Machine Learning Engineer Nanodegree

Capstone Project

Ziad Amr Elalaily

May 11th, 2019

I. Definition

Project Overview

Since the day the car was made and there has been an increasing demand for car purchase across the globe. Cars took over the roads and started becoming the main transportation method to many people in modern times.

With the increasing technological advancements in driving, controlling and producing the car comes a need to track and analyze the diverse sea of car brands that exist today. Tracking and analyzing the car brand has been very difficult and not cost-effective [1].

Vehicle Make and Model Recognition (VMMR) is a very effective way to recognize car brands in many aspects of life. A brand seller can use data mining to check which color is favored for a certain city or neighborhood [2] or check the main location where his customers go shop and push services related to their car models.

In another situation VMMR complements Automatic Number Plate Recognition (ANPR) [3] in many situations such as police chase, traffic surveillance and parking lot car finder [4], in another case VMMR strongly supports ANPR in the case of plate forgery due to theft or damaged plate number or removing the plate number all together. Even if the car has been painted, its model shape can never change that gives a huge advantage over ANPR methods. Convolutional Neural Network was proposed to train a classifier to recognize the different models and years of cars that are available in the dataset.

Problem Statement

The aim of this project is to train a machine learning algorithm to help recognize cars from pictures and footages of surveillance cameras spread across the streets. The model should be able to recognize the car model and make to a good extent with good accuracy to the year of manufacture.

Due to the huge number of classes existing within this problem (e.g. Nissan, Audi ... etc.) and inter-class distance (e.g. Nissan Altima, Nissan Sunny... etc.), this problem is classified as a hard classification problem [4].

For an image classification problem, Convolutional Neural Networks (CNNs) shall be utilized as it is known to be of great efficiency in image classification and recognition problems.The classifier will be trained on the Stanford car dataset however instead of using the annotation to classify each model, folders were created named with model and year of make. The classifier shall recognize a car's model, make and year for example (BMW X1 2005).

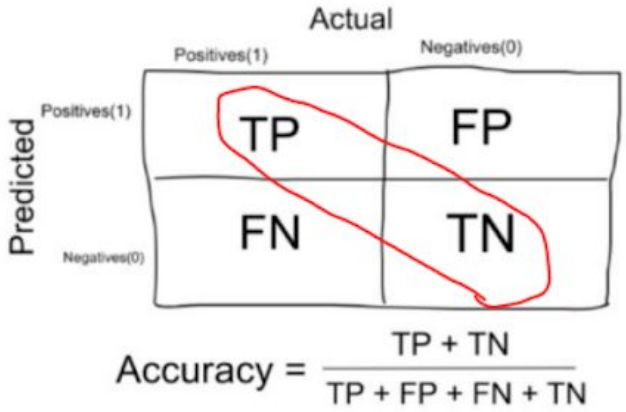
I shall try two methods of preprocessing, first I will use augmentation on raw images with no cropping of the car (leaving background noise) and another would be to apply the augmentation on the a cropped version of the cars' pictures and remove as much background noise as possible.

I will be measuring the accuracy of my algorithms by comparing it to the labeled data in the dataset and reach an ideal and close to perfect results. I shall try to implement the algorithm on many devices (if available) and see the speed of recognition of several cars and their accuracy.

Metrics

I shall implement accuracy. And since the samples in classes are balanced, the equation shall be:

$$\text{Accuracy} = \frac{\text{Correctly classified points}}{\text{All points}}$$



$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

The higher the accuracy the better the results.

II. Analysis

Data Exploration

I will use a dataset available publicly from https://ai.stanford.edu/~jkrause/cars/car_dataset.html [5]. This dataset consists of 16,185 images, divided upon 196 classes of cars models, the data is already split between 8041 images for testing and 8144 images for training, in which each class was divided almost equally.

The dataset contains different backgrounds and different angles of many car models, as well as random noise (watermarks) appearing as well.

The dataset contains pictures of different dimensions and does not have a certain size for its samples, they are all RGB colored images in jpg format of different cars in different angles with different noisy background

To show some statistics for the dataset, I shall apply it to the train set of the Acura Integra Type R 2001 on both cropped and not cropped

For non-cropped Acura Integra Type R 2001 set:

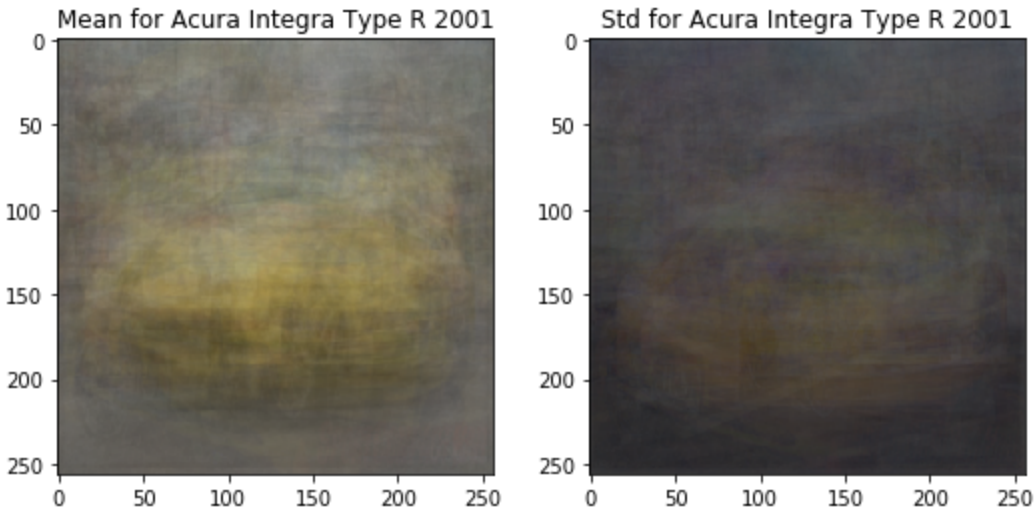
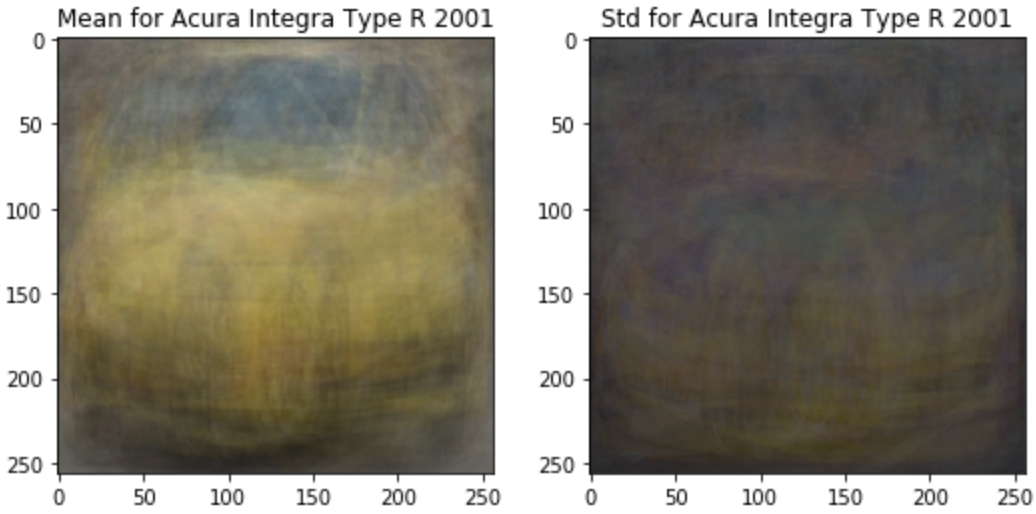
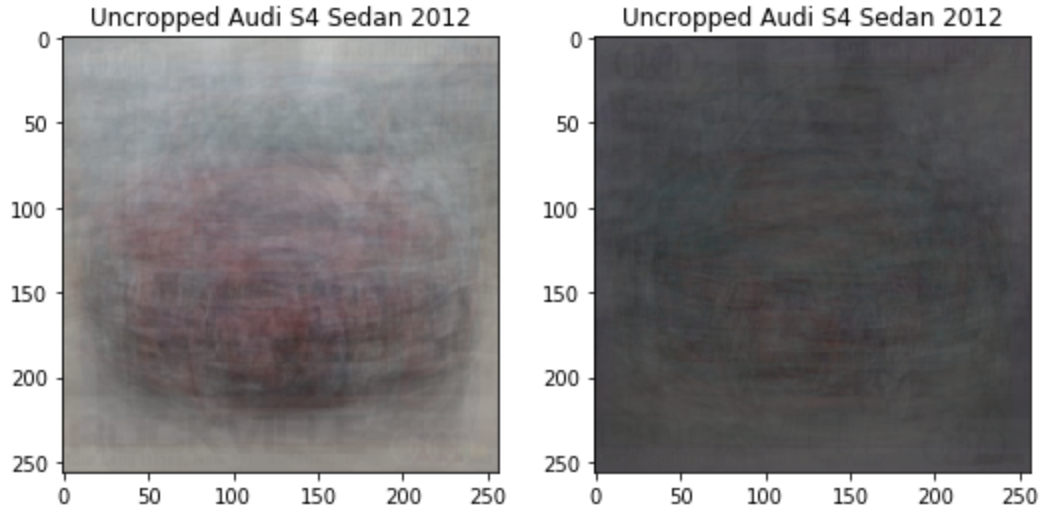


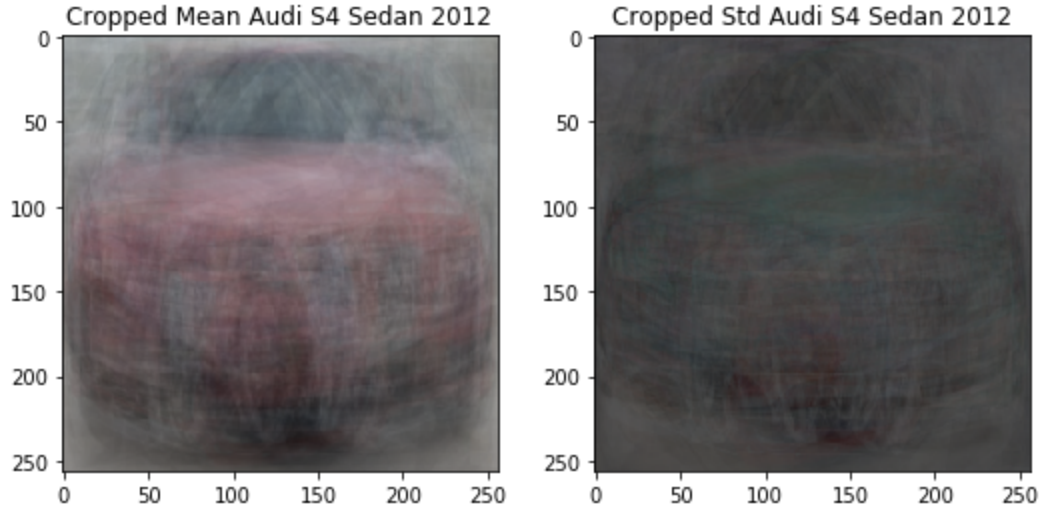
Image 1: For Cropped Acura Integra Type R 2001 set:



Trying the same thing for the cropped and uncropped Audi S4 Sedan 2012 dataset



And for the cropped Audi S4 Sedan 2012 dataset



The mean for the cropped dataset shows better details of “a car” than the uncropped one and hence the similarity in the dataset for that class is better when cropped.

For the standard deviation, the dark parts represent low std pixels values which are close to each other across this class while the bright region show great standard deviation in pixel values in that region.

When you visit the website, you will download the training set in a folder consisting of 8144 images, and the testing set in another folder consisting of 8041. A car_devkit is downloaded that contains :

['cars_train_annos.mat', 'cars_meta.mat', 'train_perfect_preds.txt', 'eval_train.m', 'README.txt', 'Cars_test_annos.mat']

The README.txt file explains these files and here are the important parts

```
Descriptions of the files are as follows:


-cars_meta.mat:

  Contains a cell array of class names, one for each class.


-cars_train_annos.mat:

  Contains the variable 'annotations', which is a struct array of length

  num_images and where each element has the fields:

    bbox_x1: Min x-value of the bounding box, in pixels

    bbox_x2: Max x-value of the bounding box, in pixels

    bbox_y1: Min y-value of the bounding box, in pixels

    bbox_y2: Max y-value of the bounding box, in pixels

    class: Integral id of the class the image belongs to.

    fname: Filename of the image within the folder of images.


-cars_test_annos.mat:

  Same format as 'cars_train_annos.mat', except the class is not provided.
```

However the same dataset with classes folder already available and free from kaggle, you can download it using this link: https://www.kaggle.com/jutrera/stanford-car-dataset-by-classes-folder it contains a zip

file with the train and test folders, anno_test and anno_train csvs and they contain the Imagefile, Bounding boxes and class number.

The available classes are:

```
Classes:

 ['AM General Hummer SUV 2000', 'Acura Integra Type R 2001', 'Acura RL Sedan 2012', 'Acura TL Sedan 2012', 'Acura TL Type-S 2008', 'Acura TSX Sedan 2012', 'Acura ZDX Hatchback 2012', 'Aston Martin V8 Vantage Convertible 2012', 'Aston Martin V8 Vantage Coupe 2012', 'Aston Martin Virage Convertible 2012', 'Aston Martin Virage Coupe 2012', 'Audi 100 Sedan 1994', 'Audi 100 Wagon 1994', 'Audi A5 Coupe 2012', 'Audi R8 Coupe 2012', 'Audi RS 4 Convertible 2008', 'Audi S4 Sedan 2007', 'Audi S4 Sedan 2012', 'Audi S5 Convertible 2012', 'Audi S5 Coupe 2012', 'Audi S6 Sedan 2011', 'Audi TT Hatchback 2011', 'Audi TT RS Coupe 2012', 'Audi TTS Coupe 2012', 'Audi V8 Sedan 1994', 'BMW 1 Series Convertible 2012', 'BMW 1 Series Coupe 2012', 'BMW 3 Series Sedan 2012', 'BMW 3 Series Wagon 2012', 'BMW 6 Series Convertible 2007', 'BMW ActiveHybrid 5 Sedan 2012', 'BMW M3 Coupe 2012', 'BMW M5 Sedan 2010', 'BMW M6 Convertible 2010', 'BMW X3 SUV 2012', 'BMW X5 SUV 2007', 'BMW X6 SUV 2012', 'BMW Z4 Convertible 2012', 'Bentley Arnage Sedan 2009', 'Bentley Continental Flying Spur Sedan 2007', 'Bentley Continental GT Coupe 2007', 'Bentley Continental GT Coupe 2012', 'Bentley Continental Supersports Conv. Convertible 2012', 'Bentley Mulsanne Sedan 2011', 'Bugatti Veyron 16.4 Convertible 2009', 'Bugatti Veyron 16.4 Coupe 2009', 'Buick Enclave SUV 2012', 'Buick Rainier SUV 2007', 'Buick Regal GS 2012', 'Buick Verano Sedan 2012', 'Cadillac CTS-V Sedan 2012', 'Cadillac Escalade EXT Crew Cab 2007', 'Cadillac SRX SUV 2012', 'Chevrolet Avalanche Crew Cab 2012', 'Chevrolet Camaro Convertible 2012', 'Chevrolet Cobalt SS 2010', 'Chevrolet Corvette Convertible 2012', 'Chevrolet Corvette Ron Fellows Edition Z06 2007', 'Chevrolet Corvette ZR1 2012', 'Chevrolet Express Cargo Van 2007', 'Chevrolet Express Van 2007',
```

'Chevrolet HHR SS 2010', 'Chevrolet Impala Sedan 2007', 'Chevrolet Malibu Hybrid Sedan 2010', 'Chevrolet Malibu Sedan 2007', 'Chevrolet Monte Carlo Coupe 2007', 'Chevrolet Silverado 1500 Classic Extended Cab 2007', 'Chevrolet Silverado 1500 Extended Cab 2012', 'Chevrolet Silverado 1500 Hybrid Crew Cab 2012', 'Chevrolet Silverado 1500 Regular Cab 2012', 'Chevrolet Silverado 2500HD Regular Cab 2012', 'Chevrolet Sonic Sedan 2012', 'Chevrolet Tahoe Hybrid SUV 2012', 'Chevrolet TrailBlazer SS 2009', 'Chevrolet Traverse SUV 2012', 'Chrysler 300 SRT-8 2010', 'Chrysler Aspen SUV 2009', 'Chrysler Crossfire Convertible 2008', 'Chrysler PT Cruiser Convertible 2008', 'Chrysler Sebring Convertible 2010', 'Chrysler Town and Country Minivan 2012', 'Daewoo Nubira Wagon 2002', 'Dodge Caliber Wagon 2007', 'Dodge Caliber Wagon 2012', 'Dodge Caravan Minivan 1997', 'Dodge Challenger SRT8 2011', 'Dodge Charger SRT-8 2009', 'Dodge Charger Sedan 2012', 'Dodge Dakota Club Cab 2007', 'Dodge Dakota Crew Cab 2010', 'Dodge Durango SUV 2007', 'Dodge Durango SUV 2012', 'Dodge Journey SUV 2012', 'Dodge Magnum Wagon 2008', 'Dodge Ram Pickup 3500 Crew Cab 2010', 'Dodge Ram Pickup 3500 Quad Cab 2009', 'Dodge Sprinter Cargo Van 2009', 'Eagle Talon Hatchback 1998', 'FIAT 500 Abarth 2012', 'FIAT 500 Convertible 2012', 'Ferrari 458 Italia Convertible 2012', 'Ferrari 458 Italia Coupe 2012', 'Ferrari California Convertible 2012', 'Ferrari FF Coupe 2012', 'Fisker Karma Sedan 2012', 'Ford E-Series Wagon Van 2012', 'Ford Edge SUV 2012', 'Ford Expedition EL SUV 2009', 'Ford F-150 Regular Cab 2007', 'Ford F-150 Regular Cab 2012', 'Ford F-450 Super Duty Crew Cab 2012', 'Ford Fiesta Sedan 2012', 'Ford Focus Sedan 2007', 'Ford Freestar Minivan 2007', 'Ford GT Coupe 2006', 'Ford Mustang Convertible 2007', 'Ford Ranger SuperCab 2011', 'GMC Acadia SUV 2012', 'GMC Canyon Extended Cab 2012', 'GMC Savana Van 2012', 'GMC Terrain SUV 2012', 'GMC Yukon Hybrid SUV 2012', 'Geo Metro Convertible 1993', 'HUMMER H2 SUT Crew Cab 2009', 'HUMMER H3T Crew Cab 2010', 'Honda Accord Coupe 2012', 'Honda Accord Sedan

2012', 'Honda Odyssey Minivan 2007', 'Honda Odyssey Minivan 2012', 'Hyundai Accent Sedan 2012', 'Hyundai Azera Sedan 2012', 'Hyundai Elantra Sedan 2007', 'Hyundai Elantra Touring Hatchback 2012', 'Hyundai Genesis Sedan 2012', 'Hyundai Santa Fe SUV 2012', 'Hyundai Sonata Hybrid Sedan 2012', 'Hyundai Sonata Sedan 2012', 'Hyundai Tucson SUV 2012', 'Hyundai Veloster Hatchback 2012', 'Hyundai Veracruz SUV 2012', 'Infiniti G Coupe IPL 2012', 'Infiniti QX56 SUV 2011', 'Isuzu Ascender SUV 2008', 'Jaguar XK XKR 2012', 'Jeep Compass SUV 2012', 'Jeep Grand Cherokee SUV 2012', 'Jeep Liberty SUV 2012', 'Jeep Patriot SUV 2012', 'Jeep Wrangler SUV 2012', 'Lamborghini Aventador Coupe 2012', 'Lamborghini Diablo Coupe 2001', 'Lamborghini Gallardo LP 570-4 Superleggera 2012', 'Lamborghini Reventon Coupe 2008', 'Land Rover LR2 SUV 2012', 'Land Rover Range Rover SUV 2012', 'Lincoln Town Car Sedan 2011', 'MINI Cooper Roadster Convertible 2012', 'Maybach Landaulet Convertible 2012', 'Mazda Tribute SUV 2011', 'McLaren MP4-12C Coupe 2012', 'Mercedes-Benz 300-Class Convertible 1993', 'Mercedes-Benz C-Class Sedan 2012', 'Mercedes-Benz E-Class Sedan 2012', 'Mercedes-Benz S-Class Sedan 2012', 'Mercedes-Benz SL-Class Coupe 2009', 'Mercedes-Benz Sprinter Van 2012', 'Mitsubishi Lancer Sedan 2012', 'Nissan 240SX Coupe 1998', 'Nissan Juke Hatchback 2012', 'Nissan Leaf Hatchback 2012', 'Nissan NV Passenger Van 2012', 'Plymouth Neon Coupe 1999', 'Porsche Panamera Sedan 2012', 'Ram C-V Cargo Van Minivan 2012', 'Rolls-Royce Ghost Sedan 2012', 'Rolls-Royce Phantom Drophead Coupe Convertible 2012', 'Rolls-Royce Phantom Sedan 2012', 'Scion xD Hatchback 2012', 'Spyker C8 Convertible 2009', 'Spyker C8 Coupe 2009', 'Suzuki Aerio Sedan 2007', 'Suzuki Kizashi Sedan 2012', 'Suzuki SX4 Hatchback 2012', 'Suzuki SX4 Sedan 2012', 'Tesla Model S Sedan 2012', 'Toyota 4Runner SUV 2012', 'Toyota Camry Sedan 2012', 'Toyota Corolla Sedan 2012', 'Toyota Sequoia SUV 2012', 'Volkswagen Beetle Hatchback 2012', 'Volkswagen Golf Hatchback 1991', 'Volkswagen Golf Hatchback 2012', 'Volvo 240 Sedan 1993',

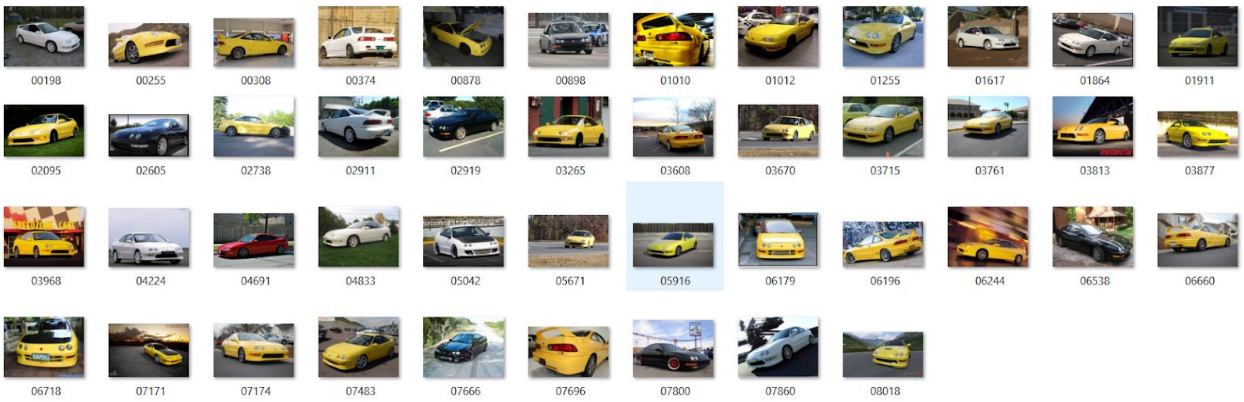
'Volvo C30 Hatchback 2012', 'Volvo XC90 SUV 2007', 'smart fortwo Convertible 2012']

They are mostly balanced with very small variation available between classes which is not a very big difference in general.

Some pictures were covered with advertisement and watermarks which added some noise but were left to add some realism to the dataset.

Exploratory Visualization

In this section I would like to show my dataset, there were two choices to go for. Either use the raw images with its background of roads, signs, and pavements along with other factors or crop the images of the cars as much as possible and using the cars' body with minimum background information. I used raw images first and these are some examples of some classes.



And here are some classes with cars cropped:

Algorithms and Techniques

In the image classification problem, Convolutional Neural Network (CNN) is the most recommended and used algorithm, a classifier is trained on a dataset of large sets of images as a way to teach the classifier to recognize a certain shape (e.g car) by adjusting the neural network weight through gradient descent to minimize the loss function.

Usually a large dataset is used to train the classifier in order to obtain good results, however in my case, the dataset contains very few samples per class which will prove challenging . In this project, the Stanford car dataset was utilized, it was used to create two different models, my own architecture and a pretrained model. For both models a 0.3 train:test split was used, the batch size was kept at 32, ImageDataGenerator was used and its parameters were (rescale=1./255, zoom_range=0.25, rotation_range = 90, horizontal_flip=True, validation_split = 0.3). Flow_from_directory was used to generate the data with the following parameters('My_Cars/trainCropped', target_size=(128, 128), batch_size=32, class_mode='categorical')

The final CNN model had 5 layers , first layer had 32 filters, second layer had 128 filters, third layer had 256 layers and the fourth and fifth had 512 layers. Batch Normalization was added between the convolutional layer and activation layer, Grid search was used to tune the filter size, kernel, pool size and strides of the model. RMSprop optimizer was used and its parameters were kept at default with learning rate kept at 0.001, momentum at 0.9. Epochs were set to 200 and the steps per epoch was 182 (number of train samples / batch size) and steps validation was 74 (number of validation samples / batch size)

Benchmark

My benchmark model will be comparing my accuracy results with results found in Yaqoob, Hashir, Shaharyar Bhatti, and Rana Raees Ahmed Khan. "Car Make and Model Recognition using Image Processing and Machine Learning." The authors followed two approaches; the first approach was to use Bag-Of-words with SURF supported by Multi-class SVM. In the second approach, it is seen that they use CNN for the extraction of features in training, they also used transfer learning on pre-trained AlexaNet network. They also used accuracy as their metric evaluation method in their paper for model evaluation.

Another benchmark that I would like to propose is “Valev, K., Schumann, A., Sommer, L. and Beyerer, J., 2018, April. A systematic evaluation of recent deep learning architectures for fine-grained vehicle classification. In Pattern Recognition and Tracking XXIX (Vol. 10649, p. 1064902). International Society for Optics and Photonics.” which explores many ways to deal with pre-trained model on the stanford car dataset.

III. Methodology

Data Preprocessing

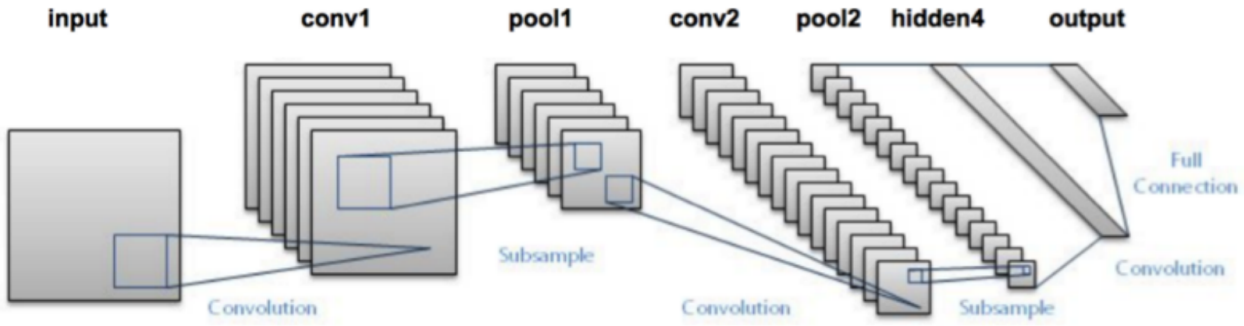
For the dataset provided from the Stanford website, a folder for each car model was created based on the name in the CSV file, however, using the same dataset classified already by classes from the kaggle website would be easier for anyone who wants to work on the same dataset.

First, the images were called into the code by using ImageDataGenerator. It was used with rescale=1./255, zoom_range=0.25, rotation_range = 90, horizontal_flip=True as augmentation factors and validation_split = 0.3 for a validation split. Along with flow_from_directory, not only does it help extensively in the splitting and augmentation process but it saves up with the memory as it does not load the whole dataset (about 18k images) in one run, however, it utilizes the transfer of images and the algorithm to avoid any memory issues by loading the images in batches of 32.

The target size was set either target_size=(256,256) or target_size=(128,128), 256 sized images provided better details for the CNN layers to understand but proved to be big and sometimes too big for the pre-trained models like the ResNet50 which requires 224x224 default size, however it can be changed in the building process.

For the first 23 trials, the training dataset was not cropped, however after that, a code was written to utilize the bounding boxes available in the CSV files to crop the car and minimize background noise.

Implementation



This was the basic architecture I had in mind, it was refined several times throughout the process of this code. I had modified a lot in the main codes to seek the highest accuracy percentages in my own architectures before using a pre-trained model. The input was kept constant with the filter and input size as much as possible and only changed once in a several layer architecture that I made where it caused a OEM memory problem. I shall be talking briefly in the refinement section to show my evolution process from a 5% accuracy code to about 30%.

The following are steps used for creating and training the model:

- Create the model with the following layers:
    1. Convolution 2D layer with 32 filters, 4 kernel size, 1 stride, random normal kernel initializer, same padding and input shape of 128, 128, 3
    2. Batch normalization
    3. Activation layer (Leaky ReLU)
    4. Max Pool 2D with 2 pool size and 2 stride
    5. Convolution 2D layer with 128 filters, 2 kernel size, 1 stride, random normal kernel initializer and same padding
    6. Batch normalization
    7. Activation layer (Leaky ReLU)
    8. Max Pool 2D with 2 pool size and 2 stride
    9. Convolution 2D layer with 256 filters, 2 kernel size, 1 stride, random normal kernel initializer and same padding
    10. Batch normalization
    11. Activation layer (Leaky ReLU)
    12. Max Pool 2D with 2 pool size and 2 stride
    13. Dropout 0.5

14. Convolution 2D layer with 512 filters, 2 kernel size, 1 stride, random normal kernel initializer and same padding
15. Batch normalization
16. Activation layer (Leaky ReLU)
17. Max Pool 2D with 2 pool size and 2 stride
18. Dropout 0.5
19. Convolution 2D layer with 512 filters, 2 kernel size, 1 stride, random normal kernel initializer and same padding
20. Batch normalization
21. Activation layer (Leaky ReLU)
22. Max Pool 2D with 2 pool size and 2 stride
23. Dropout 0.5
24. Global Average Pooling 2D
25. Dense layer with number of outputs 196 and softmax activation
26. Checkpointer was used to save the best weights, avoiding overfitted models, the model was trained for 200 epochs, 182 steps per epoch and 74 validation steps.
27. After training the model was used to predict the labels of the testing dataset and calculated the testing accuracy of the model

The VGG16, DenseNet121 and ResNet50 were used as pre-trained models on the imagenet dataset, and were used to predict the labels. Also accuracy was calculated for each model.

Refinement

My Architecture

I have prepared two folders History_accuracy and Saved_Models, in Saved_Model best saved models will be found along with a file namd what model each represent and in it, there will be the change and difference from the previous architecture and in the History_accuracy I saved the first 13 epoch visualization for the report.

In my first architecture I had the model shown in the figure below

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 128, 128, 32)	416
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 32)	0
dropout_1 (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 31, 31, 64)	8256
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 64)	0
dropout_2 (Dropout)	(None, 8, 8, 64)	0
conv2d_3 (Conv2D)	(None, 7, 7, 128)	32896
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 128)	0
dropout_3 (Dropout)	(None, 2, 2, 128)	0
conv2d_4 (Conv2D)	(None, 2, 2, 256)	131328
max_pooling2d_4 (MaxPooling2	(None, 1, 1, 256)	0
dropout_4 (Dropout)	(None, 1, 1, 256)	0
global_average_pooling2d_1 (	(None, 256)	0
dense_1 (Dense)	(None, 196)	50372

Total params: 223,268
Trainable params: 223,268
Non-trainable params: 0

And reported loss: 5.1737 - acc: 0.0100 - val_loss: 5.1683 - val_acc: 0.0075 in the 9th epoch, by then I didn't know that I have to use more than 10 to 20 epochs to train my network and kept my training at 10.

In the second trial i removed the dropouts and reported loss: 5.1397 - acc: 0.0150 - val_loss: 5.1068 - val_acc: 0.0288 .

In the third trial i continued without dropouts and added 50 steps per epoch in my fit_generator and reported loss: 5.1118 - acc: 0.0125 - val_loss: 5.0812 - val_acc: 0.0163.

By then I realised i needed to increase the number of epochs, so i increased it to 30 epochs and started playing in the fit parameters, in the fourth trial loss: 4.7058 - acc: 0.0537 - val_loss: 4.8947 - val_acc: 0.0458.

In the fifth trial i used 100 steps per epoch, val steps = 60, batch 32, removed last cnn layer due to overfit and set kernel is set to 4 and recorded loss: 4.6426 - acc: 0.0597 - val_loss: 4.8752 - val_acc: 0.0543 in the 25th epoch.

As it can be seen the validation accuracy was quite low for the first few trials, and was always under the 6% margin for the validation accuracy.

After that I used random search for my architecture, and since it was random, I felt missing on some combinations may not be beneficial and hence I changed it to grid search, however I had to split my code into 4 parts, running each part separately because 70% through my trial I would get full memory error. Part 1 and 2 were responsible for the Filters and kernel grid search and yielded :

32 Filters 6 Kernels = 4.780 Val_Loss as the best combination for the architecture. Part 3 and 4 were responsible for Pool_size and Strides and yielded:

4 pool, 2 strides 4.795 Val_Loss as the best combination for the filters and kernels previously found. The results and code can be found in GridSearchCodeModified, Results in grid_searchForStrideAndPool and Grid Search Results (for filters and kernels).

I applied my results to trials 13 ,14 and 15 and yielded

loss: 4.9971 - acc: 0.0325 - val_loss: 4.9904 - val_acc: 0.0324 for trial 13

loss: 5.2822 - acc: 0.0066 - val_loss: 5.2546 - val_acc: 0.0107 for trial 14

loss: 4.8754 - acc: 0.0394 - val_loss: 4.8996 - val_acc: 0.0438 for trial 15

Starting from trial 16 till the end I decided to save the codes with its results so that It becomes easier to track the code and changes in it, naming them with the change without the need to check on folders.

Uncropped dataset was very slow to train taking up to 200 seconds per epoch and achieving :

loss: 4.6007 - acc: 0.0591 - val_loss: 5.0139 - val_acc: 0.0383 after 18 epoch

However the cropped dataset took an average of 150 seconds for each epoch and achieving :

loss: 4.0291 - acc: 0.1263 - val_loss: 4.7155 - val_acc: 0.0623

And hence I used the cropped dataset for the rest of my work to decrease train time and have a faster whole algorithm.

I will skip ahead and represent the best code I have reached as my own architecture with its results.

That would be trial 39 and i shall go briefly through it, first I present the architecture summary:

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 128, 128, 32)	1568
batch_normalization_6 (Batch	(None, 128, 128, 32)	128
leaky_re_lu_6 (LeakyReLU)	(None, 128, 128, 32)	0
max_pooling2d_6 (MaxPooling2	(None, 64, 64, 32)	0
conv2d_7 (Conv2D)	(None, 64, 64, 128)	16512
batch_normalization_7 (Batch	(None, 64, 64, 128)	512
leaky_re_lu_7 (LeakyReLU)	(None, 64, 64, 128)	0
max_pooling2d_7 (MaxPooling2	(None, 32, 32, 128)	0
conv2d_8 (Conv2D)	(None, 32, 32, 256)	131328
batch_normalization_8 (Batch	(None, 32, 32, 256)	1024
leaky_re_lu_8 (LeakyReLU)	(None, 32, 32, 256)	0
max_pooling2d_8 (MaxPooling2	(None, 16, 16, 256)	0
dropout_4 (Dropout)	(None, 16, 16, 256)	0
conv2d_9 (Conv2D)	(None, 16, 16, 512)	524800
batch_normalization_9 (Batch	(None, 16, 16, 512)	2048
leaky_re_lu_9 (LeakyReLU)	(None, 16, 16, 512)	0
max_pooling2d_9 (MaxPooling2	(None, 8, 8, 512)	0
dropout_5 (Dropout)	(None, 8, 8, 512)	0
conv2d_10 (Conv2D)	(None, 8, 8, 512)	1049088
batch_normalization_10 (Batc	(None, 8, 8, 512)	2048
leaky_re_lu_10 (LeakyReLU)	(None, 8, 8, 512)	0
max_pooling2d_10 (MaxPooling	(None, 4, 4, 512)	0
dropout_6 (Dropout)	(None, 4, 4, 512)	0
global_average_pooling2d_2 (	(None, 512)	0
dense_2 (Dense)	(None, 196)	100548

Total params: 1,829,604
Trainable params: 1,826,724
Non-trainable params: 2,880

In this architecture, I have used 5 layers with the following attributes

Filters = 32 then 128 then 256 then 512 then 512

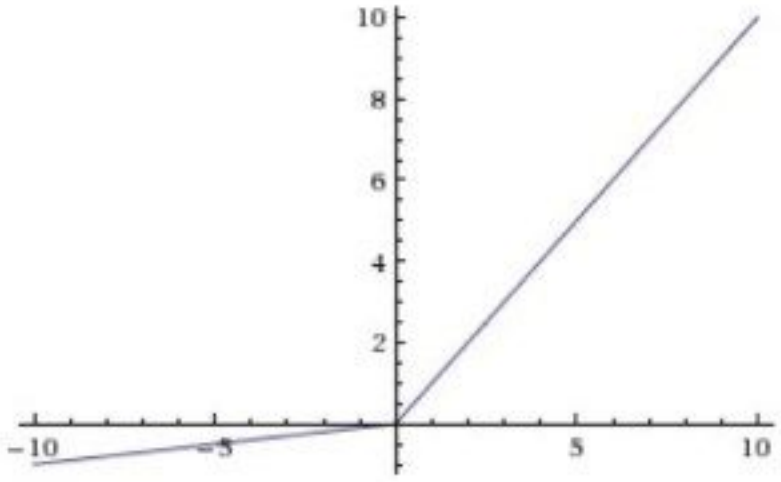
Kernel = 4 then the rest kept at 2

Pool_size and strides kept at 2 for the whole architecture

Used LeakyReLU asd activation function and that's after reading through papers from 6 to 10 from the reference. And here is what I understood for PReLU and LReLU

LReLU activation function

Leaky ReLU is a modification of ReLU which replaces the zero part of the domain in [-∞,0] by a low slope, as we can see in the figure and formula below.



The motivation for using LReLU instead of ReLU is that constant zero gradients can also result in slow learning, as when a saturated neuron uses a sigmoid activation function. Furthermore, some of them may not even activate. This sacrifice of the zero-sparsity, according to the authors, can provide worse results than when the neurons are completely deactivated (ReLU) [7]. In fact, the authors report the same or insignificantly better results when using PReLU instead of ReLU.

PReLU activation function

Parametric ReLU [8] is a inspired by LReLU which, as mentioned before, has negligible impact on accuracy compared to ReLU. Based on the same ideas that LReLU, PReLU has the same goals: increase the learning speed by not deactivating some neurons. In contrast with LReLU, PReLU

substitutes the value 0.01 by a parameter a_i where i refers to different channels. One could also share the same values for every channel

Based on [11] I used the BatchNormalization() layer between the linear and non-linear layers, because it normalizes the input to the activation function, so that it is centered in the linear section of the activation function.

I ended the 5 layers with a GlobalAveragePooling2D instead of flatten() because It applies average pooling on the spatial dimensions until each spatial dimension is one, and leaves other dimensions unchanged. In this case values are not kept as they are averaged.

The usual dense layer with 196 classes and softmax activation, kept the optimizer at default rmsprop (0.001) and loss function as categorical cross entropy with accuracy metric.

For the fitting part I used

```
steps_per_epoch=182,  # train size // batch

epochs=epochs, #200 epochs

verbose=1,

validation_data=test_data,

validation_steps=74 # validation // batch
```

The Results will be discussed in the Results section

Pre-Trained Architecture

I used VGG16, DenseNet121 and ResNet50

ResNet showed the best validation loss and accuracy so I went on and decreased the RMSprop LR to 0.0001 for better results.

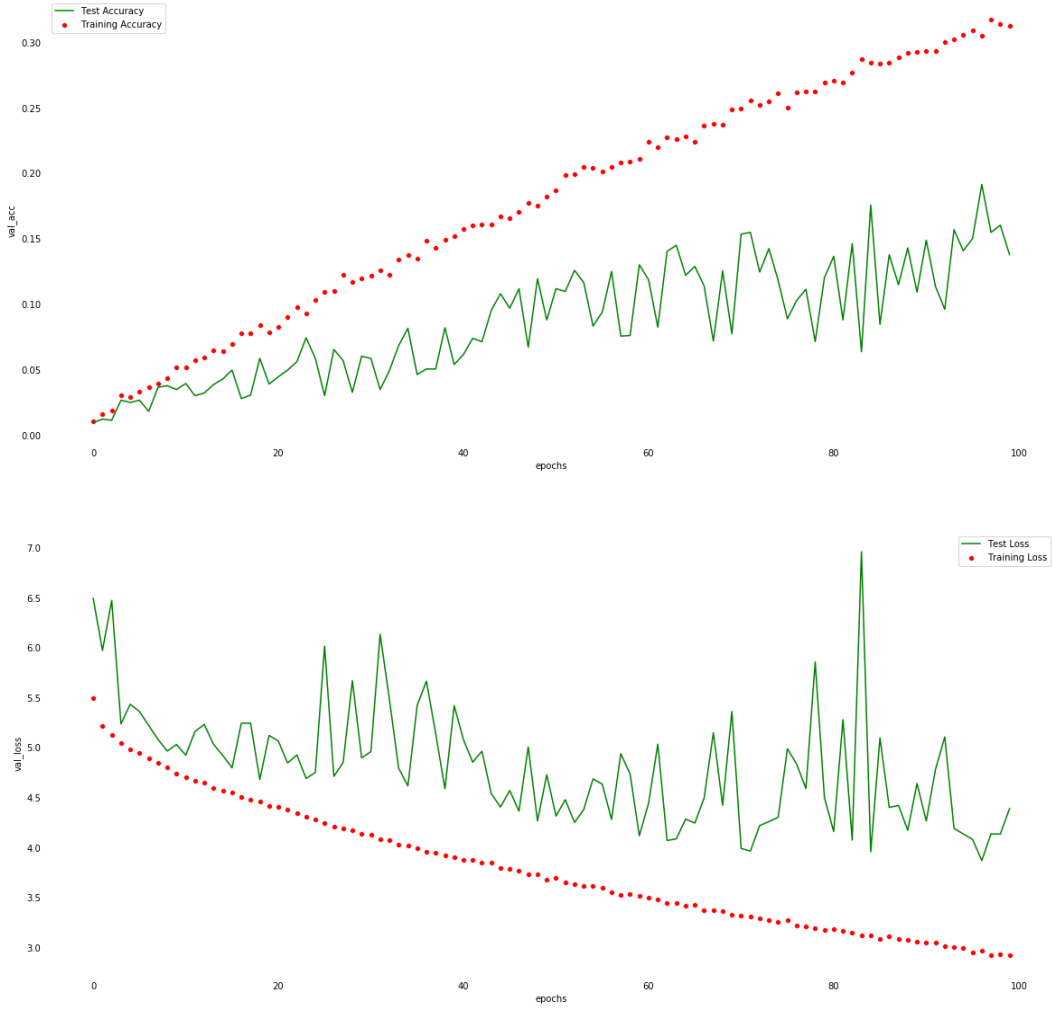
IV. Results

Model Evaluation and Validation

My Architecture (non pretrained)

For my best architecture, the chosen model was

Five layer with filters sizes (32, 128, 256, 512,512) consecutively, activation layer with Leaky ReLU. because this achieved the highest accuracy of 26.15% which is higher than a random guess model which would score 0.5%



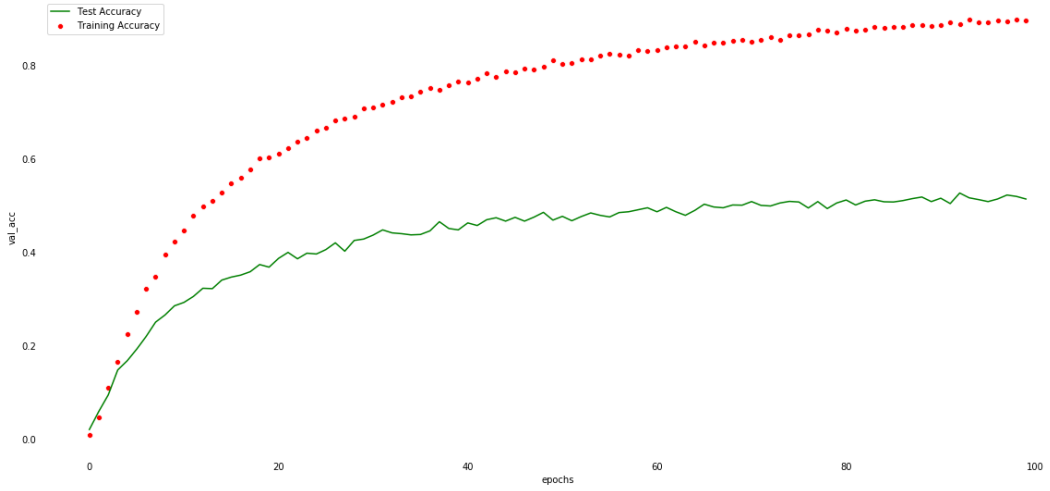
For the Pre-Trained model

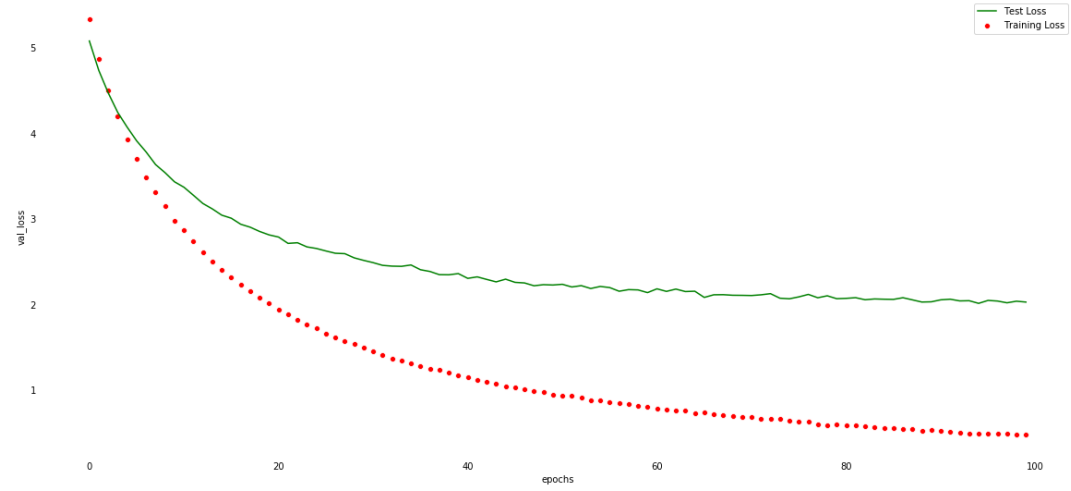
I compared VGG16, Dense121 and ResNet50 with the same architecture for all.

VGG16 gave 33.48% accuracy at the 193rd epoch

DenseNet121 gave 43.99% accuracy at the 44th epoch

ResNet50 gave the best accuracy with 51.32% at the 95th epoch





Justification

For the first benchmark, They obtained above 90% accuracy using neural networks as compared to the 75% average accuracy of BOF model, however they used logo detection + shape not just shape of the car

In the second paper explores the pre-trained models and augmentation methods that can be used in the same dataset that this project is using and the results are:

Architecture	Approach	Accuracy
ResNet-50	from scratch	84.3%
ResNet-50	fine-tune	**92.0%**
ResNet-152	from scratch	35.3%
ResNet-152	fine-tune	**92.6%**

Table 1: Differences between fine-tuning a model and training from scratch

Model	Augmentations	Accuracy
VGG16	-	83.7%
VGG16	Flip	86.6%
VGG16	Flip, Rotation	86.6%
VGG16	Flip, Noise	86.7%
VGG16	Flip, Noise, Rotation	86.7%
VGG16	Flip, Motion blur	**86.8%**

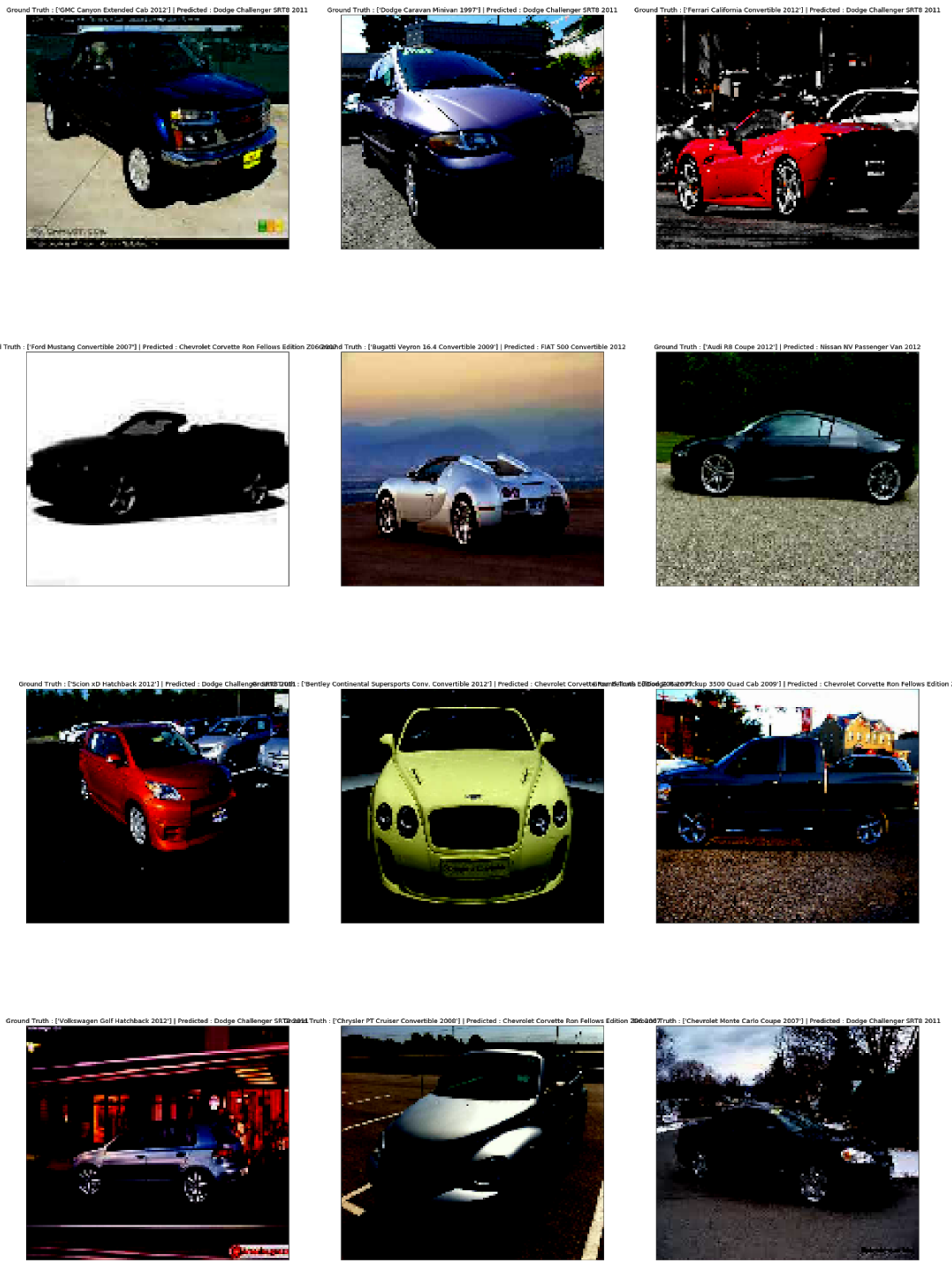
Table 2: Augmentation strategies effect on model accuracy

My Architecture reached a maximum test accuracy of 26.15% and the ResNet50 reached a test accuracy of 51.32% which is lower than the accuracies reached by the papers.
My Architecture performed better than a random guess model which would give an accuracy of 0.5%, and for the pretrained ResNet50 model, it could be used for a simple car parking finding system, where very high accuracies are not needed. I believe that I didn't reach a high accuracy due to the few number of samples per classes available in my dataset.

V. Conclusion

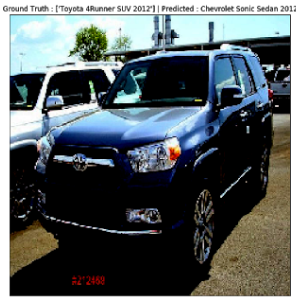
Free-Form Visualization

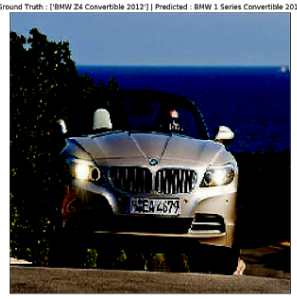
For my architecture these were the results

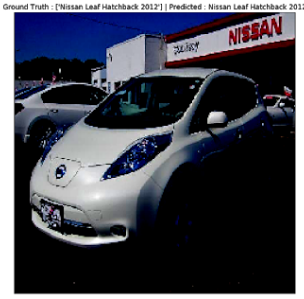


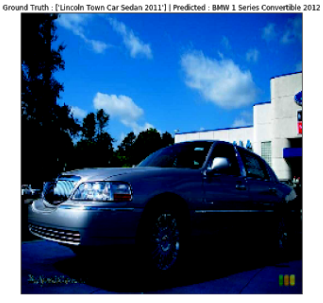
In those 16 images, none were correct with the ground truth

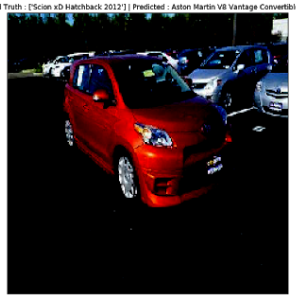
For the pretrained model, these were the results





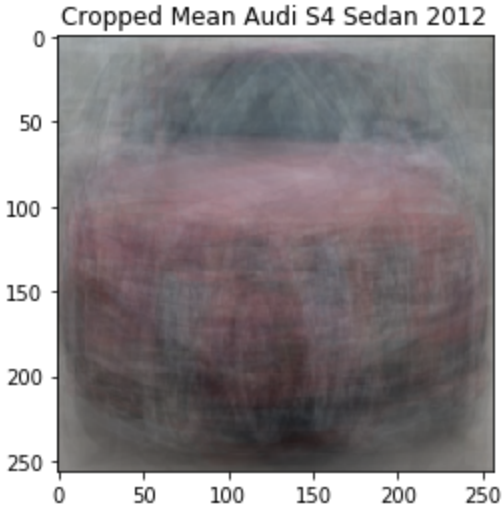


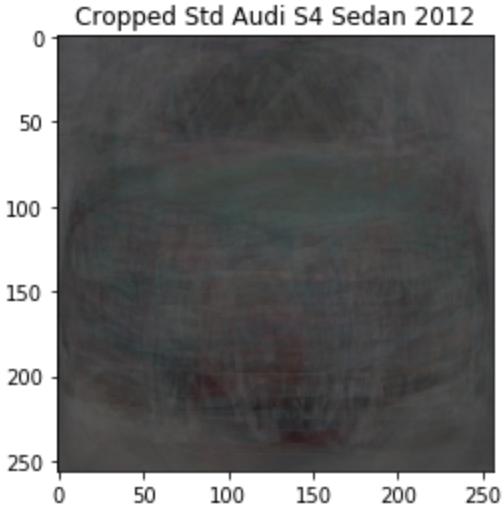




Only one of the 16 was correctly labeled, 2 had the class right but the model wrong .

It is obvious that my model was not able to score well on the testing dataset, which means it has overfitted on the training dataset. This may be due to the noise present in the dataset and the fact that there are very few samples per class available.





The mean shows that there isn't enough details visible, which means that there is a low similarity between the samples in the class.

For the standard deviation, the dark parts represent low std pixels values which are close to each other across this class while the bright region show great standard deviation in pixel values in that region, and as can be seen it is difficult to detect that from the std picture shown above.

Reflection

The project ran on:

Processor: Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz (4 CPUs), ~3.0GHz

Memory: 8192MB RAM ~ 8 GB

Card name: NVIDIA GeForce GTX 1060 3GB

The steps that were taken:

1. Import libraries
2. Download the Stanford car dataset
3. Crop the pictures using Car_Extractor
4. Augment the data
5. Many models were created to achieve the highest accuracy scores from scratch
6. Train a lot
7. Pre-trained model was used which was very useful and boosted accuracy
8. Test the models and get predictions

In my personal opinion, the training part was the most difficult part throughout the entire project, especially the long waiting hours, due to the limitation of my hardware, many algorithms weren't tested and many hours wasted on weak algorithms. The fun part for was trying my own architectures (apart from the long training hours), to see the fruit of your ideas and the improvement in your work through every code was very enjoyable. Using pre-trained models was very interesting especially when I had my money on VGG16 but ResNet50 proved to be better with my dataset.

Improvement

I shall not stop trying my architecture out for this problem, and try to reach a whole new record with it, I shall invest in a better hardware or try one of the available clouds (though running costs are expensive with currency exchange). I shall try to use more of the testing set in the training as, 51/49 split is too much for the training split in hopes it would be better. More target sizes will be tested and deeper layers shall be tested with the better hardware. I shall learn advance way to build an architecture, talk to machine learning professional and competitions winners to have an idea of how to work more upon my dataset. A better dataset shall be utilized such as the VMMR dataset which is 12 GB in size in hopes to increase the accuracy.

VI. References

[1] Ullah, Ihsan. (2017). Vehicle Make and Model Recognition System based on Constitutional Neural Network.

[2] Yaqoob, Hashir, Shaharyar Bhatti, and Rana Raees Ahmed Khan. "Car Make and Model Recognition using Image Processing and Machine Learning."

[3] Emami, Hajar, Mahmood Fathi, and Kaamran Raahemifar. "Real time vehicle make and model recognition based on hierarchical classification." International Journal of Machine Learning and Computing 4.2 (2014): 142.

[4] Biglari, Mohsen & Soleimani, Ali & Hassanpour, H. (2017). Vehicle Make and Model Recognition using Auto Extracted Parts.

[5] Krause, Jonathan, et al. "3d object representations for fine-grained categorization." Proceedings of the IEEE International Conference on Computer Vision Workshops. 2013.

[6] Nair V. & Hinton G.E. 2010. “Rectified Linear Units Improve Restricted Boltzmann Machines”

[7] Maas A., Hannun A.Y & Ng A.Y. 2013. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”

[8] He K., Zhang X., Ren S. & Sun J. 2015. “Delving Deep Into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”

[9] Xu B., Wang N., Chen T. & Li M. 2015. “Empirical Evaluation of Rectified Activations in Convolutional Network”

[10] Clevert D.A., Unterthiner T. & Hochreiter S. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)

[11] Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.