Road Signs Recognition Mobile Car

Yunhui Ma

***Abstract*—Self-driving cars implement neural network algorithms to recognize objects on the road. We can get an insight into self-driving car by learning and developing a simple version of image recognition mobile car. The way to approach is to apply neural network model into OpenMV Cam to recognize images and send the results to the mobile car. As a result, the mobile car can act in different ways based on different road signs in front of the openMV Cam.**

***Index Terms*—Deep Learning**
Convolutional neural network
Road signs recognition
Serial Communication
OpenMV Cam H7
MSP-exp432p401r

I. Introduction

A. History

The traditional autonomous driving systems relayed on the techniques of traditional image processing and pattern recognition. It was easily affected by the noise and the change of environment [1]. As a result, the prediction is not very accurate. If the weather changed such as heavy rain, the traditional image processing could predict wrong results. Recently, deep leaning has been used in autonomous driving car especially for the vision-based navigation system. One of the subsets of deep learning is convolutional neural network (CNN) significantly improves the prediction performance. By using a lot of image data sets, the CNN can solve many problems that the traditional image processing cannot solve [2]. A lot of self-driving companies have implemented the deep learning on their self-driving system.

In 2012, Geoffrey Hinton and two of his students, Alex Krizhevsky and Ilya Sutskever won the ImageNet 2012 competition with the highest accuracy. They first applied the algorithm of neural network [3]. Then, the neural network began to be applied in various fields. A lot of open resources such as TensorFlow, and Caffe provide a convenient method for people to use neural network.

B. Global Constraints

The first constraint of neural network is that it requires to collect a large amount of data to train [2]. The second constraint of neural network is that it needs a powerful machine to run the process. It is possible to train the neural network for a couple days or even a month. Especially for the 3D objects, the training and prediction process could consume a lot of resources such as electricity. In addition, the industries have been trying to produce the optimized chip which will work faster with neural network algorithm [4]. However, those new chips still need time to be widely used in our daily life. Moreover, the self-driving also needs to implement a powerful machine to deal with those data. Therefore, optimized the neural network without lowing the accuracy is necessary.

II. Motivation

The motivation of this project is to create a simple version of road sign recognition mobile car to get an insight into autonomous driving car. This project can provide a foundational method for students to study the principles of autonomous driving car. Besides, it can create a mobile car with a simple autonomous driving function to give people a chance to play and learn the knowledge of neural network.

III. Approach

- Used Tensorflow to create the road signs recognition convolutional neural network model.
- Loaded the CNN model into the OpenMV Cam.
- The OpenMV communicated with MSP-432p601r board.
- Activated the motor based on the prediction from OpenMV.

A. Team Organization

Personal project. Yunhui Ma worked on the whole project including designing, programming, implementing, and testing on the OpenMV and MSP board.

B. Plan

	Plan	Implementation
Week 3	Communication between the OpenMV and MSP board	Used TX/RX Serial communication. Two wires are required
Week 5	Collect image data sets and train the CNN model	Created the model and tested in the PC
Week 7	Try to load the CNN model into the OpenMV	Failed to load the model into the OpenMV. Tried to send images from the OpenMV to PC through WiFi hotspot
Week 9	Learn the STM32Cube.AI	Successfully loaded the CNN model into the OpenMV
Week 10	Implement the mobile car	The mobile car can perform different orders based on the images in front of the OpenMV

C. Standard

- TensorFlow: a free and open-source Python Library, developed by Google Brain team. It allows users to create their own trained neural network and use in different fields. It is de facto standard for symbolic math library and machine learning application. [5]
- STM32Cube.AI: a tool allows use to convert the trained neural network model to optimized C code and firmware source code which can be loaded by OpenMV Cam. It is developed by STM32 MCU [6].
- OpenCV: an open source computer vision library, created by Intel Corporation, Willow Garage, Itseez. It is de facto standard for computer vision [7].

D. Theory

- *Convolutional neural network*

Deep learning Neural network is one of the subsets of machine learning. There are two types of neural network which are the typical neural network and convolutional neural network (CNN). The convolutional neural network is used in this project.

The CNN has three components including input, hidden layer, and output layer [8]. Fig.1 shows the total processing of the CNN. For the input layer, it is the input image array. Each pixel has its own value. For example, a size of input image is 28x28, therefore it includes 784 weights.

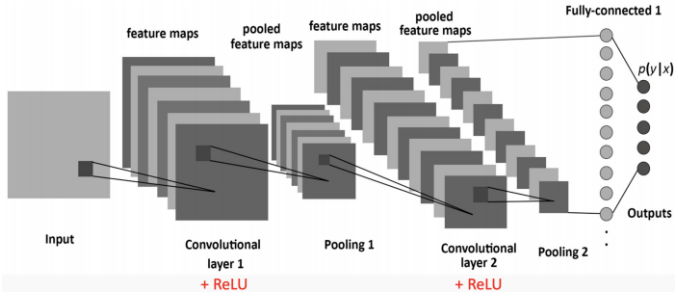

Fig.1 Convolutional neural network layers. Credit: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

In the convolutional layers, there a 3 by 3 convolution filter crosses the input image from top left to bottom (see Fig.2) [8]. It is a dot product process. A feature map is generated after the filter crosses the input image. ***f(x)*** is an activation function (ReLu) which is applied to the feature map. This function changes the negative values to zero and keep the positive value. The convolution operation can be applied multiple times by different filters, and it generates more different feature maps.

$$f(x) = \begin{cases} x & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases} \quad (1)$$

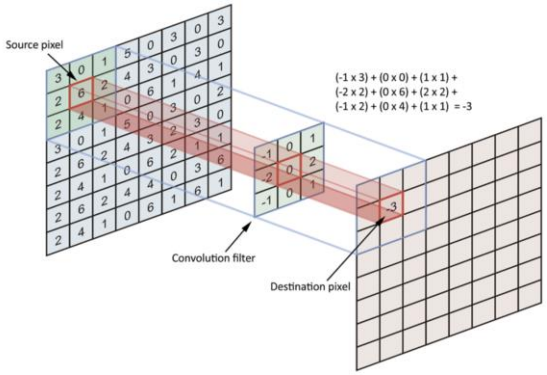

Fig.2 Convolution operation. Credit: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

During the convolutional operation, Fig.2 shows that the feature map is down sample compared to the original image. In order to make sure the output has the same size as original image. The padding operation can be applied. The convolution filter crosses the edge of the input image at the beginning to the end. In this project, it uses the same padding. (see Fig.3)

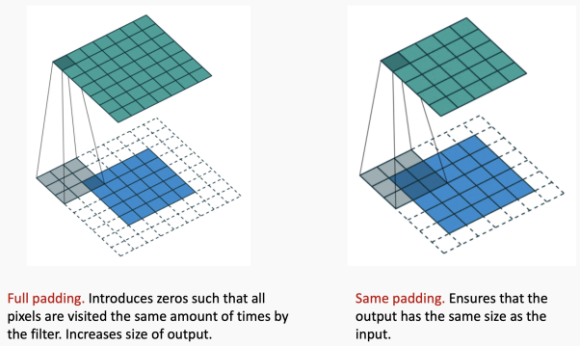

Fig. 3 Padding operation. Credit: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

For the pooling layer, the Max Pooling is applied in this project. Fig.4 shows a down sampling of input image by a half. It takes the maximum value and puts it to the output. The Max Pooling can reduce the dimensionality of the network to increase the processing speed.

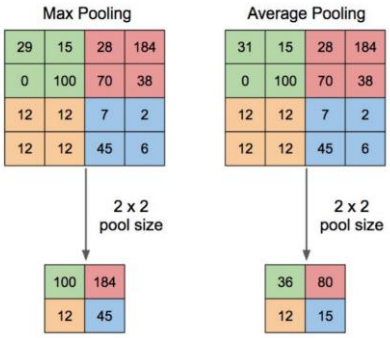

Fig.4 Pooling Layer. Credit: https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf

The fully connected layer (dense layer) is used before the outputs of CNN. It converts the 3D dimension into a 1D dimension. The process is called Flatten. The final predicted probability uses the softmax function ***P (y = j | x).*** The function produces a discrete probability vector which represents the result of prediction. The input image is identified by the higher probability prediction.

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}} \qquad (2)$$

Fig.5 is the overall CNN structure for this project. The input image is reshaped to 50*50 size.

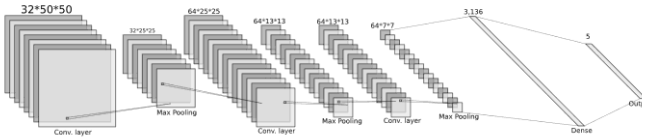

Fig.5 CNN structure for this project

- *Communication*

Another important theory for this project is the Serial communication. The data is transmitted by the way of binary pulses. A logic high represents 1, and logic low represents 0. In this project, it only requires one-way transmission. The advantage of serial communication is that it requires 2 wires only.

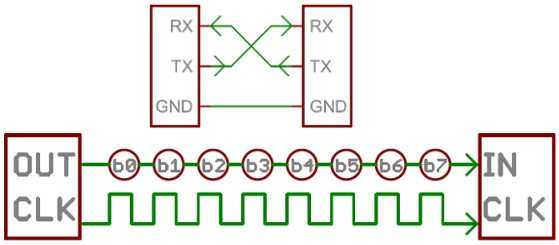

Fig.5 Serial interface. Credit: https://learn.sparkfun.com/tutorials/serial-communication/all

E. Software / Hardware

- *Software*

To perform the CNN, Python, TensorFlow, and OpenCV are used for this project. The TensorFlow is an end-to-end open source machine learning. The important functions that TensorFlow provides include Conv2D, MaxPooling2D, Flatten, and Dense. Those functions are beneficial to create the convolutional neural network. OpenCV is another library in Python. It is used to load the images and labels corresponding to those images. In this project, Xcode is used to modify the code as a text editor. The Python code (CNN program) is run by Mac terminal.

The STM32Cube.AI is a tool which can convert the CNN model to optimized C code. It generates a firmware that can be loaded into the OpenMV camera. The detail process of STM32Cube.Ai can be found on the STM32MCU wiki website [6].

- *Hardware*

For hardware, the OpenMV Cam H7 (Fig.6) is used to capture and predict the images. The MSP-exp432p401r Kit (Fig.7) is a mobile car system which performs different actions based on the result sent from the OpenMV Cam.

Fig.6 OpenMV

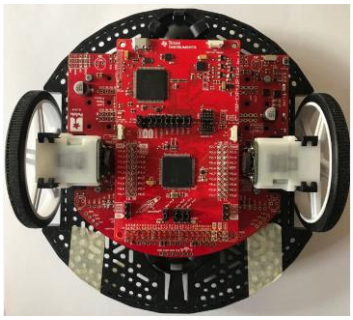

Fig.7 MSP-exp432p401r LaunchPad Kit

Energia IDE is used to modify the C code and upload to the MSP-exp432p401r board. OpenMV IDE modifies the Python code and upload to the OpenMV Cam.

The schematic connection between the OpenMV and MSP-exp432p401r board can be seen in the image below.

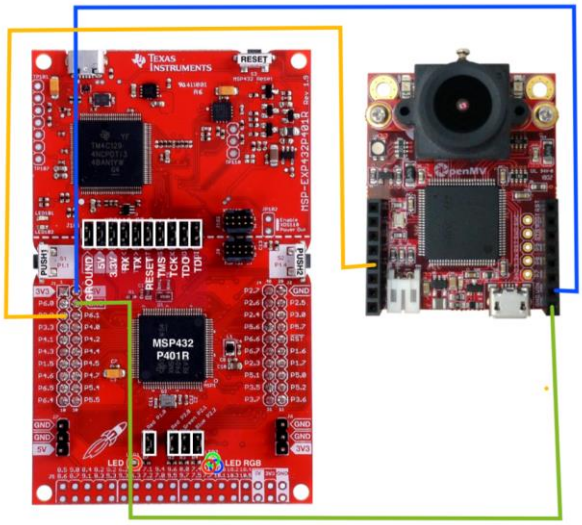

Fig.8 Schematic diagram for openMV and MSB board

- *Collecting Data*

There are five classes of image data for this project. The image data sets include stop sign, left turn, right turn, and speed limit. Each class has 50 images.

Fig.9 Road Signs data sets

The last one is the background class (Fig. 10), and it includes 30 blank images. The way to create a background class is because the prediction function sometimes identifies other objects as one of the road signs.

Fig.10 Background data set

- *Code for loading and training images*

Fig.10 shows the way to read images by using OpenCv library. The images are read in gray scale because the gray scale only has one dimension.

```
def create_training_data():
    for category in CATEGORIES:
        path = os.path.join(DATA_DIR, category)
        class_num = CATEGORIES.index(category)
        print(class_num)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                training_data.append([new_array, class_num])
            except Exception as e:
                pass

create_training_data()
```

Fig.10 Python code for reading images

Fig.11 is the Python code to create the CNN model by using TensorFlow. It includes two convolutions and three Max Pooling. The CNN model is saved as traffic_sign.h5 file.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 1), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(5, activation='softmax'))

#Training the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=8)

model.save('/Users/yunhuima/Documents/Python/traffic_sign.h5')
```

Fig.11 Python code for training images, the process of convolutional neural network

- Code for OpenMV and MSP board

To successfully convert the CNN model, Python 3.7.6, TensorFlow 2.0.0, and TensorFlow.keras 2.2.4-tf are required. Other version of Python and TensorFlow can fail the converted process. Fig.12 shows that the OpenMV loads the CNN model. For this project, the prediction function only predicts whole image captured from the OpenMV camera.

```
# [STM32Cube.AI] Initialize the network
net = nn_st.loadnnst('network')

traffic = ["Others","Speed_limit_30", "Stop_Sign", "Right_turn","Left_turn"]

while(True):
    clock.tick()                # Update the FPS clock.
    img = sensor.snapshot()  # Take a picture and return the image.
    data = []
    x = 0
    # [STM32Cube.AI] Run the inference

    out = net.predict(img)
    if out.index(max(out)) == 0:
        img.draw_string(0, 0, str(traffic[0]))
        x = 0
        #print(traffic[0])
```

Fig.12 CNN model used in OpenMV

Fig.13 shows that the communication between OpenMV and MSP board. Based on different numbers MSP board received, it can control the mobile car to run in different way.

```
data.append(x)
data_out = json.dumps(set(data))
uart.write(data_out)
```

```
void loop() {
  if(Serial1.available() > 0)
  {
    String s = detectString();
    int num = s.toInt();
    if(num == 0)
    {
      left_wheel = 20;
      right_wheel = 20;
      Serial.println("Other");
    }
    else if(num == 1)
    {
      right_wheel = 50;
      left_wheel = 50;
      Serial.println("Speed_limit_30");
    }
```

Fig.13 OpenMV transmits data and MSP receives data

F. System Build / Operation

The first step is to create a CNN model by using TensorFlow in the PC perform. Testing and evaluating the model are necessary step which make sures the recognition is correct.

Second, used the STM32Cube.AI to convert the CNN model into a firmware file which can be loaded by OpenMV Cam.

Third, using USB to connect the OpenMV and PC. The OpenMV loads and flashes the firmware file by using OpenMV IDE. Testing the results in the OpenMV Cam. Using the prediction function to test road sign images.

Fourth, by using the serial communication, the OpenMV sends the resulting data to the MSP-exp432p401 board.

Fifth, using the Energia IDE to modify the code to control the mobile car. Based on the results sent from the OpenMV, the mobile car is set by different commands. For this project, the commands include “stop,” “Left turn,” “right turn,” “go forward.” Fig.14 is the final installation for Road Signs Recognition Mobile Car.

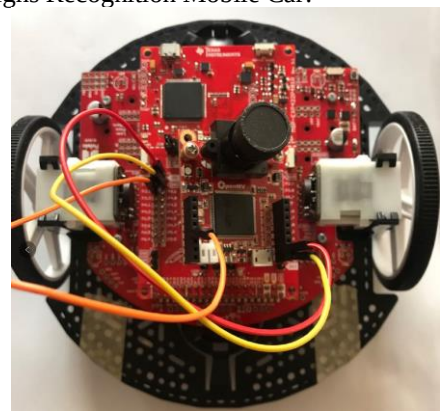

Fig.14 Installation

IV. RESULTS

The result meets the project requirement. The accuracy of CNN model is high enough for testing process. The CNN model successfully loads into the OpenMV Cam. The OpenMV can predict the image and send the result to MSP board to control the mobile car. The prediction process is executable in the OpenMV.

A. Description of Results

The accuracy for this model is about 0.98 which is high enough to use. The loss is 0.0613 which is small. Since there are not many images, the training process is fast. 9 epochs are used in this process. Total parameters are 67,269 (see Fig.15, Fig.16)

```
Epoch 1/8
8/8 [==============================] - 0s 37ms/step - loss: 1.5645 - accuracy: 0.2217
Epoch 2/8
8/8 [==============================] - 0s 39ms/step - loss: 1.4321 - accuracy: 0.4565
Epoch 3/8
8/8 [==============================] - 0s 40ms/step - loss: 1.1491 - accuracy: 0.6522
Epoch 4/8
8/8 [==============================] - 0s 42ms/step - loss: 0.6628 - accuracy: 0.9348
Epoch 5/8
8/8 [==============================] - 1s 74ms/step - loss: 0.3558 - accuracy: 0.9087
Epoch 6/8
8/8 [==============================] - 0s 43ms/step - loss: 0.1642 - accuracy: 0.9609
Epoch 7/8
8/8 [==============================] - 0s 39ms/step - loss: 0.0865 - accuracy: 0.9913
Epoch 8/8
8/8 [==============================] - 0s 44ms/step - loss: 0.0613 - accuracy: 0.9870
```

Fig.15 the CNN trained model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 50, 50, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 5)                 11525
=================================================================
Total params: 67,269
Trainable params: 67,269
Non-trainable params: 0
```

Fig.16 Parameters for CNN model

Fig.17 is the result of the prediction on the OpenMV. Basically, the prediction function can identify road signs correctly. Switching the image results in the wrong prediction, but it eventually predicts the image with correct result.

Fig.17 Prediction result on OpenMV

The table below shows that the long distance between image and the OpenMV camera can result in the failure of prediction. On average, the OpenMV can make right prediction when the distance between the images and the camera is less than 13.95cm.

	Distance between image and camera for wrong prediction
Stop	>12.6cm
Right turn	>14.5cm
Left turn	>12.8cm
Speed limit	>15.9cm
average	13.95cm

Table.1 distance and wrong prediction

For the communication, the delay between the OpenMV and MSP board is less than 0.5 second. The MSP board receives the result immediately. The mobile performs the action correctly if the OpenMV can predict the image correctly. Table 2 shows that the stop sign has low accuracy compared to others. Overall, the accuracy is acceptable.

	Number of Tests	Number of Right actions on mobile car	Percent
Stop	10	8	80%
Right turn	10	9	90%
Left turn	10	10	100%
Speed limit	10	10	100%

Table.2 Testing result

The Demo can be found by this link.
https://www.youtube.com/watch?v=uHticAYpQQI
https://www.youtube.com/watch?v=tsjZjavVysU

B. Discussion of results

The CNN model itself has high accuracy. The higher accuracy can result in the over feeding, meaning that the prediction function would identify other irrelevant images as one of the class in the CNN model. For example, it can identify a tennis ball as the stop sign, the wall as the left turn sign, my face as the speed limit sign. Therefore, I create an empty class in order to avoid this situation. But including the empty class makes the process ineffective, and it requires an empty background behand the images of road sign.

For this project, the hardest thing is to apply the CNN model into the OpenMV cam. It is hard to find the example of applying neural network on the OpenMV. However, with STM32Cube.AI, the CNN model successfully is loaded into the OpenMV cam.

A couple things can be improved. First, it can train a better CNN model by reading more images. Also, it needs more tests and evaluations. Having a better CNN model can avoid the over feeding problem. Second, instead of taking the whole images to test, it is better to capture the road signs location inside the whole image. For example, it can take a certain part of the image to test and see if it is one of the road signs. Repeating this process from the top left bottom right until it finds the road signs. But it will require more algorithms. Third, since the operating the CNN in the OpenMV slows down the speed (4fps only), it can add another powerful machine to deal with data. The OpenMV can send the image data to the MSP board which can do the prediction. By doing this, it will require to save the CNN model data and write the prediction function on the MSP board. Finally, it is a good learning experience for us to apply the CNN model into other devices.

REFERENCES

[1] Pomerleau D.A. (1993) Knowledge-Based Training of Artificial Neural Networks for Autonomous Robot Driving. In: Connell J.H., Mahadevan S. (eds) Robot Learning. The Springer International Series in Engineering and Computer Science (Knowledge Representation, Learning and Expert Systems), vol 233. Springer, Boston, MA

[2] Mahony, N., Campbell, S., Carvalho, A. Harapanahalli, S., Hernandez, G., Krpalkova, L. “Deep Learning vs. Traditional Computer Vision” (n.d.).

[3] Krizhevsky, A., Sutskever, I., Hinton, G. E., (3 December 2012). "ImageNet classification with deep convolutional neural networks". Nips'12. Curran Associates Inc.: 1097–1105.

[4] B. E. Boser, E. Sackinger, J. Bromley, Y. leCun and L. D. Jackel, "Hardware requirements for neural network pattern classifiers: a case study and implementation," in IEEE Micro, vol. 12, no. 1, pp. 32-40, Feb. 1992, doi: 10.1109/40.124378.

[5] "TensorFlow: Open source machine learning" "It is machine learning software being used for various kinds of

perceptual and language understanding tasks" — Jeffrey Dean, minute 0:47 / 2:17 from YouTube clip

[6] Anonymous. “How to integrate STM32Cube.AI generated code in OpenMV ecosystem” (n.d). https://wiki.st.com/stm32mcu/wiki/How_to_integrate_STM32Cube.AI_generated_code_in_OpenMV_ecosystem

[7] Pulli, Kari; Baksheev, Anatoly; Kornyakov, Kirill; Eruhimov, Victor (1 April 2012). "Realtime Computer Vision with OpenCV". Queue: 40:40–40:56. doi:10.1145/2181796.2206309 (inactive 2020-05-21).

[8] Stewart, M. “Simple Introduction to Convolutional Neural Networks.” https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac. Feb 26, 2019