Simple Driving-Skill Evaluation System

Yuan-Pu Hsu
University of California, Riverside
yhsu018@ucr.edu

Tianxiang Sun
University of California, Riverside
tsun024@ucr.edu

***Abstract*—In this project, we implemented a simple driving-skill evaluation system, which can be put behind the windscreen of the car and recognizing patterns showing up on the road. By comparing the pattern we got with the speed of the car, we can evaluate the driving skill of the driver. We built our own iOS application, constructed multiple Convolutional Neural Network (CNN) models to recognize traffic signs and traffic lights and achieving 99.8% test accuracy for recognizing images taken in South California. The application can connect with different platforms, like Google Compute Engine (GCE) and local terminal (PC) via PHP protocol. In the end, we found that there existed a performance trade off between the accuracy and the time latency between different platform.**

***Index Terms*—Driving-skill evalution, Convolutional Neural Network (CNN), Google Compute Engine, PHP**

I. Introduction

Nowadays, over 90% of families in the United States at least have one vehicle. Driving has already been a big portion of American's daily life for decades. Therefore, we want to make a simple system to evaluate the driving skill of drivers after finish a tour, and the only thing needed is a smart phone. We believe it could help drivers to evaluate their driving skill and improve their skill to some extent. Furthermore, insurance company can also take advantage of this system to evaluate the credit of their clients by using their scores, while those who has higher scores, which means better driving skill could be benefit with lower insurance payment.

When it comes to details, firstly we build an application on our mobile device (in our case, iPhone6s) and use it to implement our system on. The user would be asked to put the phone behind the windscreen, and the application would automatically capture images of front view to recognize different kinds of patterns on the road. In our project, the recognized pattern has been restricted to traffic signs and traffic light. Additionally, in order to recognize patterns, we proposed a 6-layers CNN network, which consisted by 6 convolutional layers, 3 max-pooling layers, 3 drop-out layers and a fully connected layer with softmax. The test accuracy of our model for GTBSRB is 96.4%. Beside, by using Lisa dataset, test accuracy of traffic signs is 99.8% and it is 99.5% for traffic light. What is more, we implement our model on three different kinds platforms, iphone, google cloud and local terminal. We calculate the data transfer time, CNN model running time and server running time (if they have)for those three platforms, respectively. The result

Identify applicable funding agency here. If none, delete this.

The contribution of our project is shown as follows:

- We design our own iOS application by Swift deploying on iOS 11.2. In addition, we are able to measure the speed and acceleration, and compare with the recognized traffic sign and traffic light from road image in nearly real-time
- We propose and construct our own CNN architecture which is a 6-layer CNN with 3 max-pooling layers, 3 drop-out layers and 1 fully-connected layer with softmax to conclude the results. The model is trained by using German Traffic Sign Recognition Benchmark (GTSRB) and LISA dataset for different kinds of scenarios.
- We implement three platforms to run our model on: mobile device, GCE and PC. In order to implement our platform, evaluations had been done to decide which GPU we can used in GCE. Besides, we use those three platforms to test the model we build.
- We use PHP protocol to enable our application to communicate with other platforms. It could upload images captured by the application to other platforms. After the server received the test image, they could run the CNN model to determine the pattern. Meanwhile, we evaluated the data-transfer time, running time with different Internet connection and on different platforms.

II. Background and Related Work

A. Background

1) Convolutional Neural Network:

Convolutional Neural Network(CNN) is a important, widely-used model in machine learning. It is a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery. The following is brief introduction about the layer architecture we chose:

- **Convolutional layer** apply a convolution operation to the input and pass it to the output. By doing convolution between original image and filters, more detailed information are gotten and a feature map has been created.
- **Max-pooling layer** is used to reshape the image by using the maximum value from each of the cluster of neurons at the prior layer.
- **Drop-out layer** is a regularization technique to drop out some units in a neural network to reduce over-fit in neural network.
- **Fully-connected layer with softmax** is used to flatten the result we gotten before. What is more, softmax can be using as the classifier to get the classification of the network.

2) Tensorflow and Keras:

- **Tensorflow** is an open source soeftware libaray for high performance numerical computation. Its flexible architecture allows easy deployment of computation acrosss a variety of platforms, and from desktop to clusters of serves to mobile and edge devices. In machine learning and deep learning, it does strong support and has been widely used.
- **Keras** is a high-level neural networks API, written in Python and capable running on top of Tensorflow, CNTK, or Theano. It is much easier to deploy our own neural network by using it. Furthermore, it can decrease the running time significantly, which makes fast experimentation enable.

B. Related Work

Before we began our work,we had been inspired by the work of Kang,Y.P. et al. [1]. In their paper, they did partitions for the neural network and running various parts on different platforms. Besides, Kang and his team evaluated the latency and energy consumption of their new system, which showed that their new system can achieve up to $40.7\times$ latency speed-up, reduce mobile energy consumption by up to 94.7%, and improve data-center throughput by up to $6.7\times$. Additionally, we also got some inspired by the work of Shustanov, A. et al. [2] and Habibi Aghdam, H., et al [3]. while constructing our CNN model. In the work of former one, the team describe a revised end to end technology for detecting and recognizing traffic sign in real-time by CNN. It could use the speed received from the vehicle to predict not only the presence of the object but also the scale and its exact coordinates in the neighboring frame. In the latter work, the team proposed a light CNN which is designed for detecting traffic sign on high-resolution images. They trained the network in two steps. First, the negative sample would be randomly selected from the training image. Second, it has been used to train the CNN using more appropriate data. The result of prediction has increased to 99.89% on GTSRB dataset. Besides, we learned from the work of Simonyan, K., et al. [4] about how to increase the performance while meeting a large-scale image recognition setting. The team showed in the paper that they designed an architecture which increased the depth of the CNN model. Their model showed only 6.8% test error on top-5 testing.

III. Proposed Work

The implementation of our project can be separate to three parts: iOS application (deploy on iPhone6s), Google Compute Engine with TensorFlow for speed-up development and accessing GPU computation, and the core CNN model for recognition. The system flow chart is shown in Figure 1.

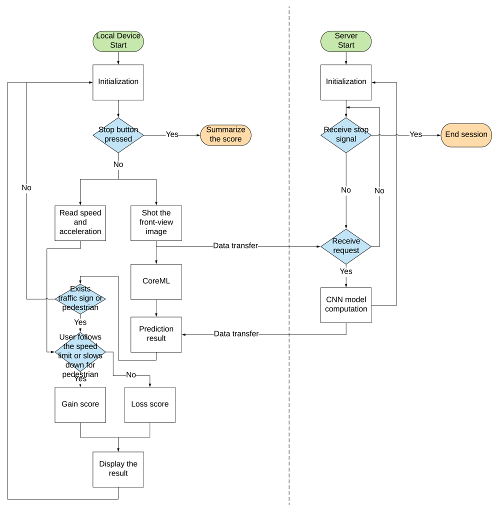


Fig. 1. System Model Flowchart

A. iOS application Design

We built a simple application with Swift and deploy on iOS 11.2. The application could take the front-view picture periodically, and record both the speed and acceleration at the same time via GPS and accelerometer. The picture will then pass through the trained CNN model (coreML) installed in the bundle, or upload to the cloud service for the prediction. After the translation of the speed limit is done, the application can determine if the user was over speed or not. The illustration of the application design is shown in Figure 2.

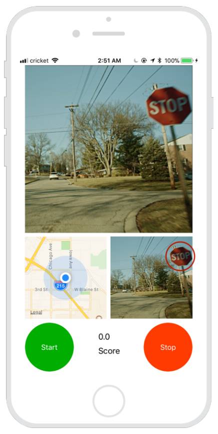


Fig. 2. iOS app design

B. Convolutional Neural Network Model

1) Dataset: There were two popular datasets: GTSRB and LISA dataset. GTSRB contains 43 kinds of different traffic signs in Germany with 39,209 images in the set; LISA traffic sign dataset contains 47 kinds of different US traffic signs with 7,855 annotations on 6,610 frames; LISA traffic light dataset, on the other hand, contains 6 super-classes of traffic lights with total 43,007 frames and 113,888 annotations. Specifically, the images in LISA were taken in South California, which is more close to our proposed scenario.

2) Preprocessing: The training set would first be preprocessed with normalizing, centering and histogram equalizing to help accelerating the train process in the later section. In addition, they would also be augmented, by applying rotation, mirroring and flipping, etc., to mimic more possibilities in the real world cases. And then be separated into training, validating and testing sets. The examples of the common type of sign images in the LISA dataset is shown in Figure 3.

Fig. 3. Example of dataset

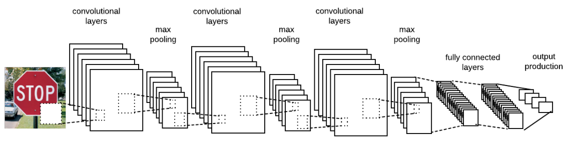
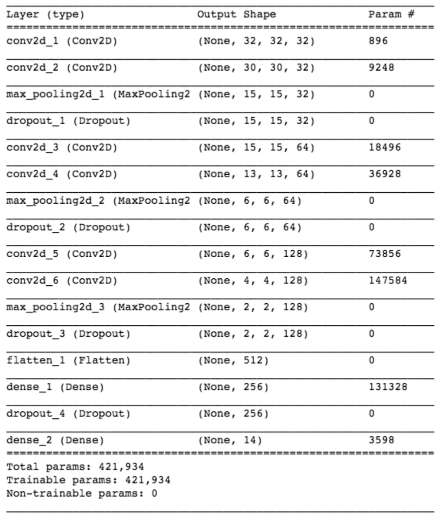


Fig. 4. Architecture of the proposed CNN model

3) Model Architecture: In order to do the recognition in our system, we constructed a 6-layer CNN. The model architecture and details of each layer are shown in Figure 4 and Figure 5 respectively. Each convolutional block contains 2 convolutional layers with the same number of kernel and input/output nodes. Additionally, the each block would also followed by a max-pool layer and a drop-out layer. The features extracted by the previously mentioned layers would then be feed to a fully-connected layer with dropout. Finally, the softmax layer would aggregate the result and output the probabilities of the input image with regard to each class.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 128)         73856
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 128)         147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 14)                3598
=================================================================
Total params: 421,934
Trainable params: 421,934
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5. CNN layer detail with parameters

C. Cloud Instance Implementation

There are a lot of cloud instances architecture provides by Google cloud. In order to get most suitable one for our project. We do some investigation and analysis for the cloud instance implementation.

As mentioned above, we use CNN model to recognize traffic signs and traffic lights. While talking about CNN model, it can be run either on CPU and GPU. However, GPU instance is having better performance on running machine learning model. For GPU, the major function decides it needs to do a lot of vector and matrix computation. Those operations are also widely used in machine learning model. Besides, by using GPU, we can do the computation on a large amount of data in higher parallelism. Those make GPU a better choose than CPU to run the machine learning model.

According to official references, Google cloud provides three different available GPU. They are Nvidia Tesla V100, Nvidia Tesla P100 and Nvidia Tesla K80 respective. However, the Nvidia V100 is for google cloud beta users only. Therefore, we would only do comparison between Nvidia P100 and Nvidia K80.

Firstly, we show the key hardware differences betweeen Nvidia Tesla P100 and K80 in table I.

In the following part, we are going to focus on memory bandwidth and processing power, which are two key characteristics of GPU relating to machine learning. Besides, we would show the performance of various benchmarks running on different kinds of GPU to verify our analysis.

1) Memory Bandwidth: Memory bandwidth would be the most important performance metric while running machine learning model on GPU. It shows the ability of GPU to handle large amount of data. The memory bandwidth of different GPU are shown in table II. From the table, we can get that P100 's memory bandwidth is $3\times$ bigger than K80 per core.

TABLE II
PROCESSING POWER

GPU	Memory bandwidth
Nvidia Tesla K80	240GB/s per core
Nvidia Tesla P100	720GB/s per core

2) Processing Power: Processing power is also a key characteristic we need to focus on. It indicates the highest speed GPU can achieve to crunch data. It can be presented by the number of CUDA cores multiplies the frequency of the core.

$$Processing\,power = Number\,of\,CUDA\,cores \times frequency$$

The result is shown in table III:

TABLE III
PROCESSING POWER

GPU	CUDA cores	frequency	Processing power
Nvidia Tesla K80	2496(per core)	562MHz	1402752
Nvidia Tesla P100	3584(per core)	1126MHz	4035584

3) Benchmark Result: In order to make the result more intuitive. We show some evaluation about the influence of running various benchmarks on different kinds of GPUs and

TABLE I
CLOUD INSTANCES GPU PARAMETERS

GPU	Cores	CUDA cores	Frequency	GFLOPs(double)	Memory	Memory bandwidth
Nvidia Tesla K80	2 × 13(SMX)	2 × 2496	562MHz	2 × 1455	2 × 12GB	2 × 240GB/s
Nvidia Tesla P100	56(SM)	3584	1126MHz	4670	16GB	720GB/s

CPU below.Those evaluations are gotten from microway.com [5]

The result are shown below:

TABLE IV
K80 BENCHMARK RESULT

	AlexNet	Overfeat	GoogleNet	VGG (ver.a)	Speedup Over CPU
Caffe	365	1,187	1,236	1,747	(9x - 15x speedups)
TensorFlow	181	622	979	1,104	(4x - 10x speedups)
Theano	515	1,716	1,783	-	(8x - 16x speedups)
cuDNN fp32(Torch)	171	379	914	743	(9x - 12x speedups)
geometric average over frameworks	276	832	1,187	1,127	(9x - 11x speedups)

TABLE V
P100 BENCHMARK RESULT

	AlexNet	Overfeat	GoogleNet	VGG (ver.a)	Speedup Over CPU
Caffe	80	288	279	393	(35x - 70x speedups)
TensorFlow	46	144	253	277	(16x - 40x speedups)
Theano	161	482	624	2075	(19x - 43x speedups)
cuDNN fp32(Torch)	44	107	247	222	(33x - 41x speedups)
geometric average over frameworks	71	215	331	473	(29x - 42x speedups)

TABLE VI
CPU BENCHMARK RESULT

	AlexNet	Overfeat	GoogleNet	VGG
Caffe	4529	10350	18545	14010
TensorFlow	1823	5275	4018	7341
Theano	5278	13579	26829	38687
cuDNN	1838	3604	8234	9166
geometric average over frameworks	2991	7190	11326	13819

From the experiment, we can get that the speed up over CPU of Nvidia P100 is $4\times$ higher than speed up over CPU of Nvidia K80 on Tensorflow.

The comparison above shows the Nvidia Tesla P100 GPU is outperform than Nvidia Tesla K80 from every aspects.

IV. EVALUATION

A. Experimental setup

Mobile device, Google Compute Engine and terminal machine are used as three platforms we implement our system on. The parameters of them are shown in Table VII. There's a Python script running on the server, checking if there's any uploaded image. It would ensure the model is preloaded and ready to make inference anytime.

B. Experiment Evaluation

TABLE VIII
CNN MODEL TEST RESULT

Dataset	No. of classes	Test accuracy
GTSRB	43	96.4%
LISA Traffic Sign	14	99.8%
LISA Traffic Light	6	99.5%

TABLE IX
CNN MODEL SIZE

Model type	Model size	Test accuracy
Keras *(h5)*	4.51 MB	99.8%
coreML *(mlmodel)*	2.23 MB	83.2%

1) CNN model: The proposed CNN model was already trained by three kinds of datasets with test accuracy shown in Table VIII. Specifically, since the LISA dataset is a better fit to our proposed scenario, the experiments in the following would focus on LISA rather than GTSRB. In Table IX, we can see that after the conversion of Keras *h5* to coreML *mlmodel*, the size of the model reduced to the half of the origin. Moreover, the test accuracy (using the same test images from LISA Traffic Sign) decreased from 99.8% to 83.2%. The result of this experiment shows that the compressing process would affect the accuracy of the model.

TABLE X
INTERNET CONNECTION

	Wi-Fi	LTE	4G
Download	17.36 Mbps	7.17 Mbps	3.92 Mbps
Upload	11.90 Mbps	4.20 Mbps	1.33 Mbps
Ping	12 ms	29 ms	49 ms

2) Internet connection: To test the data-transfer time, the test images were all scaled to 32x32 (10 kB). In order to

TABLE VII
PLATFORMS SETUP

	Google cloud instance	PC	Mobile device (iPhone 6s)
CPU	2.2GHz Intel Xeon E5 v4(Boardwell)	I5 6400	1.85GHz-dual core-64-bit ARMv8-A
Memory	4.25G	15G	2G
GPU	Nvidia Tesla P100	Nvidia GTX950	PowerVR GT7600
GPU memory bandwidth	720GB/s	105GB/s	Unknown
CUDA cores/ALU cores	3584 CUDA cores	768 CUDA cores	192 ALU cores
Core frequency	1126MHz	1024MHz	Unknown

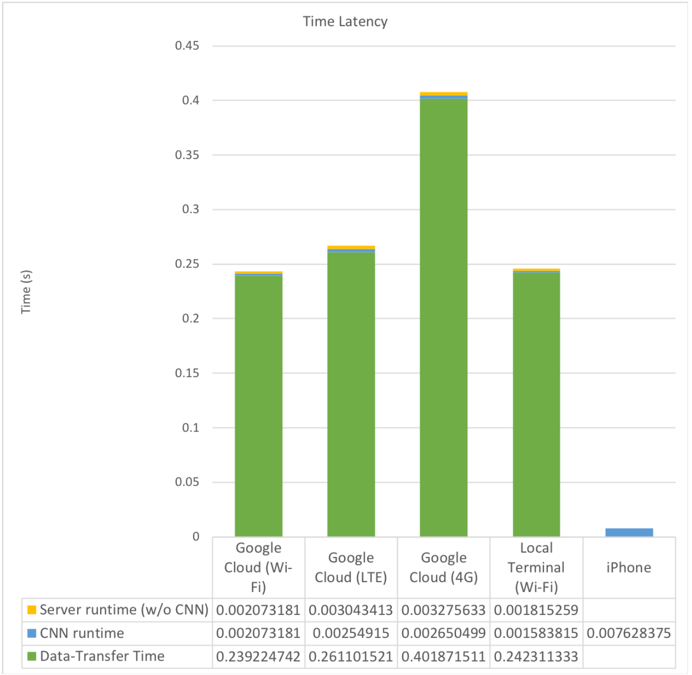
evaluate different kinds of scenario that the driver would encounter while driving, the test image would be uploaded from the smart phone with different Internet connection: Wi-Fi, LTE, and 4G. The download, upload speed with ping time are shown in Table X.

3) Running time: The time latency with different platforms and different time-consuming tasks are shown in Figure 9. It's obvious that most of the time are spent in transferring data, which includes the time of uploading the image from mobile device to server and the time of passing the result from server to mobile device. The GCE would need to spend 20 ms to 30 ms for read and preprocess the image and 20 ms to make an inference. The local device averagely uses less than 20 ms for the preproces, and 16 ms for the prediction. On the other hand, It took around 70 ms for the iPhone to make a prediction with coreML.

4) Summery: Generally, if the iPhone was connected to the Internet with LTE steadily, the speed of the inference could reach nearly 4 frames per second with GCE. Meanwhile, it could reach more than 10 frames per second with only coreML model implemented in the application. However, it would trade-off the inference accuracy with the small-size model on the mobile device. Therefore, we think it's better to use the server for inference because the 4-frame-per-second speed is enough for our scenario while the traffic signs would occur in the front view for at least 3 seconds with proper driving speed. If the server took too long with bad Internet connection, the application could still use the inference by the coreML model with lower accuracy after, say, 0.5 second time-out.

V. CONCLUSION

Firstly, we designed our own iOS application with the ability of measuring the speed and acceleration, and compare with the recognized traffic sign and traffic light from road image in nearly real-time. Secondly, we constructed our own 6-layer CNN architecture trained by GTSRB and LISA dataset for different scenarios with high accuracy. Thirdly, we implemented the model on three kinds of platforms: mobile device, GCE and PC. Evaluations had been done for deciding which GPU we can used in GCE. Finally, we evaluated the data-transfer time, running time with different Internet connection and on different platforms. With our proposed implementation, the system could make 4 inferences per second in average using the full-size model on GCE with higher accuracy, and 10 inferences per second using coreML model with lower accuracy, which is plausible for the proposed scenario.



	Google Cloud (Wi-Fi)	Google Cloud (LTE)	Google Cloud (4G)	Local Terminal (Wi-Fi)	iPhone
🟨 Server runtime (w/o CNN)	0.002073181	0.003043413	0.003275633	0.001815259	
🟦 CNN runtime	0.002073181	0.00254915	0.002650499	0.001583815	0.007628375
🟩 Data-Transfer Time	0.239224742	0.261101521	0.401871511	0.242311333	

Fig. 6. Time latency details

REFERENCES

[1] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, L. Tang "Neurosrgeon:Collaborative Intelligence Between the Cloud and Mobile Edge," Processing of the Twenty-second International Conference on Architectural Support for Programming Languages and Operating System. ACM, pp. 615–629, April 2017.

[2] A. Shstanov, P. Yakimov, "CNN Design For Real-Time Traffic Sign Recognition,"Procedia Engineering, vol.201., 2017, pp. 718–725.

[3] H. Habibi Aghdam, E. Jahani Heravi and D. Puig,"A practical Approach for Detection and Classification of Traffic signs using Convolutional Neural Networks," Robotics and Autonomous System, vol. 84., 2016, pp. 97–112.

[4] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in ARXIV, eprint arXiv:1409.1556, 2014

[5] Microway.com https://www.microway.com/hpc-tech-tips/deep-learning-benchmarks-nvidia-tesla-p100-16gb-pcie-tesla-k80-tesla-m40-gpus/