Car Detection with Make and Model Classification

Chizuram Nwosu, Brian Law, Harshil Suthar
The William States Lee College of Engineering
University of North Carolina at Charlotte
Charlotte, North America
Email: cnwosu@uncc.edu, blaw5@uncc.edu, hsuthar1@uncc.edu

***Abstract*— With the increase in computational capacity of modern hardware (GPU's) and big data collection, computer vision algorithms have become more robust than ever before. Algorithms such as faster RCNN and YoloV4 are capable of object detection and classification in real time, whereas classification models like ResNet, DenseNet, and InceptionNet have demonstrated great accuracy in image classification. Hence, for the task of car detection and classification of its model name and year, both object detection and classification algorithm are applied in sequence. The YoloV3 model is used for car detection while DenseNet161 is used to get the model name and year of the detected car.**

***Index Terms*—YoloV3, DenseNet161, ResNet34, MobileNetV2, GoogleNet**

I. Introduction and Motivation

The field of computer vision has shown exponential growth in recent past. It has been an area of great interest for both industrial application and academic research in the last few years. The impact of computer vision applications on human routine and lifestyle is more prominent than ever before. This technology makes computers and edge devices process the given information and infer the result, same in manner as we humans do. Computer vision applications at their core are neural network models. Every model type is made up of hidden layers arranged in a specific order. For a given model with hidden layers, convolution layers are responsible for feature extraction from various receptive fields of the input image. This is separate from dense fully connected layers who perform the task of classification and prediction based on extracted features from image. This technology existed back from 1960-70's but was constrained by the amount available training data and computational power. Now there are better data recording sensors, edge devices and facilities that are available to store collected data in the form of datasets. Along with that, inexpensive and fast parallel processing (required for large number of MAC operations) from modern GPUs, make the training process for computer vision applications more efficient and reliable.

Currently, this technology is so precise that it has been deployed across various industries for performing vital operations. For example, computer vision has been used by FinTech for check deposition process, biometric authorization on mobile phones, in automatic control of automobiles, and in examination of cancer like disease in healthcare sector. Hence, computer vision technology provides powerful capabilities to edge devices which offer computers inferring capabilities as we humans do.

Looking at the impact of computer vision and machine learning applications on our daily routine, along with their consistent performance, it is apparent that it has immense potential to be a solution for problems across different domains. Detecting the quantity, type classification and localization of automobiles from any given image is one such problem which can be resolved by computer vision. Its application can be very effective for highway traffic authorities and other businesses. The report drafted in this paper summarizes our project work of detecting cars in the given image frame and identifying the make and model of that car. Various applied models will individually localize any detected object in image, classification which object is a car or not, and further classify the car into its respect vehicle type, containing its make and model status.

II. Approach

To achieve the objective of this project, different CNNs are trained for make year and car model classification. The best performing CNN will then be connected with another CNN that will handle the object localization and classification that will determine whether or not a car is located in an image. For the make year and model CNN, the following CNNs were considered, analyzed, and compared to determine which model performed best.

Resnet34, Densenet161, MobileNetv2 and GoogleNet are different models used for image classification. The GoogleNet architecture consists of stacking multiple Inception blocks with occasional max pooling to reduce the height and width of the feature maps. An Inception block applies 1x1, 3x3, and 5x5 convolutions, and a max pool operation which allows the network to look at the same data with different receptive fields from the input image. The overall ResNet architecture consists of stacking multiple ResNet blocks, of which some are down-sampling the input. The pre-activation ResNet block applies the non-linearity before the skip connection. One difference to the GoogleNet training is that SGD with Momentum is used as the optimizer instead of the Adam optimizer.

DenseNet uses Dense Layers, Dense Blocks and Transition Layer to provide an efficient way of reusing features by having each convolution depends on all previous input features, but add only a small number of filters to it. MobileNetv2 uses residual connections at the bottleneck layers. This inverted

residual structure has lightweight depth-wise convolutions to extract the main features as a source of non-linearity. Each one of these models follows different architectures to extract features from an image. They are all pretrained from the ImageNet dataset with 200 labels. For this project, the Cars196 dataset is used to detail the label of cars into its appropriate make and model. Once this project dataset is formatted to run through these different models, the best performing model will be chosen as the main CNN for classification.

Then a variety of object detection algorithms and models were researched to determine which will be used with the spot which object in the image is a car or not. Notable options were; Fast R-CNN, Faster R-CNN, Histogram of Oriented Gradients (HOG), Sliding Window, Region-based Convolutional Neural Networks (R-CNN), Region-based Fully Convolutional Network (R-FCN), Single Shot Detector (SSD), Spatial Pyramid Pooling (SPP-net). The functionality of YoloV3 showed promise when analyzed alongside the previous options. YoloV3 incorporates classification and localization in one shot and was consistently accurate. It will be integrated with the chosen classification model to form the full machine learning system for this project.

III. Dataset and Training Setup

Selection of appropriate dataset is a crucial step for any machine learning project. Car196 is the dataset selected for this project. It consists of a total of 16,185 images (8041 testing and 8144 training images). Where each class is divided into a ratio of 50-50 for training and testing. In addition to that, every class is labeled on basis of Model name, company name and production year. The default dimensional resolution for images in dataset is 360x240 pixels. So, for efficient preprocessing and feeding it into various classification models, all the images were trimmed down to 240x240 pixels resolution.

Further for training different models google co-lab platform is utilized. Initially for preprocessing and training of models like MobileNet, Google Co-lab pro was used. But for heavier models like ResNet, DenseNet, and GoogleNet, a switch was made to use Google Co-Lab Pro Plus as the model training demanded more runtime RAM. YoloV3 was already pretrained with the ImageNet dataset and was used without any internal edits to its functionality.

The car detection flowchart helps illustrate the process of images being sent through the YoloV3 for preliminary car selection. The discovered cars will be run through the four different CNNs to find out which CNN will accurately label the appropriate make year and model of the car.

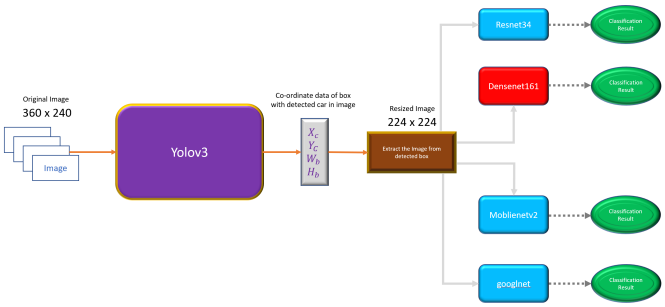

Fig. 1. Car Detection Flowchart

IV. Results and Analysis

A. ResNet34

$LearningRates$	$1e-2$
$1(epoch)$	5.20398
$2(epochs)$	1.65546
$3(epochs)$	1.80844
$4(epochs)$	0.35518
$5(epochs)$	0.51135
$6(epochs)$	0.50849
$7(epochs)$	0.17633
$8(epochs)$	0.26775
$9(epochs)$	0.36398
$10(epochs)$	0.21061

$Time$	$ValAcc$	$TestAcc$
$1HR:12M:68S$	77	80

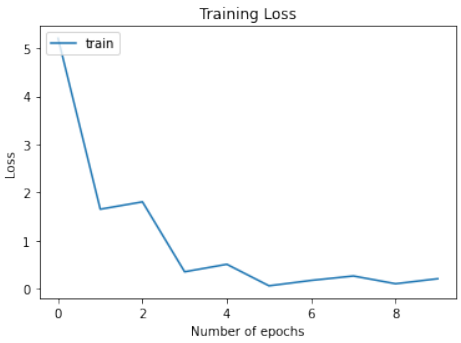

Fig. 2. ResNet34 Training Loss

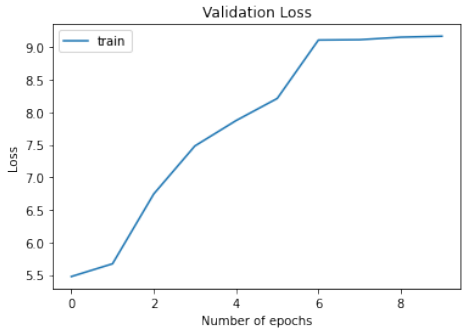

Fig. 3. ResNet34 Validation Loss

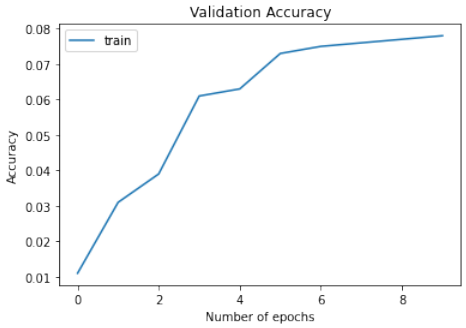

Fig. 4. ResNet34 Validation Accuracy

When using ResNet34, both the training and validation loss experience a noticeable saturation at around the same time, despite training loss having a decreasing trend over time as the validation loss is increasing during that time. These figures show the steady progression of the ResNet34 model becoming more accurate up until 8 epochs, where its accuracy plateaus.

B. DenseNet161

$LearningRates$	$1e-2$
$1(epoch)$	5.9374
$2(epochs)$	2.92588
$3(epochs)$	0.6393
$4(epochs)$	0.39702
$5(epochs)$	0.11396
$6(epochs)$	0.10713
$7(epochs)$	0.10475
$8(epochs)$	0.04458
$9(epochs)$	0.05706
$10(epochs)$	0.03031
$11(epochs)$	0.04757
$12(epochs)$	0.02853
$13(epochs)$	0.04431
$14(epochs)$	0.06152
$15(epochs)$	0.04123

$Time$	$ValAcc$	$TestAcc$
$18HR:25M:08S$	78	85

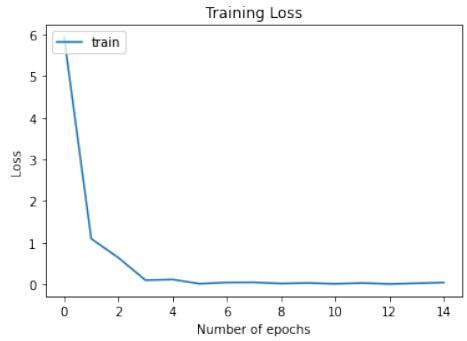

Fig. 5. DenseNet161 Training Loss

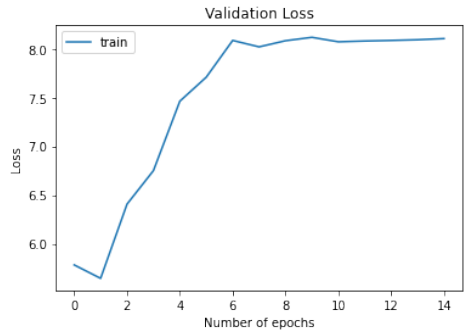

Fig. 6. DenseNet161 Validation Loss

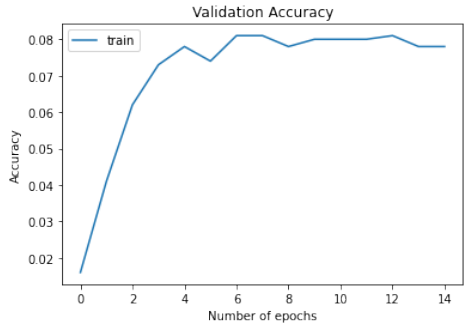

Fig. 7. DenseNet161 Validation Accuracy

Fig. 8. DenseNet161 Correct Predictions

The DenseNet161 model used, at one point in the training process, over 40 gigabytes of ram. The large amount of ram

required a Google Colaboratory Pro Plus account running on the CPU which held a limit of 51 gigabytes. The significant increase in training time can be attributed to running on the CPU and the increase in complexity in the DenseNet model.

The model converged at around 4 epochs, as seen through Figures 5, 6, 7, and held the greatest test accuracy of 85%, shown through Figure 8.

C. MobileNetV2

$LearningRates$	$1e-2$
$1(epoch)$	5.20777
$2(epochs)$	5.31726
$3(epochs)$	2.28317
$4(epochs)$	2.12671
$5(epochs)$	0.8382
$6(epochs)$	0.90102
$7(epochs)$	1.04931
$8(epochs)$	0.61112
$9(epochs)$	0.85799
$10(epochs)$	0.50751
$11(epochs)$	0.76895
$12(epochs)$	0.43352
$13(epochs)$	0.67657
$14(epochs)$	0.93879
$15(epochs)$	0.59922
$20(epochs)$	0.67391

$Time$	$ValAcc$	$TestAcc$
$0HR:59M:25$	71	74

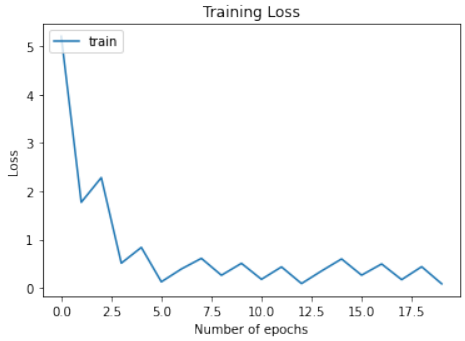

Fig. 9. MobileNetV2 Training Loss

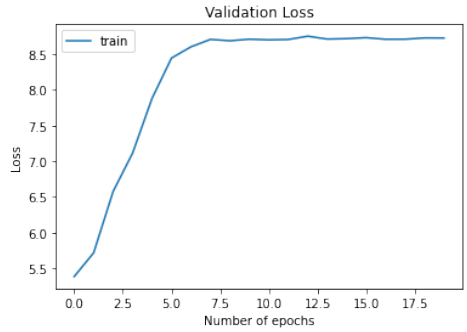

Fig. 10. MobileNetV2 Validation Loss

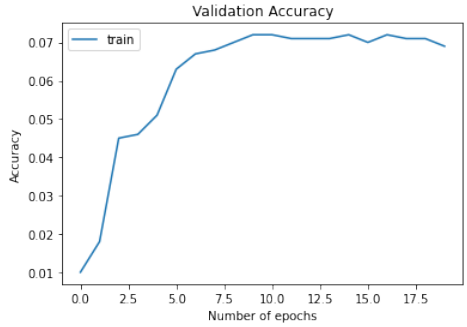

Fig. 11. MobileNetV2 Validation Accuracy

Fig. 12. MobileNetV2 Correct Predictions

For MobileNetV2, The model converged at around 7 epochs, as seen through Figures 9, 10, 11, and held a respectable test accuracy of 74%, shown through Figure 12. MobileNetV2 had less of a smooth training loss and made fewer correct guesses than DenseNet161 by comparison. Despite how lightweight MobileNetV2 is, the amount of guesses that are correct highlights how good its capabilities are. This model still outputs solid results and has the chance to rival some of the other CNNs out there, especially if this application will run on mobile devices.

D. GoogleNet

$LearningRates$	$1e-2$
$1(epoch)$	5.29185
$2(epochs)$	7.0818
$3(epochs)$	3.86282
$4(epochs)$	4.90986
$5(epochs)$	2.36833
$6(epochs)$	3.01384
$7(epochs)$	3.3839
$8(epochs)$	2.01684
$9(epochs)$	2.79813
$10(epochs)$	1.67263
$11(epochs)$	2.46429
$12(epochs)$	1.37211
$13(epochs)$	2.18521
$14(epochs)$	3.00373
$15(epochs)$	1.9103
$20(epochs)$	2.18072

$Time$	$ValAcc$	$TestAcc$
$0HR:26M:29$	50	51

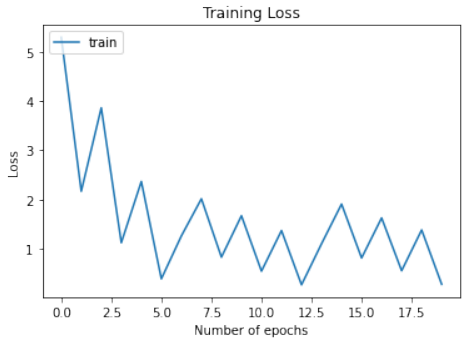

Fig. 13. GoogleNet Training Loss

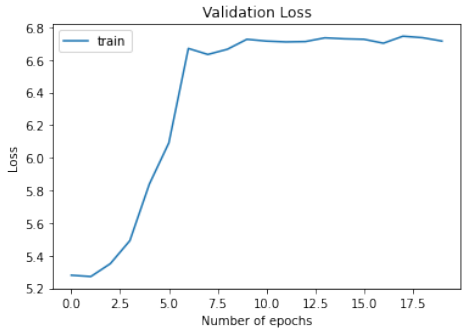

Fig. 14. GoogleNet Validation Loss

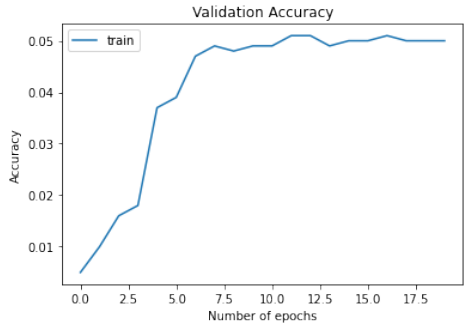

Fig. 15. GoogleNet Validation Accuracy

Fig. 16. GoogleNet Correct Predictions

The model converged at around 7 epochs for validation loss and accuracy, as seen through Figures 14 and 15; however, the training loss continued to oscillate between one and two, shown through Figure 9. The oscillation can be attributed to the inherent dropout feature in GoogleNet. When the data loaders added a dropout component during the testing phase, the accuracy of each model significantly decreased. It can be assumed the use of dropout is detrimental to models with a higher number of classifications due to the loss of key input features and this reduction of accuracy can be seen through Figure 16, having a far lower test accuracy of 51%.

E. Comparison

1) ResNet IV-A
   a) Suzuki SX4 Sedan 2012
   b) Acura RL Sedan 2012
   c) Acura TSX Sedan 2012
   d) Suzuki Aerio Sedan 2007
   e) Chevrolet Impala Sedan 2007
2) DenseNet IV-B
   a) Acura RL Sedan 2012
   b) Chrysler 300 SRT-8 2010
   c) Suzuki SX4 Sedan 2012
   d) Honda Accord Sedan 2012
   e) Lincoln Town Car Sedan 2011
3) MobileNet IV-C
   a) Acura RL Sedan 2012
   b) Suzuki SX4 Sedan 2012

c) Suzuki Aerio Sedan 2007
d) Mitsubishi Lancer Sedan 2012
e) Hyundai Veracruz SUV 2012

4) GoogleNet IV-D
a) Chevrolet Malibu Hybrid Sedan 2010
b) Mitsubishi Lancer Sedan 2012
c) BMW 3 Series Wagon 2012
d) Suzuki Kizashi Sedan 2012
e) BMW ActiveHybrid 5 Sedan 2012

Fig. 17. Random Image Pulled

The correct image was an Acura RL Sedan 2012 which DenseNet and MobileNet properly interpreted, while ResNet had the car as its number two most likley. GoogleNet failed to correctly interpret the image, however, it did recognize that the car was a sedan.

The DenseNet161 model was then used in accompany with YoloV3. YoloV3 served as image detection and area localization. If the YoloV3 model label equaled to ”car” then the bounding boxes, as seen through Figure 18, would be used to crop the image and pass it to the DenseNet161 model for image prediction.

Fig. 18. YoloV3 Bounding Boxes

The final results, as seen though Figure 19, shows the cropped image and the top five cars which the image could be, successfully completing the model.

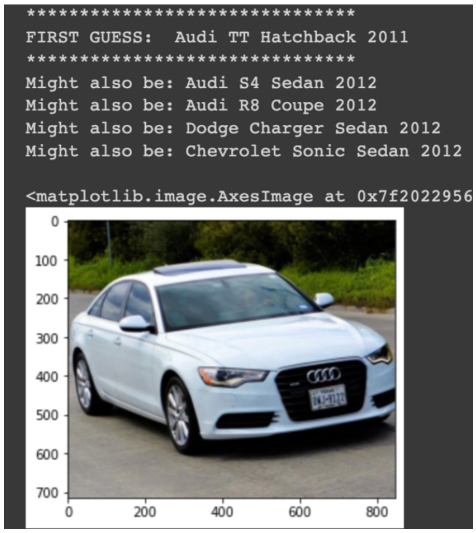

Fig. 19. Cropped Image With Top Five

V. Lessons Learned

During the span of this project, a number of issues revealed themselves and caused a delay in progress time. The first of which was the attempt of creating the initial make year and model classification CNN from scratch. DenseNet161 was the architecture in question and implementing the its Dense Layers, Dense Blocks and Transition Layers all while managing customized hyper-parameters and assigned weights, presented multiple opportunities for errors to occur. When it was made known that developing CNNs this way was not necessary, the choice was made to use the pretrained models of Resnet34, DenseNet161, MobileNetv2, and GoogleNet for the project.

This mentality was also applied to the CNN that had the task of object detection and classification to identify any cars in an image. Initially, the search to design an object detection algorithm to discern the location of objects in an image was paramount as it would be affixed to another CNN that will strictly do generic classification to determine if the object is a car. Since there was no obligation to internally construct these models, an investigation to find a suitable candidate to handle these tasks led to the decision of using YoloV3. Being trained and tested on the ImageNet dataset made YoloV3 a good choice for object detection and initial car classification.

Lastly, a big hurdle that arose was the limitations of using the default version of Google Co-lab. Training some of the models would take up a lot of RAM and spike the system to throttle Google Co-lab while running on the GPU hardware accelerator. In order for training to be completed, the hardware accelerator would then switch over to CPU instead. Google Co-lab was later upgraded to its premium version which allowed for higher RAM usage over all. This also shorted the amount of time needed to process thousands of images for the make year and model classification CNN. Creating a system that could recognize the make year and model of a car in an image became easier after resolving these issues.

VI. GitHub and Dataset References

The GitHub source for this project can be found at:
https://github.com/blaw5/real_time_Final.git.

The project extends James Wang's "Car Make and Model Detection" found at:
https://github.com/jameswang287/Car-Detection/blob/6770ad1cf5a3302d1c3c4863bc67ca352e83a3ed/car_detection_resnet.ipynb.

The cars data set is found at:
https://ai.stanford.edu/~jkrause/cars/car_dataset.html.

The code utilizes Arshren's "Yolo Step by Step" for YoloV3 object detection, found at:
https://github.com/arshren/YOLOV3/blob/e50143d7300f9c8e5bc51a3695d9b3dfad75556b/YOLO%20Step%20by%20Step.ipynb.