unseen data.

To address this, the team utilized "feature extraction with data augmentations" where they created an input layer for images of size 224x224x3, applied data augmentation to the inputs, and preprocessed the images using VGG16's preprocessing function. The preprocessed images were then passed through the VGG16 base. A dense layer with 960 units and ReLU activation was added, followed by batch normalization

to standardize the inputs to the next layer, and a dropout layer to reduce overfitting. This sequence was repeated once more.

Finally, a dense layer with 196 units and softmax activation was added to output probabilities for each of the 196 classes. The model was compiled with sparse categorical cross-entropy loss, RMSprop optimizer, and accuracy as the metric. During training, the model checkpoint with the best validation loss was saved, and training was stopped if there was no improvement in validation loss for 10 epochs. The model achieved a training accuracy of 93.71% and a validation accuracy of 81.74% at epoch 17.

The model was fine-tuned by unfreezing the last four layers of the VGG16 base and retraining it with the rest of the model. It was trained for 100 epochs, but training stopped at epoch 11 due to early stopping. At this point, the model had a training accuracy of 95.24% and a validation accuracy of 84.83%. The model was then evaluated on the test dataset. The first model, trained with feature extraction and data augmentation, achieved a test accuracy of 82.6%. The second model, additionally fine-tuned, achieved a slightly higher test accuracy of 84.2%. This shows that fine-tuning improved the model's performance on unseen data.

B. Evaluation of Xception

Further to get some better accuracy, the team built a model using Xception architecture. The model was trained using data augmentation to increase the diversity of the training data and improve the model's ability to generalize. The model was trained using K-Fold cross-validation, which splits the dataset into 'K' subsets and trains the model 'K' times, each time using a different subset as the validation set.

In the first fold of the cross-validation, the model was trained for 50 epochs, and the model performed significantly better, with a high validation accuracy of approximately 99.50% and a low validation loss of approximately 0.0150. The model was then used to make predictions on the test data. The model performed well in predicting the classes of the test images, achieving an average accuracy of approximately 94.60%.

VI. Error Analysis

The first model, which utilized VGG16 with Fast Feature Extraction without Data Augmentation, achieved a high training accuracy of 97.02%, but the validation accuracy was significantly lower at 66.02%. This discrepancy, along with the validation loss being much higher than the training loss, indicated overfitting, suggesting that the model may not generalize well to unseen data.

To address this, the team utilized feature extraction with data augmentations in the second model. This improved the model's performance, with the training accuracy at 93.71% and the validation accuracy at 81.74%. However, the model still showed signs of overfitting, as the training accuracy was higher than the validation accuracy.

In the third model, fine-tuning the VGG16 further improved its performance. The gap between training and validation accuracy decreased, suggesting reduced overfitting. The training accuracy was 95.24% and the validation accuracy was 84.83%.

The fourth model, which used the Xception architecture with K-Fold Cross-Validation, demonstrated the best performance. It achieved significantly higher validation accuracy compared to previous models and performed well in predicting the classes of the test images, achieving an average accuracy of approximately 94.60%.

While the initial models showed signs of overfitting, the performance improved with the use of data augmentation and fine-tuning. The Xception model, trained with K-Fold cross-validation, demonstrated the best performance, achieving a high validation accuracy and performing well on the test dataset. This analysis suggests that the choice of architecture, the use of data augmentation, and fine-tuning are critical factors in improving model performance.

VII. Methods of Improving the Model

In order to improve the results, the team worked to fine tune the selected model beyond the methods already discussed within the *Model Evaluation*. The statistics for the evaluation of the effect these changes had on the overall result for the model is investigated in the following section.

A. Early Stopping

In order to reduce overfitting, early stopping was introduced to halt training once model performance stops improving on the validation dataset.

B. Modifying the Number of Epochs

In attempt to further reduce overfitting, the number of epochs used in the model was increased from 50 to 30. The team found that this, incorporating a larger number of instances per batch (approximately 400) resulted in significantly better results than previous.

C. Adding Custom Layers

In addition to the layers provided by the pre-trained VGG16 model, the team incorporated a few additional layers specific to this dataset for better performance. The added layers included drop outs to eliminate some neurons during training and limit overfitting, new dense layers with ReLU activation and He normal kernel initializer, and global average pooling to extract features and reduce dimensionality.

D. Adding Optimizers

The team found that the inclusion of the Nadam optimization function helped to reduce the overall loss and improve the accuracy throughout training and validation.

VIII. Model Evaluation After Fine Tuning

The team received very impressive results from the model in evaluations of the model's predictions against the test set after error analysis and making minor (but significant) improvements.

A. Metrics

The team tracked a variety of important metrics in order to evaluate the success of the final model, including model accuracy, loss, F1 score, and learning rate. The model trained for 43 epochs before stopping. Over the course of its training, the model saw very good accuracy scores with a greater number of epochs.

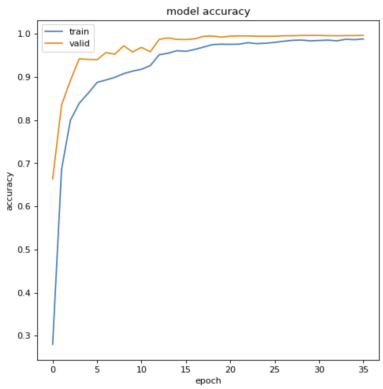

Fig. 4. The accuracy of the model's training and validation set throughout model training.

The loss became incredibly low for both training and validation after around 10 epochs, with validation loss always remaining just lower than training.

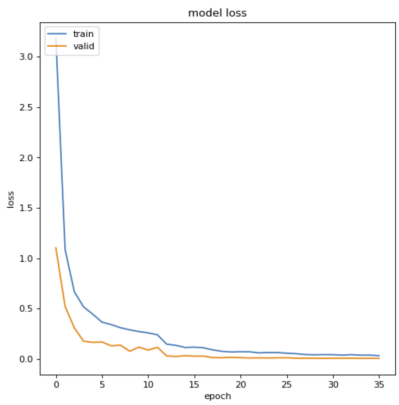

Fig. 5. The loss of the model's training and validation set throughout model training.

The F1 score also became very high after around 10 or 15 epochs, indicating a high harmonic mean of precision and recall for the model.

Overall, the model performed very well against the test set, especially considering the large number of classes it contains with few instances per class.

IX. Conclusions and Further Work

A. Issues with Stanford Cars

The final model was quite a bit more successful than the team had expected it to be, largely due to the complex issues

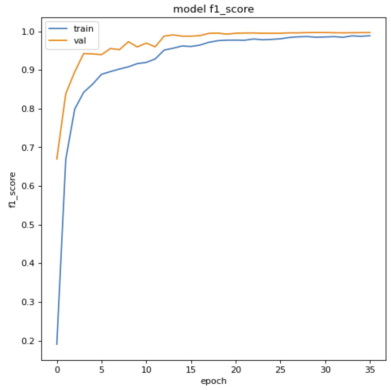

Fig. 6. The F1 score of the model's training and validation set throughout model training.

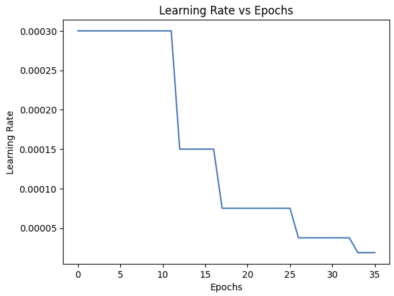

Fig. 7. The learning rate of the model throughout training.

uncovered within the Stanford Cars Dataset. It seems that many other teams that had developed classification models using the same dataset had achieved surprisingly low accuracies of correct predictions on the test set. It had come to the team's attention (much too late into the project to repair) that there are some major issues with the Stanford Cars Dataset that may cause confusion during a model's training.

There were some images within the dataset that had their classes mislabeled, belonging to classes other than their assigned ones.

There were also several ambiguous images, which include multiple cars within a single images, vehicles that do not belong to any of the labeled classes, and images that do not very well capture the vehicle's defining characteristics.

The issues found within the dataset were noticed much too late into the project to repair – because of its massive size and popularity, the believed it unnecessary to comb through images for nuanced errors. Other teams working with the same dataset and the same errors for similar tasks had also achieved lower accuracies, presumably for similar reasons.

B. Limitations

The limiting factors that the team found prevented the model from exceeding expectations were primarily as follows:

- Nuanced, but significant issues with the dataset.
- A lack of time for model development, training, and research.

Fig. 8. An array of vehicles alongside their predicted classes and actual classes. Of the twelve images shown in the figure, eleven of them had correctly-identified classes.

Fig. 9. A 2012 Audi S5 Convertible labeled as a 2012 Audi A5 Coupe.

Fig. 10. An Audi TT RS Coupe labeled as a 2011 Audi TT Hatchback.

- A lack of computing resources and funding.

Provided more time to work on the project, the team would have been able to attempt more approaches to developing and training the model and more deeply research and understand what the best course of action would be for the given dataset, specifically. In combination with a greater amount of computing resources and funding (for a Google Colab subscription or similar, in order to use more powerful hardware for faster calculations), the team would have been able to train a deeper and more detailed CNN that could use the limited dataset as efficiently as possible. While the team would be given the opportunity to explore models other than Xception, it might be more worthwhile that the team continue fine tuning the model already developed until its classification and computational limits are reached.

Fig. 11. An ambiguous top-down image of a vehicle.

C. Future Work and Project Extensions

Provided the necessary resources, the model would be given the opportunity to produce a much better evaluation against the test set. More specifically, the team could take the below next steps to compile and train an improved model.

Re-clean the data. Explained in greater detail above, it is clear that some of the images in the dataset should be removed

or repaired, and that the labels should be reexamined for validity and correctness prior to incorporation in other models. Unfit images should be removed, and incorrect class label associations either manually repaired or the image removed altogether.

Increase the class sizes. Although the dataset is rather large, the number of instances per class is too small for comfortably training. A greater number of images for each vehicle make and model would serve to greatly benefit the model.

Train with a variety of vehicles. Because the dataset is limited to 196 classes, the model can only predict classes for 196 types of vehicles. In order to allow the software to be generalized for a wider variety of cars, images for more classes could be added to Stanford Cars.

Train with images that don't have bounding boxes. Similarly to the previous point, the model could be expanded by modifying it to accept images that do not have bounding box coordinates provided. The team could incorporate this by adding some type of object recognition in order to automatically compute the bounding box values once given a clean image.

Train with more epochs and smaller batches. Because of the lack of time and computing resources, the team had to limit the number of epochs and instances per batch during model training and compilation. Having the flexibility to adapt these values to the needs of the model and dataset, rather than being limited by other factors, has the potential to produce a better model overall.