CS230

Vehicle Make, Model and Color Recognition based on Deep Learning

Jiupeng Sun
Stanford Center for Professional Development
Stanford University
samparly@stanford.edu

Abstract

Vehicle detection is essential for many areas such as traffic monitor and suspicious car tracking. This paper describes the implementation of Vehicle's make, model and color recognition system (MMCR) based on deep convolutional neural network. In this project, I started from simple networks as the benchmark, and tested multiple state-of-the-art deep CNNs, compared the results and selected the most optimal network, then tuned the networks to pursue a better performance. At the end, I gave analysis to the results of the multiple experiments.

1 Introduction

Vehicle recognition has been widely leveraged nowadays. For example, the camera on the highway can adopt such technique to monitor the traffic load and track suspicious vehicles. In addition, this technique can benefit common consumers as well. It would be convenient for users to retrieve more information of a car in the street by just taking a picture, even they know nothing about the car. This project aims to provide an efficient and accurate system to prove the practicability of the idea.As an image recognition task, the state-of-art deep learning will be used. The input would be a RGB picture which displays a car in arbitrary angle, and output would be an explicit string that describes the make, model and color of that car.

2 Related work

Much work have been done in recent years related to this topic. As the rising popular of machine learning, almost all of the other works took the advantage of this blooming technology. Some work are using SIFT and SURF[13] algorithm to retrieve features and use SVM to classify. But most of others choose to train a deep neural network for end-to-end learning, such as [12] and [6]. There are also some investigations on many popular CNNs such as SqueezeNet, CoffeeNet and GoogleNet, in which people train different networks and compare their performance. [7], [3], [6]. As demonstrated in [7], transfer learning is essential and dramatically useful when we need to train a network on different dataset. Furtherly, some people try to modify the original structure of these ready-made networks to achieve higher accuracy on their dataset such as [1] and [8], and integrate data augmentation or adding Gaussian blur / noise to migrate over-fitting [11][4]. Most of above studies focus on make and model recognition only, while in my work I will take the color classification into consideration because color is also a non-negligible character of a car.

CS230: Deep Learning, Summer 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset and Features

A well-collected and suitable dataset is vital to the success of this project. As we know, numerous car manufacturers will release new models or even makes almost every year, and the difference between two consecutive years is usually insignificant, let alone the various colors of the same type, thus the data collection work can be extremely time consuming. To save this effort, fortunately, some work has been done to provide accessible dataset online such as Yang's dataset[12] and Stanford vehicle dataset[5]. I preferred to use Stanford data than Yang's because it can be directly downloaded.
This dataset contains 16,185 images from 196 classes. And the data is split into 8,144 training images and 8,041 test images. Each car is described with Make, Model and Year, which means the 196 classes may contain cars of same make or model but different years.

Figure 1: Sample images from Stanford Vehicle dataset

The vehicles in the dataset are all RGB images but the data quality differs significantly. For example the largest resolution is (5400, 7800), which was well taken in a suitable brightness and perfect angle. While the lowest resolution is (58, 78) which was most probably a screenshot from internet. I further split the test set with validation set and prediction set with 5:5 fold. The validation set is used to validate the training result in each iteration and tune the models at the end, but will never be learned by the model. Finally, I will use the model to predict test dataset and justify the performance of my network. Besides, to accelerate the development cycle, I also extract around 10% images from train set as developing set to debug my program.

4 Methods

As a classification problem, cross entropy is enough to be the loss function. Firstly, the softmax function produces:

$$softmax(s_i) = \frac{e_i^s}{\sum_{j=1}^{N} e_j^s}, i = 1, ..., N \tag{1}$$

where s_i is the similarity of the class i, N is the number of the classes.
The cross entropy measures the quality of a model for a probability distribution.

$$CE = -\frac{1}{M} \sum_{k=1}^{M} \sum_{i=1} N y_{ik} \log(softmax(s_{ik})) \tag{2}$$

where k is the index of the element in the batch, M is the batch size.
I will train a DCNN to classify the color (CN), and use a separate DCNN to recognize the make & model (MMN). Although using two networks mean more memory usage, but it can dramatically reduce the number of predicted classes and fasten the learning pace. On the other hand, color classification is relatively simpler than make & model recognition, not only because of the less output classes, but also more straightforward to classify the color of an image than recognizing the object inside the image. I directly leverage the work from [9], which has already gave an efficient and highly accurate CNN (97% accuracy) to classify the color of vehicle.
Another powerful technique to be used is YOLO algorithm, which outputs a cropped image, then I use this new image as the input of CN and MMN. Theoretically the cropped image would contain less noise and easier to be learned. According to the later experiments this change did significantly

improve the accuracy of prediction. There are already numerous studies and work about YOLO, I downloaded an open source implementation of YOLO from [10]. According to the paper, YOLOv3 can provide around 60% mean average precision. To understand the accuracy of YOLO, I calculated the intersection of union (*IOU*) between YOLO outputs and the ground of truth, which is the bounding boxes provided within the dataset that marks the actual location of the vehicle in each image. The average *IOU* is near 95%. Thus the overall architecture of the application is shown as Figure 2, notice I also resize the output of YOLO to (224, 224, 3) before feeding it to the neural network. For the core module "Make and model recognition CNN", I chose several mature networks from the Application model of Keras[2], which can help mitigate the time constraint on my work. The networks I used includes VGG, SqeezeNet, ResNet50, InceptionResNetV2 and MobileNet. To leverage transfer learning, I used ImageNet weight and **freeze** the whole network as base model, and add two blocks at the end of base model, each block contains one dense layer, one batch normalization layer and one dropout layer, so I only need to train the parameters in last two blocks for each network 3. In addition I also implemented one logistic regression model and two simple CNNs as the baseline models 1.

Model	**Size**	**Top 1 accuracy** [1]	**Top 5 accuracy**	**Parameters**	**Depth**
Logistic Regression	-	-	-	-	-
One-layer CNN	-	-	-	205,576,388	1
Two-layer CNN	-	-	-	48,016,188	2
VGG16	528MB	0.713	0.901	138,357,544	23
SqueezeNet	0.5MB	0.563	0.801	730,688	-
ResNet50	98MB	0.749	0.921	25,636,712	-
InceptionResNetV2	215MB	0.803	0.953	55,873,736	572
MobileNet	16MB	0.704	0.895	4,253,864	88

Table 1: Comparison between experiments models

[1] The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

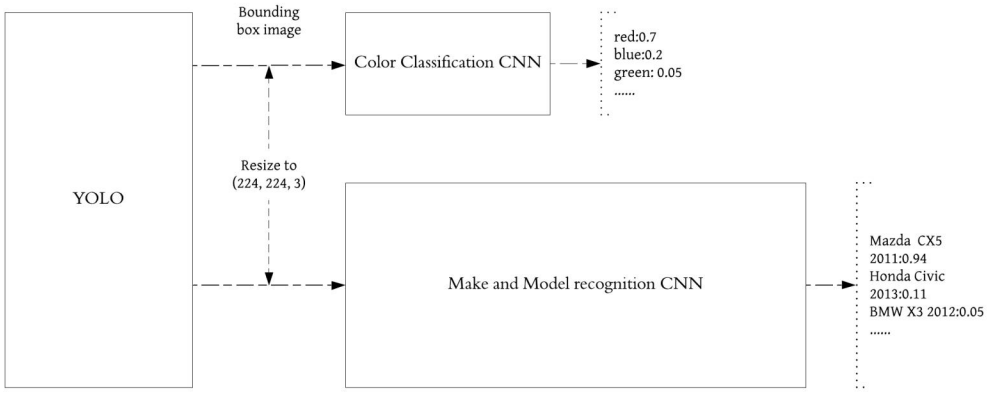

Figure 2: Overall Architecture of the System

5 Experiments and Results

During the training stage, I experimented several approaches:

Directly train the MMN

I performed several experiments on baseline networks and other MMNs with $iterations = 50, learningrate = 0.001, learningratedecay = 1e-4$ and used ***RMSProp*** optimization. The choice of these hyper-parameters were based on multiple experiments, which means it can represent the best performance that most of the networks can achieve. For example, I found after 50 iterations,

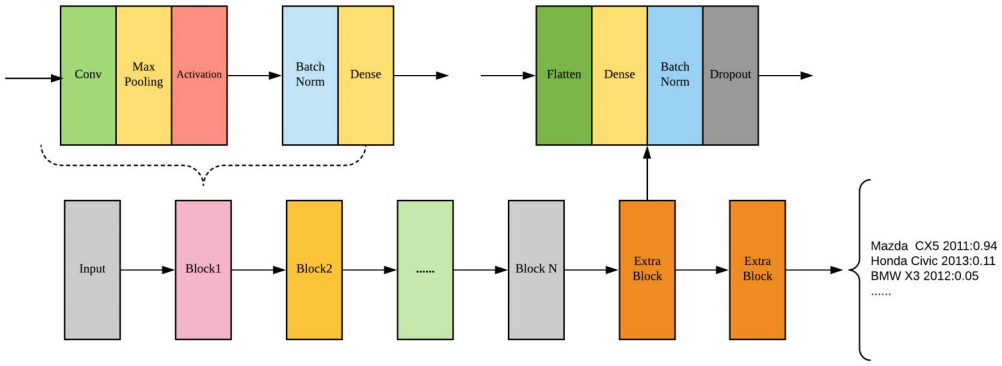

Figure 3: Convolutional Neural Network Architecture

all of the experiments networks can almost reach the asymptotic line of accuracy, longer training didn't help much. I also tried three types of optimizer: *RMSProp*, *SGD* and *Adam*. With same learning rate and decay rate, *RMSProp* can reach the saturated accuracy most quickly 2.

Model [1]	**Top 1** [2]	**Top 5**	**Loss**	**Top 1**	**Top 5**	**Loss**
Logistic Regression	0.0044	0.4752	16.3919	0.0056	0.9926	16.5312
One-layer CNN	0.0059	0.9854	16.0229	0.0049	0.9816	16.0398
Two-layer CNN	0.9969	0.9999	0.0296	0.0685	0.0227	15.1865
VGG16	0.9781	0.9980	0.0911	**0.3769**	**0.6185**	3.2768
SqueezeNet	0.8722	0.9846	0.4632	0.2648	0.5259	3.5150
ResNet50	0.9962	0.9998	0.0260	0.0054	0.0266	7.5539
InceptionResNetV2	0.9841	0.9991	0.0924	0.1170	0.3115	4.9813
MobileNet	0.9972	0.9993	0.0265	0.3327	0.6082	2.9891

Table 2: Experiments 1 results

[1] $iterations = 50, learningrate = 0.001, learningratedecay = 1e-4$

[2] Column II to column IV represents training metrics, and column V to column VII represents validation metrics.

Train the MMN with data augmentation

In the next experiment, I use data augmentation technique to remedy the insufficiency of training dataset. I set the $rescalefactor = 1./255, shearrange = 0.2, zoomrange = 0.2, horiontalflip = True$. The choice of shear range and zoom range is arbitrary, I tested with 0.2, 0.4 and 0.6 respectively, the results didn't differ much, but in reality, the vehicle image are not usually highly sheared or distorted, so I prefer smaller value. Also, I eliminate the possibility that user inputs a upside-down image so I didn't set $verticalflip = True$ 3. Figure 4 gave a couple of examples that visually demonstrated how does the data augmentation manipulate the original picture.

(a) Original (b) horizontal flip (c) zoom (d) compress

Figure 4: Data augmentation effects

Add YOLO with pretrained network

Model [1]	Top 1	Top 5	Loss	Top 1	Top 5	Loss
Logistic Regression	0.0053	0.3799	16.3648	0.0054	0.0222	16.2742
One-layer CNN	0.0055	0.9989	16.0289	0.0049	0.9995	16.0398
Two-layer CNN	0.9670	0.9985	0.1142	0.0833	0.2415	15.1732
VGG16	0.8334	0.9674	0.6233	**0.4930**	**0.7503**	2.2686
SqueezeNet	0.2250	0.4852	3.5426	0.2714	0.5323	3.4863
ResNet50	0.9508	0.9956	0.2284	0.0046	0.0258	6.6878
InceptionResNetV2	0.6588	0.8948	1.3518	0.1590	0.4058	4.2085
MobileNet	0.9487	0.9936	0.2271	0.3969	0.6969	2.5281

Table 3: Experiments 2 results

[1] $rescale_factor = 1./255, shear_range = 0.2, zoom_range = 0.2, horiontal_flip = True$

After pre-trained with data augmentation, I add YOLO layer in front the the MMN. The other hyper-parameters remained the same with previous step. 4

Model	Top 1	Top 5	Loss	Top 1	Top 5	Loss
VGG16	0.8889	0.9838	0.3946	**0.7531**	**0.9300**	1.0433
SqueezeNet	0.4015	0.6853	2.6378	0.4914	0.7726	2.2564
ResNet50	0.9605	0.9955	0.1813	0.0066	0.0263	7.4918
InceptionResNetV2	0.7128	0.9155	1.1399	0.2315	0.5134	3.6729
MobileNet	0.9630	0.9982	0.1804	0.5924	0.8513	1.6588

Table 4: Experiments 3 results

Well-tuned a network

At this stage, we already collected three groups of metrics for each network, and it can be obviously observed that VGG network demonstrated the best performance. In this experiment, I chose to further tune VGG network only, continue training the weights from previous experiment with $iterations = 100, optimizer = SGD, learningrate = 0.0001, learningdecay = 1e-4$. With longer training and slower learning rate, we can observe the VGG network achieved 0.7731 top 1 accuracy in validation set 5.

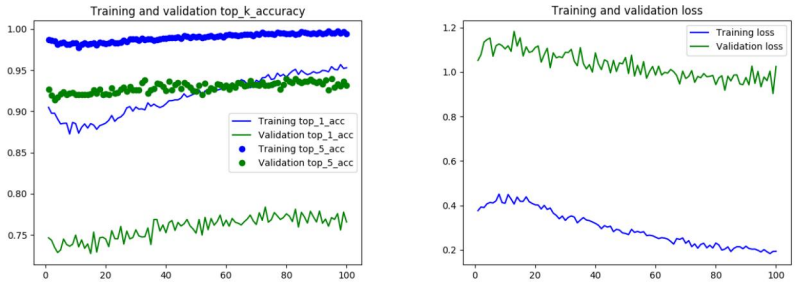

Figure 5: Accuracy and loss curve of VGG network

Prediction results

After all, the tuned VGG won over all the other networks, thus I chose this network as the final model and predicted the test set, which contains 4064 images that was never been used during training and validation stage. I run my experiment on AWS p2.xlarge instance (11.75 ECUs, 4 vCPUs, 2.7 GHz, E5-2686v4, 61 GiB memory, EBS only), the prediction process spent around 40s to finish. Table 5 displays the prediction results.

Test loss	1.0075
Top 1 accuracy	0.7953
Top 5 accuracy	0.9391

Table 5: Prediction results

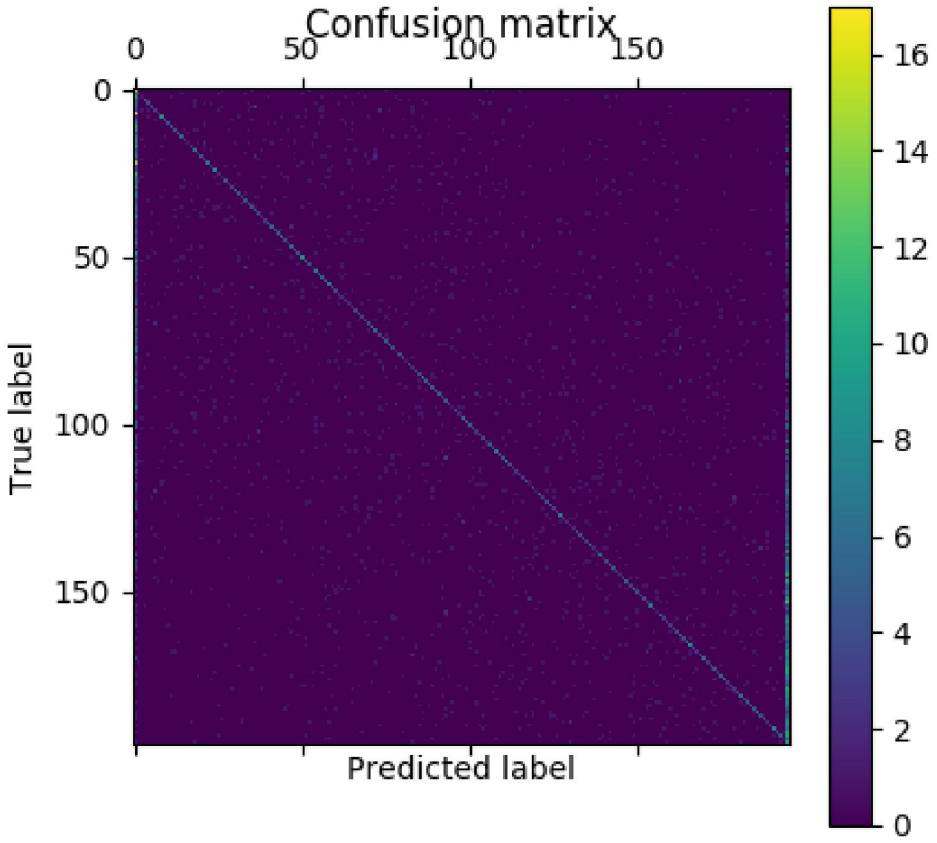

Figure 6: Confusion matrix of final results

6 Conclusion

Baseline

As shown in the tables 2, none of the baseline networks are satisfied. Logistic regression and one-layer network are too naive to learn the huge number of classes. For the self-defined two-layer-CNN, although it can achieve high enough accuracy for training set, but it was also overfitting which struggled in validation. Thus we need more sophisticated and well-design architecture.

Transfer learning

Transfer learning is extremely useful. I also tried to train a VGG network from scratch

without using ImageNet for 300 iterations. However the learning progress was very slow and the loss function didn't drop significantly at the end. On the contrary, with the help of transfer learning, I can almost reach the asymptotic of accuracy with 50 iterations only. [7] also proved transfer learning can achieve outstanding performance.

Data augmentation

Comparing with the inestimably large number of vehicles in the real world, my dataset is still insufficient. Thus data augmentation played an important role in this project. By using this method, I made the most of our few training examples and "augment" them via a number of random transformations. This obviously helped prevent overfitting and helped the model learn better.

Prediction result

YOLO almost improved the top-1-accuracy metric to 50%. I also tried to pre-process the images with the bounding coordinate provided in the Stanford dataset, and trained the network with these dataset. However the result didn't beat the accuracy of YOLO. I believe this is expectable since although YOLO didn't output the exact same bounding box with the ground truth data, it can still cover the most part of the vehicle in the image. And even YOLO may leave more context surrounding the cars, these extra pixels didn't harm the network a lot.
As displayed in the confusion matrix, the errors produced by VGG seems pretty random, but with more careful observation we can find the errors appear to be closer to the diagonal than otherwise. And because the dataset groups identical make & model as sequential classes, some vehicles are pretty easy to be classified as its sibling classes. Another root cause of the errors was introduced by the poor quality of some images, the low resolution and intolerable brightness makes the network confusing. I picked several mistakenly classified images from the test set demonstrated in 7. Figure.(a) is wrongly classified because there isn't no picture in training set taken in the same angle. Figure.(b) is labeled as *Audi TTS Coupe 2012*, one reason is that these two cars have similar appearance, another is because of the low quality of this image. Figure.(c) is marked as the same type with figure.(d), you can see these two vehicles are barely distinguished by human beings' eyes.

(a) Acura RL Sedan 2012 (b) Audi R8 Coupe 2012

(c) Audi 100 Sedan 1994 (d) Audi 100 Wagon 1994

Figure 7: Mislabeled images

7 Future Work

Data collection

Because of the time limitation, I don't have time to test other open dataset. No doubt that abundant training data can help mitigate overfitting. And we definitely need to add new make & models to train the network. In addition, pre-processing the data is also worth a shot such as removing the low resolution data and the low quality images which are difficult to identify the car even for an expert.

Feature extraction

Due to the similarity of cars in same make & model but different years, I believe there are already studies done by others to tackle with this problem. For example, use special feature extracting method to enhance partial parts of the vehicles, because most of the minor changes happens at the edges like the bumper and hub, so augmenting some features can help neural network to know which parts have higher weights and drive them to a right direction.

Model selection

I would like to try more models such as GoogleNet. Currently I freeze the weights of the model and only append some normalization and dense layers at the end of the models. But I can try to insert several blocks in the middle and re-train the network for more iterations and different combination of hyper-parameters. Meanwhile although cross entropy is the most common loss function for multiple-class classification problem, I may also try other loss functions such as mean squared error and sparse categorical cross entropy.

Online real-time MMCR system

For demo purpose or even practical application, it is more convenient to create a web application which allows user to upload multiple car images and try the model. Currently the response time of single image processing can be done within second, but it requires around 500MB memory to load the mode, thus it is still a little overhead if we want to fit this model into a smart phone's memory, we may try to compress the weights or use other models which are tuned for mobile devices.

8 Acknowledgement

I do appreciate **CS230 course staffs** for their dedicated work to make this class successful, especially give thanks to my mentor **Tugce Tasci**, who contributed a lot of advice to help me finish this project.

References

[1] Qi Bu, Shanzhen Lan, and Pin Xu. A cnn based car model recognition improvement. In *Proceedings of the 2017 International Conference on Deep Learning Technologies*, pages 86–89. ACM, 2017.

[2] François Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[3] Afshin Dehghan, Syed Zain Masood, Guang Shu, Enrique Ortiz, et al. View independent vehicle make, model and color recognition using convolutional neural network. *arXiv preprint arXiv:1702.01721*, 2017.

[4] Syed Hasib Akhter Faruqui and Rajitha Meka. Vehicle make and model classification using convolutional neural networks.

[5] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[6] Hyo Jong Lee, Ihsan Ullah, Weiguo Wan, Yongbin Gao, and Zhijun Fang. Real-time vehicle make and model recognition with the residual squeezenet architecture. *Sensors*, 19(5):982, 2019.

[7] Derrick Liu and Yushi Wang. Monza: image classification of vehicle make and model using convolutional neural networks and transfer learning, 2017.

[8] Mohamed Nafzi, Michael Brauckmann, and Tobias Glasmachers. Vehicle shape and color classification using convolutional neural network. *arXiv preprint arXiv:1905.08612*, 2019.

[9] Reza Fuad Rachmadi and I Purnama. Vehicle color recognition using convolutional neural network. *arXiv preprint arXiv:1510.07391*, 2015.

[10] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[11] Burak Satar and Ahmet Emir Dirik. Deep learning based vehicle make-model classification. In *International Conference on Artificial Neural Networks*, pages 544–553. Springer, 2018.

[12] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3973–3981, 2015.

[13] Hashir Yaqoob, Shaharyar Bhatti, and Rana Raees Ahmed Khan. Car make and model recognition using image processing and machine learning. 2014.