Systematic Car Classification System (SCCS)

Hamza Yousaf

CSC420H1
Department of Computer Science, University of Toronto

Abstract- In this paper, an algorithm for the systematic detection of cars is proposed. The algorithm uses several Neural Networks to achieve a step-by-step detection of a car's make, model, and year. We introduce an authentication system that utilizes the Scale Invariant Feature Transform (SIFT) to assist in the detection. This extra layer would allow us to be more positive in the output of the system.

This project had originally started with two students but one of the students dropped the course at the deadline, making it too late for the other to find a new partner. The project was undertaken by a single individual, so the original scope of the project was difficult to meet. It should also be noted that it was difficult to allocate enough time for this project due to the workload of this course and other courses. Therefore, the collection of data, implementation, and the end results are satisfactory at best.

I. Introduction

In today's day and age our lives have become heavily dependent on cars. There are billions of cars on the road today, serving the main and primary purpose of transporting people from one location to another. It is then necessary that we know the type of cars that we see and drive every day.

When criminal activity occurs, security cameras often capture a suspect vehicle. Being able to quickly detect and identify vehicle details can narrow down targets and allow law enforcement officials to act sooner. For dealerships and private sellers in the car market, they can come across many cars for which they need to create listings for. It is expected that a car listing contains all the necessary information about the car.

Our aim was to develop a car classification system where a user can input a picture of any car and then get all vehicle specifications about the car. The classifier would also be able detect multiple cars within an image, highlight them using a bounding box, and then display information about each of the cars. Our primary objective was speed. We wanted users of the system to quickly retrieve the information they needed.

II. Study Of Previous Works

We discovered various methods of detecting cars makes. In particular, a research paper that suggested the use of SIFT to detect car logos, called "*Vehicle Logo Recognition Using a SIFT-Based Enhanced Matching Scheme*". The paper described using segmented car images, where the logo is the primary object, to detect which car manufacturer the logo belonged to. The system descried by the paper performed logo detection and recognition in 1400ms (380 ms + 85ms).

We knew that a system of this caliber would need to incorporate some sort of deep learning. Several neural network architectures were considered, before we decided on MobileNetV2. The architecture, developed by Google, was introduced to drastically decrease complexity and model size of the network. The network was made to work efficiently on mobile devices. The network is known for achieving good results on ImageNet classification.

YOLOv3 is one of the most commonly used object detection neural net model. Although the accuracy of the model isn't as good as R-CNNs, they are known for objection detection speed. We found the system would serve us well for getting bounding boxes in our images in a quick manner.

III. Dataset

We used the *The Comprehensive Cars (CompCars) dataset* which contains 163 car makes and 1,716 car models. There are a total of 136,726 images in the dataset. The dataset contains cars models ranging from the years 2009-2015. We found this dataset to be comprehensive enough for our needs. Although, outside of the scope of this project, there would be a need for even more comprehensive dataset.

IV. Architecture

The Systematic Car Classification System (SCC) was developed to systematically determine the car make, model, and year. We took several existing architectures for individual tasks and combined their results to reach our goal.

Our architecture is split into three main modules: car detection & localization, car make recognition, and model + year recognition with authentication. We shall describe the three modules separately, as each of them utilize their own architectures.

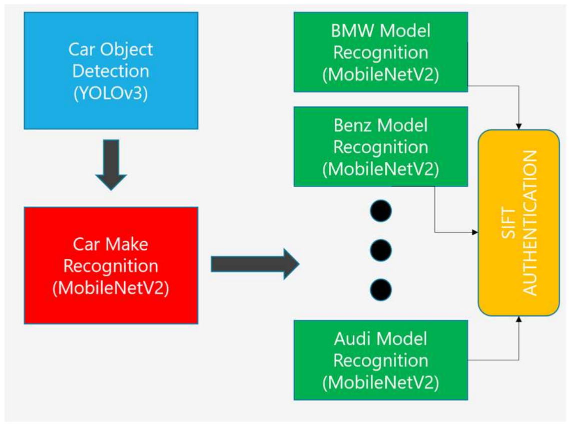

Car Object Detection (YOLOv3)
Car Make Recognition (MobileNetV2)
BMW Model Recognition (MobileNetV2)
Benz Model Recognition (MobileNetV2)
Audi Model Recognition (MobileNetV2)
SIFT AUTHENTICATION

Figure 1 - Systematic Car Classification System Architecture

A. *Car Detection & Localization*

The system can receive any size image, with any number of cars, and any distance of cars from the camera. It is necessary that we detect the cars in the image (if any), and then localize them for the second component. Localizing would involve creating cropped images of the cars as the main object of focus.

We used the classic YOLOv3 architecture (see Figure 2) for detecting the cars in the image. Due to the complexity of the model and the scope of the project, it was not ideal to train the model. Instead, we used an existing pre-trained model from creators themselves. The model was made to detect bounding boxes only around cars. We used the bounding boxes to get only the car images and packaged as input to the second component.

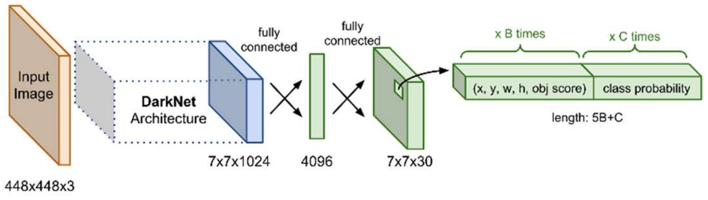


Figure 2 - YOLOv3 Architecture

B. *Car Make Recognition*

Most neural net models are trained to detect the car, make, and year all at once. This results in thousands of classes to detect. For our dataset, there would be a need to detect ~2446 classes. We thought this would be highly inefficient to train a model on and could result in less accuracy (we could not test this due to the scope of the project). Instead, we decided to first accurately detect the make of the car and then figure out the model + year.

We used the MobileNetV2 architecture (see Figure 3 and Figure 4) with pre-trained weights to detect all possible makes of cars (due to the scope of the project, we detect only **15** common car makes). The architecture used was the same as described in the original paper, except we added several dense layers and a softmax layer that would output to 15 different classes.

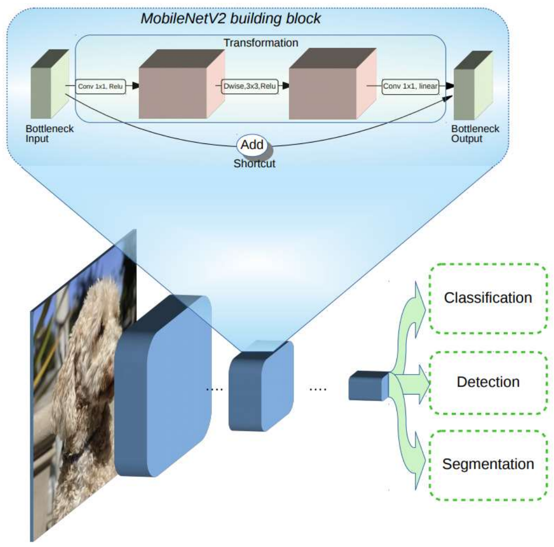
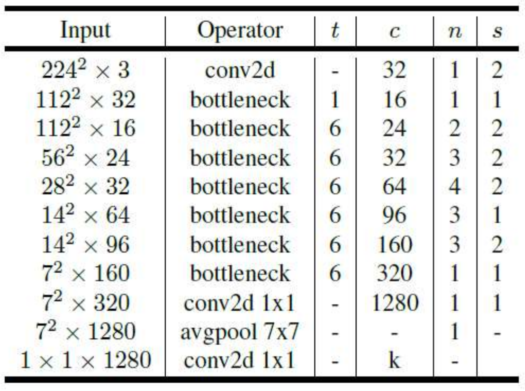


Figure 3 - MobileNetV2 Architecture as described by Google



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 4 - MobileNetV2 Overall Architecture

C. *Car Model + Year Recognition with Authentication*

Once the make of the car is known, the final step is to feed the car through a neural network model tailored for the specific make. We created 15 models, each of which are based on the MobileNetV2 architecture. Each model was trained using the images of cars of the corresponding make. Each model, then with some confidence, predicts which model + year the car is.

It is not enough to depend solely on the neural net to accurately predict the car model. We decided to add an extra layer of authentication and use SIFT to also figure out the make of the car. Our model outputs softmax scores for the prediction of the car model + year. Starting with the top-most score, we retrieve an image from our dataset that reflects the class of the score. The image retrieved will ideally have the same pose as the detected car. We first detect and compute the matching keypoints and descriptors. We then use Lowe's ratio test to keep the best matches. Finally, we compute a homography, estimated with RANSAC.

The accuracy of our homography is entirely dependent on the number of matching points that we find using Lowe's ratio test. If we did not find enough good matches, the homography would not look satisfactory. Based on some **confidence threshold**, we determine whether how accurate was the model's prediction. If SIFT predictor isn't confident enough, we repeat the process, but for the second highest score given by the model. We check this for the top 5 matches, under the guise, that the match's scores are above some **model confidence threshold**.

V. Implementation

We started by first implementing a simple way to access the YOLOv3 model. We download the pre-trained weights and config file for the model, and then read it in using OpenCV's *readNet* function. The input image is resized to 416x416 and then converted to a blob (required as input into the model). The model is run on the image and it outputs a list of detections on the image. If the detection is above some confidence threshold, then we go ahead and extract the bounding box. Finally, we take all the bounding boxes and use them to create separate crops of the image containing only the pixel data within the box. We output this list of images. **See *yolo.py* for implementation details.**

We split the implementation of the MobileNetV2 into three parts: data loading, model definition, and training definition. Our data is given to us in .csv files. We use Pandas to read the .csv files, and then input it into the *get_generators* function. The function uses Kera's *ImageDataGenerator* to read the data from the dataframes. Since we have a lot of data, the generators would load *(# number of images / batchsize)* at a time and feed them into the model for training. This allows us to not require a lot of RAM at once when working with large datasets. **See dataloader.py for implementation details.**

The model is defined using the base model augmented with additional layers. The pretrained base MobileNetV2 model is retrieved from Keras and stripped of the top layers. We add a GlobalAveragePooling2D layer to average out the activations of previous layers (since the model was pretrained). We then add 3 Dense layers (with 1024, 1024, 512 channels respectively) so our model can be trained on the new data. And finally, we add a softmax layer at the end which outputs scores for the total number of classes the model expects (passed in as an argument). **See model.py for implementation details.**

Training starts by first getting the model and determining which layers will be finetuned (passed in as an argument). We then compile the model using the *nadam* optimizer so that we can actively modify the learning rate during training. The loss function is categorical cross-entropy since we have multiple categories. After adding some callbacks for monitoring, we fit our model to the generators from the data loading step. The model is given a training generator, validation generator, number of epochs to run for, steps per epoch (number of samples / batch size), and the callbacks. **See train.py for implementation details.**

We use the same data loading, model definition, and training definition for all of our MobileNetV3 models. We utilized a Google Colab Notebook to train our car make model, and all of our car model models. We did this by actively changing the data that was fed in and the training parameters. Our trainer outputted a .h5 file for each model that we trained, so the model could be reloaded later to use for predictions. **See car-classifer.ipynb for implementation details.**

Once a prediction for the car model has been made, it is run through the SIFT-authenticator. The function finds images of the same model in the dataset and calculates features matches between those images and the image that was originally input into the system. The function calculates the accuracies using the formula: *(# number of good keypoints / total number of keypoints).* The final confidence accuracy is calculated by summing up all those accuracies and diving by the total number of accuracies. **See sift_auth.py for implementation details.**

VI. Results

Note: Due to issues during debugging, we were unable to accurately test any of our models. All our tests were done manually by feeding the models several images and then taking the final output. Due to time constraints, we were unable to retrieve full testing results.

We fed several images into YOLOv3 detector and got the following results:

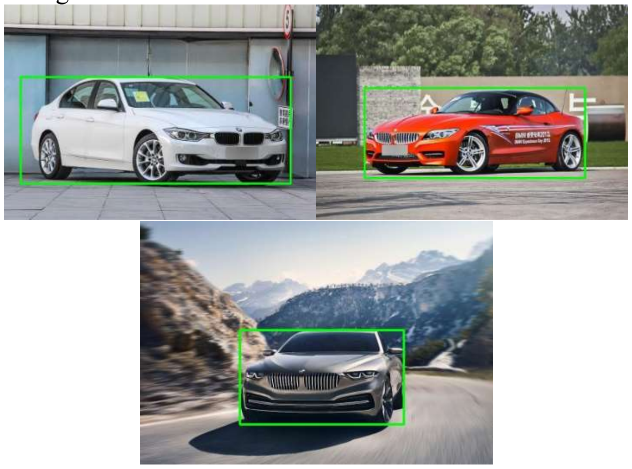
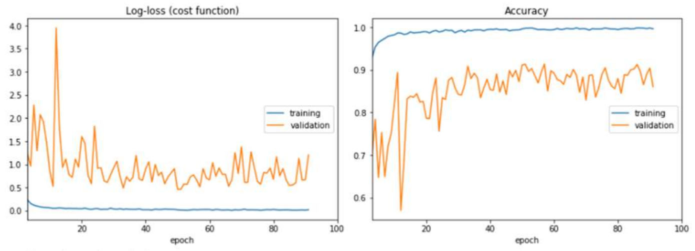


Figure 5 - Bounding boxes detected by the pre-trained YOLOv3 model. The boxes presented are for visual purposes only, the actual function returns the images cropped to these boxes

During training of our car make MobileNetV2 model, we achieved a training accuracy of 99.6% and a validation accuracy of 91.3% after running for 100 epochs, with a "nadam" optimizer, and a categorical cross entropy function.



```
Log-loss (cost function):
training   (min:    0.005, max:    1.053, cur:    0.020)
validation (min:    0.459, max:    6.382, cur:    1.203)

Accuracy:
training   (min:    0.695, max:    0.998, cur:    0.996)
validation (min:    0.375, max:    0.913, cur:    0.860)
Epoch 00091: early stopping
```

Figure 6 – Car Make model training results.

The output of the YOLOV3 detector was then inputted into our newly trained Car Make model, which predicted the results in

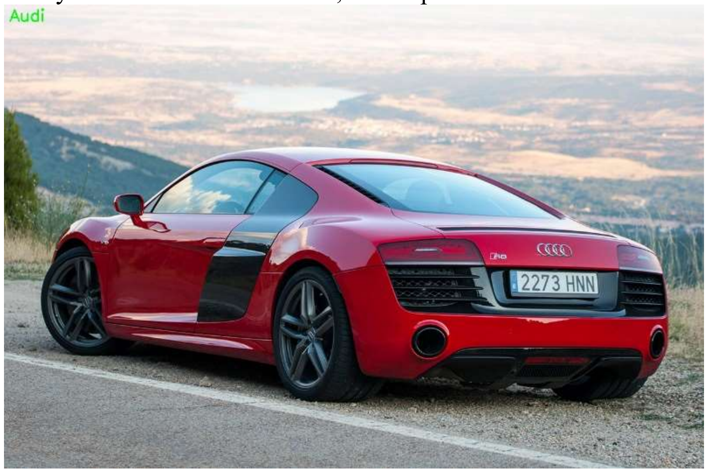
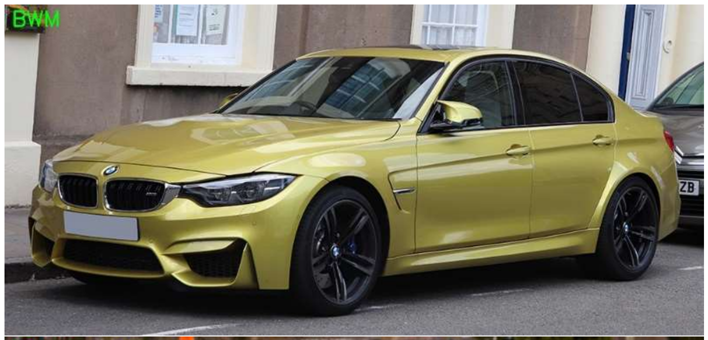


Figure 7.



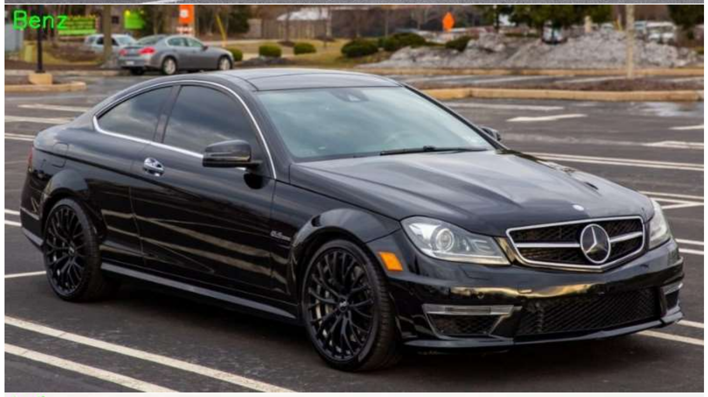
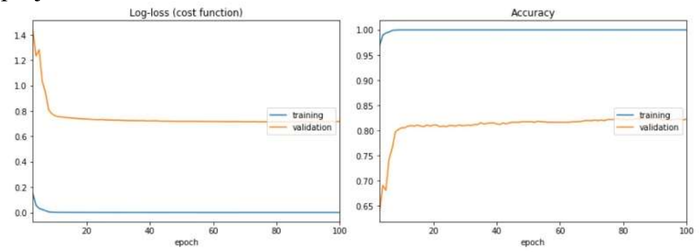


Figure 7 - Predictions made by the Car Make Model. The car make prediction is reported on the top left. This output is only presented for visual purposes, the actual output is just the car make string.

During training of our BMW MobileNetV2 model, we achieved a training accuracy of 100% and a validation accuracy of 82.4% after running for 100 epochs, with a "Adam" optimizer, a learning rate of 1e-3, and a categorical cross entropy function. It is quite obvious that our model overfit the training data, and as a result did not perform as well on the validation data. Additional experimentation with the model layers and training parameters would have let to better results, but that is out of the scope of this project.



```
Log-loss (cost function):
training   (min:    0.000, max:    3.162, cur:    0.000)
validation (min:    0.714, max:    4.226, cur:    0.717)

Accuracy:
training   (min:    0.339, max:    1.000, cur:    1.000)
validation (min:    0.248, max:    0.824, cur:    0.823)
```

Figure 8 - BMW Model training results

The output of the Car Make model was then inputted into our newly trained BMW Model, which predicted the results in Figure 9. All the predictions made were incorrect to a certain degree. We did expect incorrect predictions due to a low validation accuracy. But, if one observes the differences between the actual car models and the predicted car models, it isn't surprising that our model was wrong. Some of the car model predictions were correct but with the wrong year, and other predictions had different models but the differences in the models were too minor. If we had trained our BMW model accurately, then we are sure the class predictions would have been a lot more closer to the actual classes.

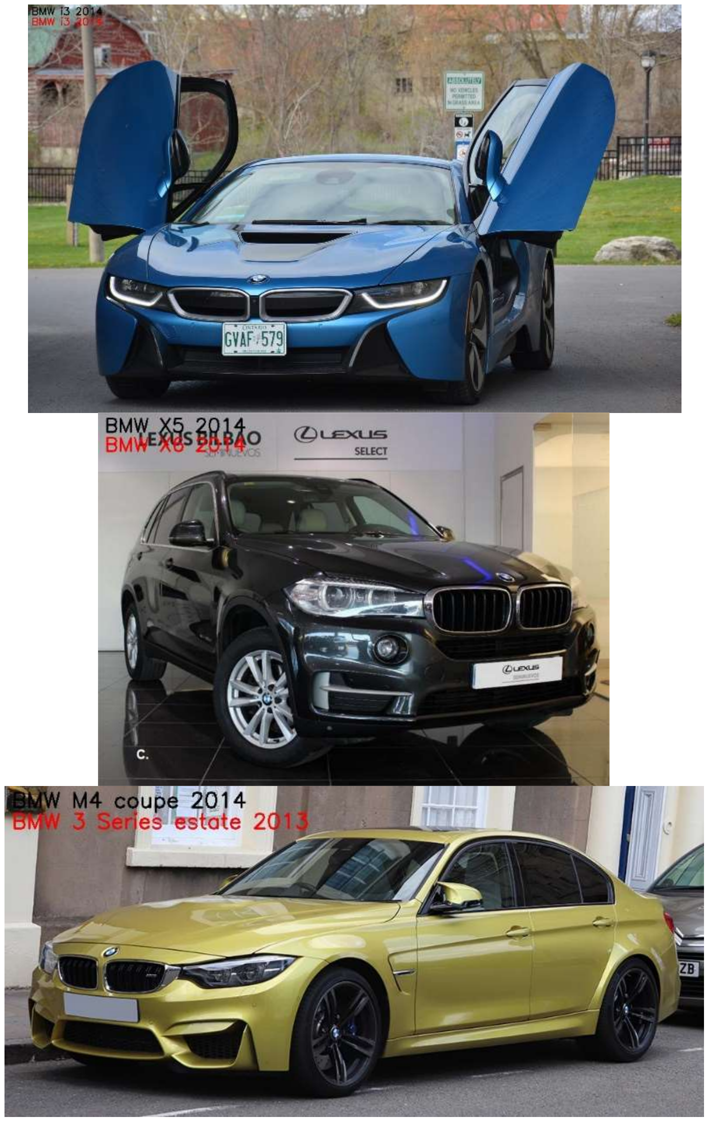


Figure 9 - Predictions made by the MobileNetV2 BMW Model. The black text label is the actual model, and the red text label is the predicted model.

The results for all three of the modules combined is presented in Figure 10.

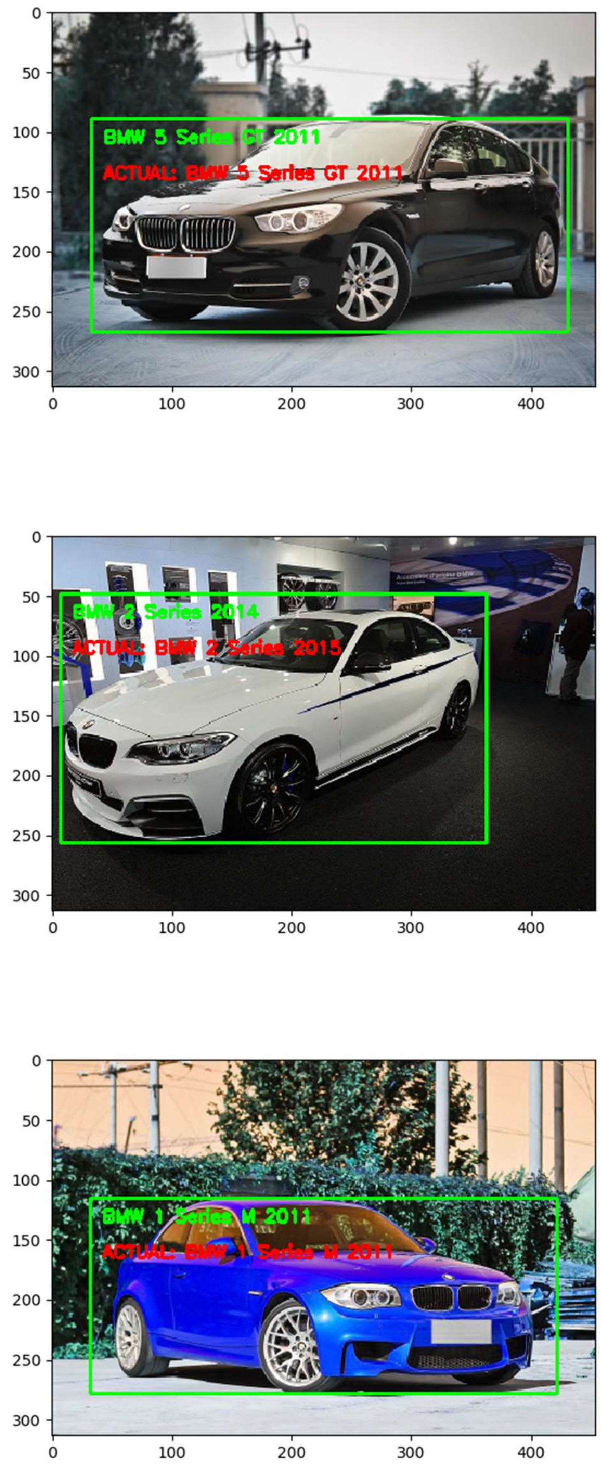

BMW 5 Series GT 2011
ACTUAL: BMW 5 Series GT 2011
BMW 2 Series 2014
ACTUAL: BMW 2 Series 2015
BMW 1 Series M 2011
ACTUAL: BMW 1 Series M 2011

Figure 10 - Final predictions made by all entire system

VII. Challenges

The project’s goal was to efficiently and quickly identify the exact make, model, and year of a car. It was quite ambitious to think we would be able to do this for any vehicle (car, truck, van, etc.). The main barrier towards achieving our goal was time.

The dataset we used had to be limited to only a 15 car makes. It is quite time consuming to train a model on hundreds of thousands of images. We lacked the proper hardware to do it in an efficient manner. So, we had to work with 1/4th the size of the dataset to train our models.

Time also constrained the number of Car Model models we could train. We were only able to train 3 models (BMW, Benz, Audi) for recognition. These models did not have enough time to train as they achieved only 80% validation accuracies and ended up overfitting the training data. Given more time to play around with parameters, an ideal validation accuracy for the models would have been 95%+ (with no overfitting/underfitting).

We were unable to fully develop the SIFT Authenticator algorithm. With the current implementation, given an image of a car and its class, the algorithm would retrieve a list of images from that class and compute the accuracy of the prediction. It would have been better instead to choose an image which had the same pose as the input image, therefore making the accuracy prediction of the algorithm more authentic. Currently, the authenticator throws out low confidence accuracies due to a faulty implementation that we were unable to fix.

VIII. Future Works

We plan on first overcoming the challenges discussed in the previous section by allocating more time to the training process. We will like to experiment with different models for detecting makes and models. The SIFT authentication can be improved by automatically finding a similar pose to the image being authenticated. We want to create an easy to use API to allow developers to access and deploy our system onto their own applications.

IX. Conclusion

This project allowed us to explore the various areas of computer vision and experiment with the different techniques. Our final design was to incorporate various machine learning models along with SIFT to create a promising system for classifying cars specifications. In the future, we intend on breaking the barriers encountered during the project and implementing the future improvements we discussed to create a better system.

References

[1] A. P. Psyllos, C.-N. E. Anagnostopoulos, and E. Kayafas, “Vehicle Logo Recognition Using a SIFT-Based Enhanced Matching Scheme,” IEEE Transactions on Intelligent Transportation Systems, vol. 11, no. 2, pp. 322–328, 2010.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[4] Linjie Yang, Ping Luo, Chen Change Loy, Xiaoou Tang. A Large-Scale Car Dataset for Fine-Grained Categorization and Verification, In Computer Vision and Pattern Recognition (CVPR), 2015.

Authors

First Author – Hamza Yousaf, B.Sc. Computer Science, University of Toronto