Pattern recognition

Traffic Sign Detection Project

Thanks Mo.T

2021-07-28

Traffic Sign Detection Project

Do you have experience with driverless vehicles? What is the driverless vehicle seeing and doing when you sit comfortably in the back seat? It's probably watching its surroundings with a wider field of view than you. There may also be traffic signs to be monitored. It will ensure that the vehicle obeys the traffic rules and protects you from terrifying accidents.

01

Prepare dataset for training

02

Train the machine with the prepared dataset.

03

The sophisticated machine detects traffic signs with high accuracy.

You must have heard about the self-driving cars in which the passenger can fully depend on the car for traveling. But to achieve level 5 autonomous, it is necessary for vehicles to understand and follow all traffic rules.

About Traffic Signs Recognition

Can you sit comfortably in the back seat of a fast-paced driverless vehicle?

Can a machine have human-like judgment and control?

Of course, it can't follow humans. But machines can also be trained, and a well-trained machine can make better decisions than humans in some respects. Quick and error-free. Because the machine doesn't drink...

The important thing in driverless vehicles is the detection of traffic signs. It ensures driving safety and steers the vehicle to take you exactly to your desired destination.

There are several different types of traffic signs like speed limits, no entry, traffic signals, turn left or right, children crossing, no passing of heavy vehicles, etc. Traffic signs classification is the process of identifying which class a traffic sign belongs to.

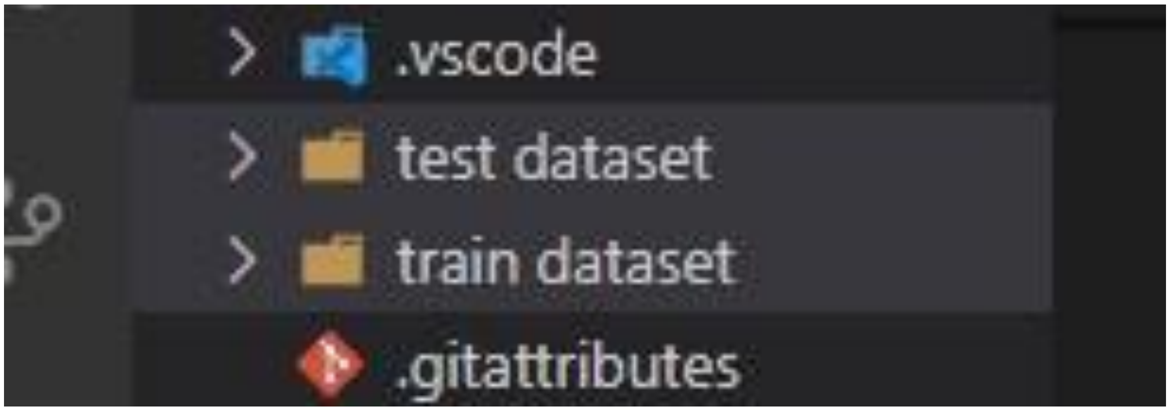
This project includes the ability to read and understand traffic signs through this model after building a deep neural network model that classifies traffic signs in images into various categories.

9.png
10.png
11.png
12.png
20.png
21.png
22.png
23.png
31.png
32.png
33.png
34.png

Dataset for machine training

The project used an online dataset with more than 50000 images of various traffic signs.

These images can be classified into 43 different classes.



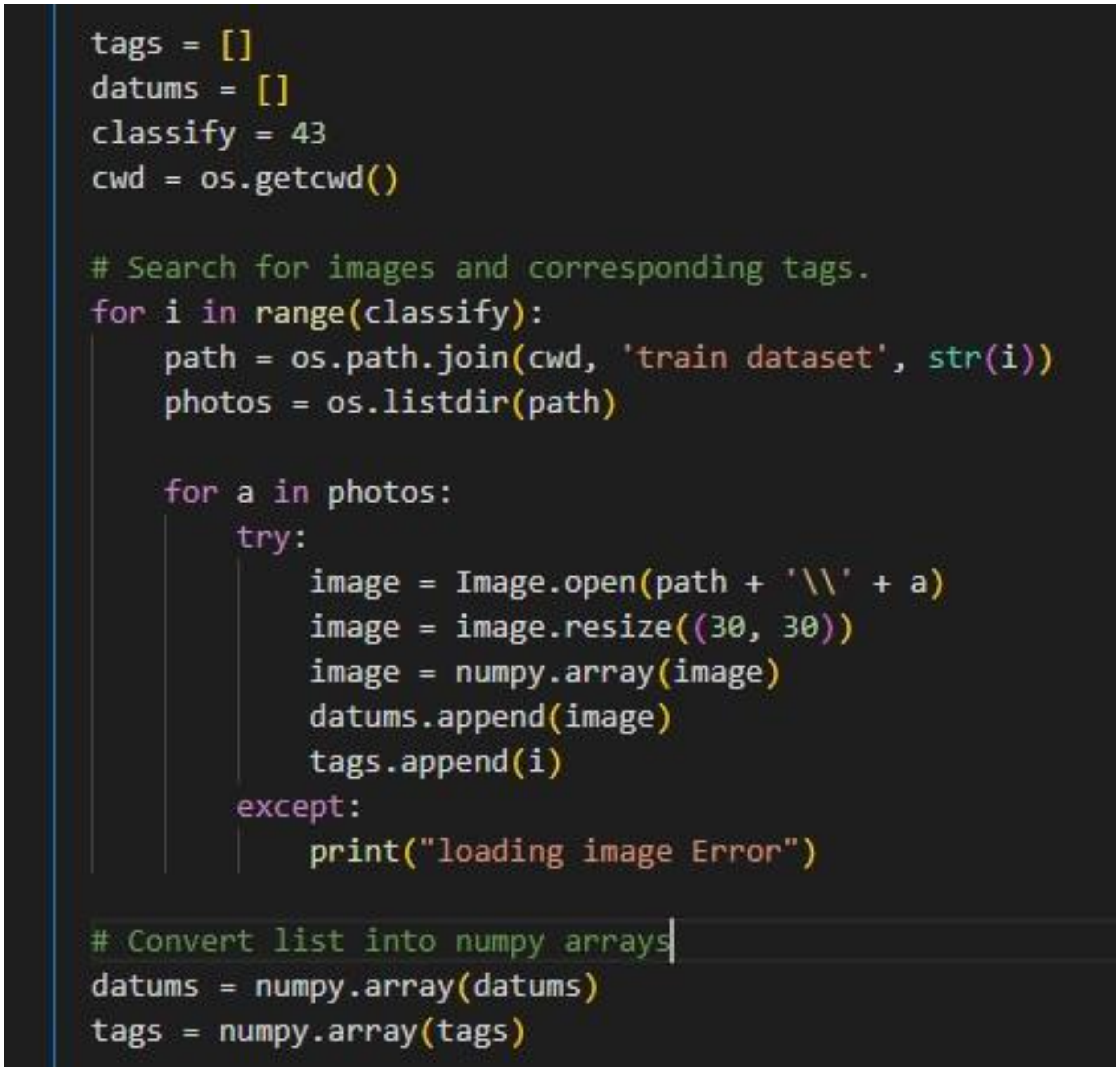
Train your machine

1. Explorer dataset

   There are 43 different classes of images in the 'train dataset' folder, which are separated into folders with numerical names.

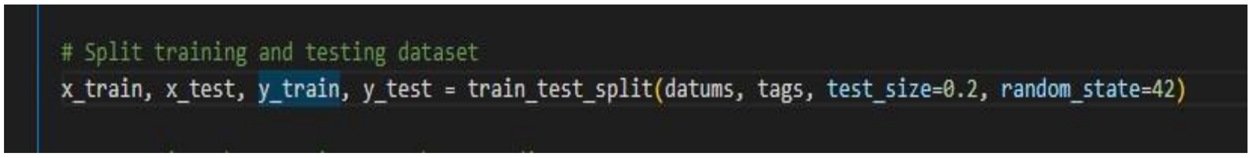
   We use os module to read images of all classes and add images and corresponding image tags to the tag list.

   We assigned the image and its tags to the *datums* and *tags* variables.

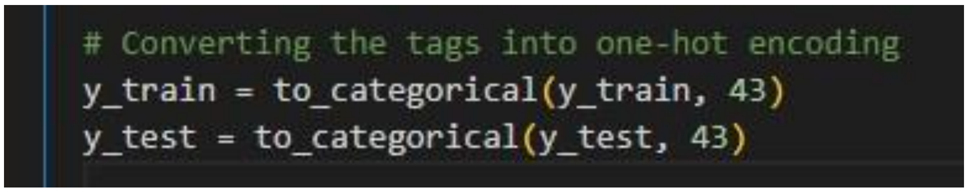

```
tags = []
datums = []
classify = 43
cwd = os.getcwd()

# Search for images and corresponding tags.
for i in range(classify):
    path = os.path.join(cwd, 'train dataset', str(i))
    photos = os.listdir(path)

    for a in photos:
        try:
            image = Image.open(path + '\\' + a)
            image = image.resize((30, 30))
            image = numpy.array(image)
            datums.append(image)
            tags.append(i)
        except:
            print("loading image Error")

# Convert list into numpy arrays
datums = numpy.array(datums)
tags = numpy.array(tags)
```

To feed these lists to the model we need to convert them to numpy arrays.

Then, using the train_test_split() function of the sklearn package, it is split into train data and test data.


```
# Split training and testing dataset
x_train, x_test, y_train, y_test = train_test_split(datums, tags, test_size=0.2, random_state=42)
```

Convert the tags in y_train and y_test to one-hot encoding using the to_categorical function of the Keras.util package.

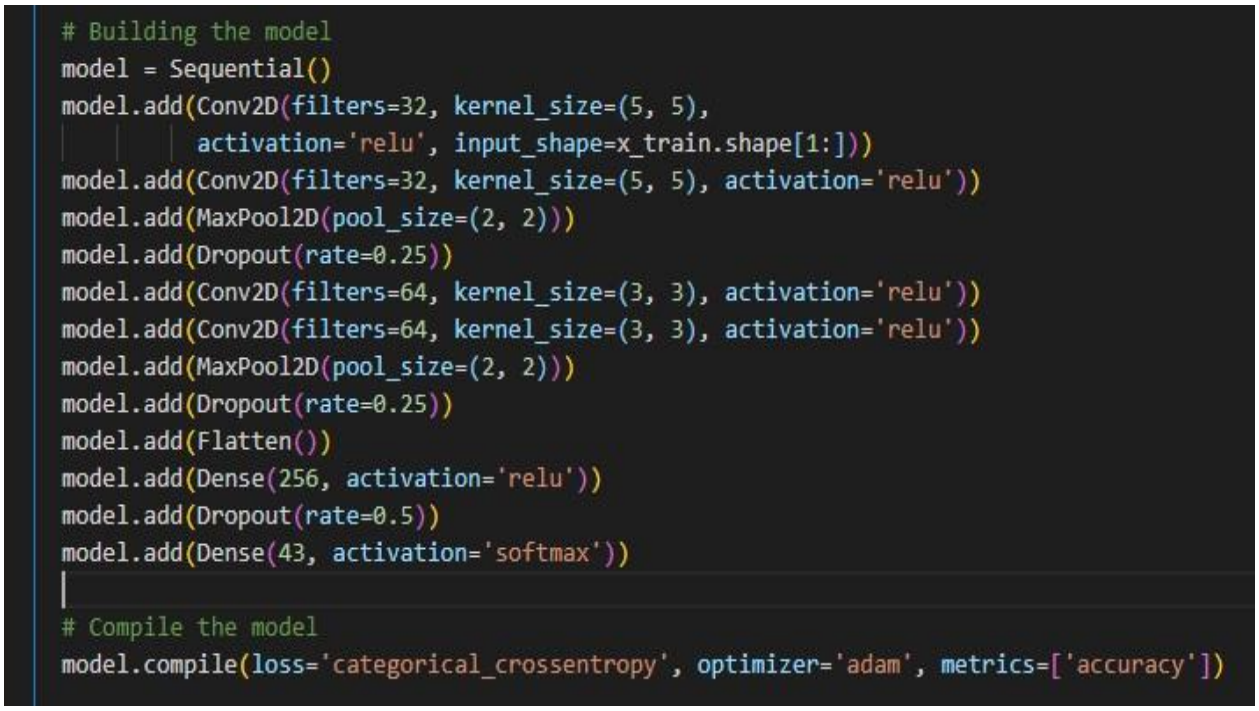


```
# Converting the tags into one-hot encoding
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```

2. Build CNN model

We build a CNN model to classify images into the corresponding classification.

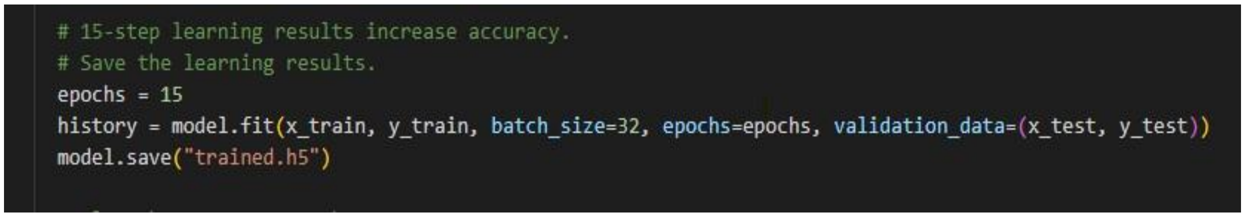
CNN is **a type of deep learning model for processing data** that has a grid pattern, such as images, which is inspired by the organization of animal visual cortex [13, 14] and designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns.



```
# Building the model
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5, 5),
                 activation='relu', input_shape=x_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5, 5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

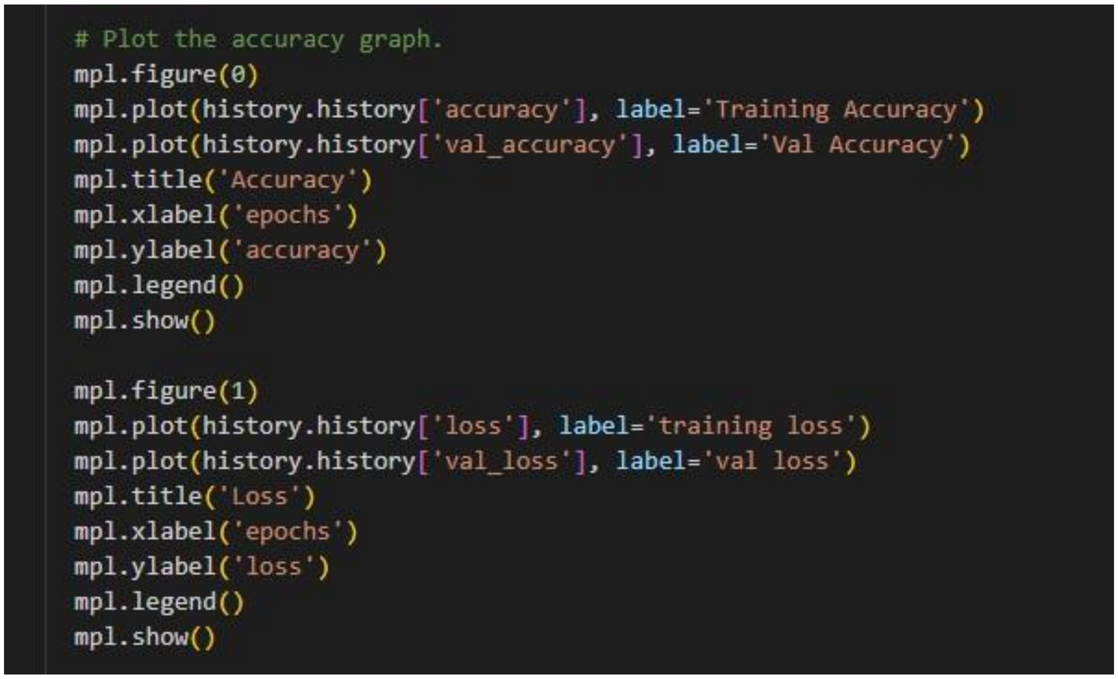
After building the model, use model.fit() to train the model. After 15 epochs, the accuracy was definitely high.

Then, the model is saved as “trained.h5”.



```python
# 15-step learning results increase accuracy.
# Save the learning results.
epochs = 15
history = model.fit(x_train, y_train, batch_size=32, epochs=epochs, validation_data=(x_test, y_test))
model.save("trained.h5")
```

Finally, draw a graph for accuracy and loss.



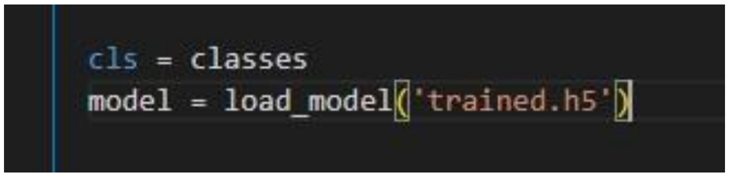
```python
# Plot the accuracy graph.
mpl.figure(0)
mpl.plot(history.history['accuracy'], label='Training Accuracy')
mpl.plot(history.history['val_accuracy'], label='Val Accuracy')
mpl.title('Accuracy')
mpl.xlabel('epochs')
mpl.ylabel('accuracy')
mpl.legend()
mpl.show()

mpl.figure(1)
mpl.plot(history.history['loss'], label='training loss')
mpl.plot(history.history['val_loss'], label='val loss')
mpl.title('Loss')
mpl.xlabel('epochs')
mpl.ylabel('loss')
mpl.legend()
mpl.show()
```

Let’s Recognize Images

The interface of the test app was built using Tkinter.

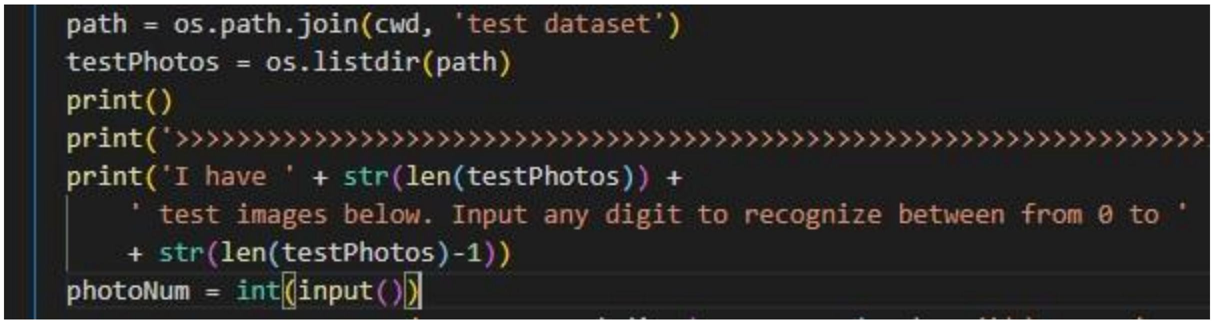
The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.)

It’s loaded “trained.h5” model and classify the images.



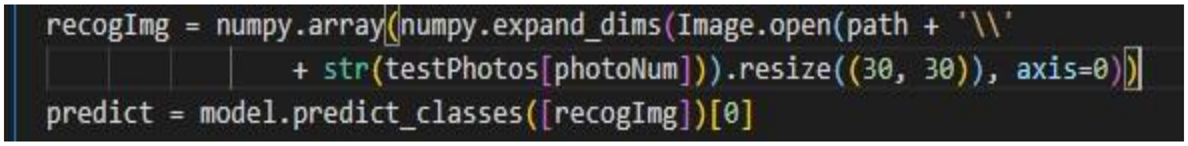
```
cls = classes
model = load_model('trained.h5')
```

Then input image file number your want.



```
path = os.path.join(cwd, 'test dataset')
testPhotos = os.listdir(path)
print()
print('>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
print('I have ' + str(len(testPhotos)) +
    ' test images below. Input any digit to recognize between from 0 to '
    + str(len(testPhotos)-1))
photoNum = int(input())
```

model.predict_classes() returns a number between 0~42 which represents the class it belongs to.



```
recogImg = numpy.array(numpy.expand_dims(Image.open(path + '\\'
                + str(testPhotos[photoNum])).resize((30, 30)), axis=0))
predict = model.predict_classes([recogImg])[0]
```

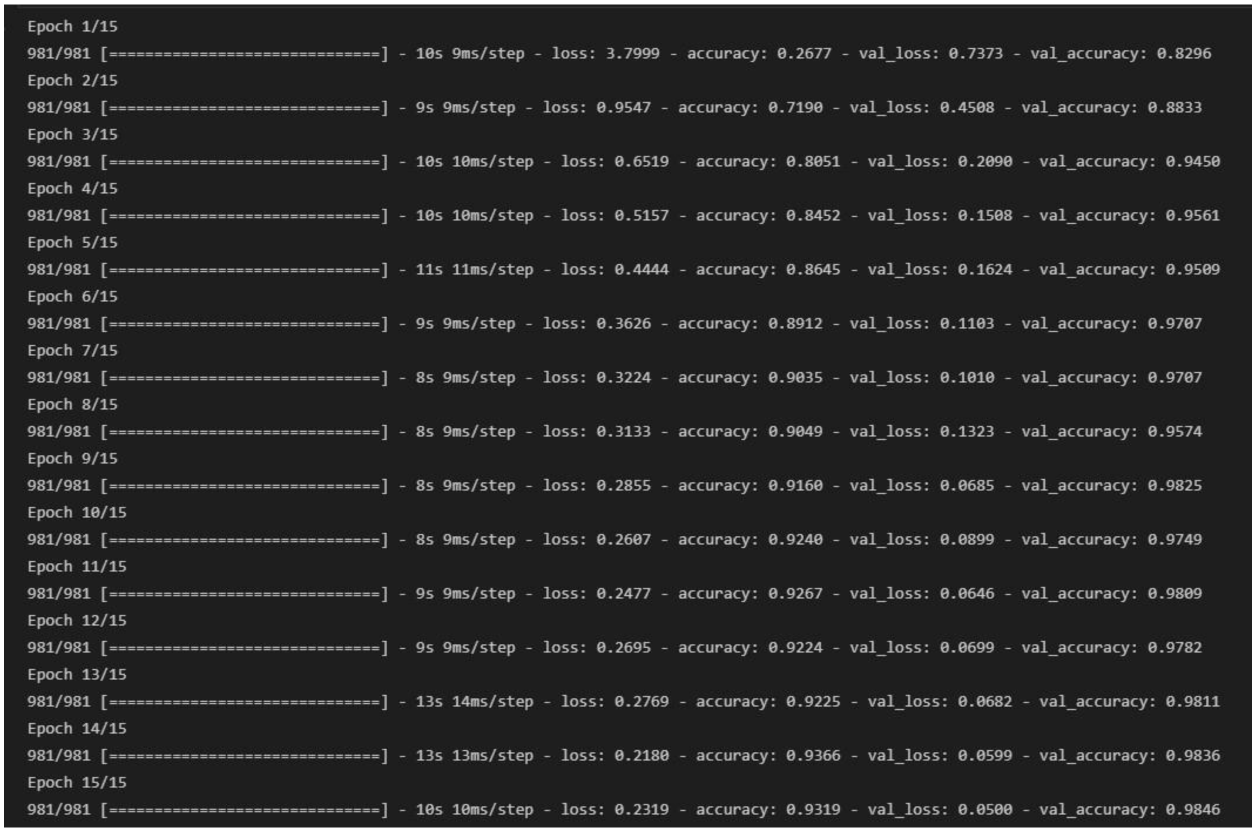
Execution result

First this machine will be studied.

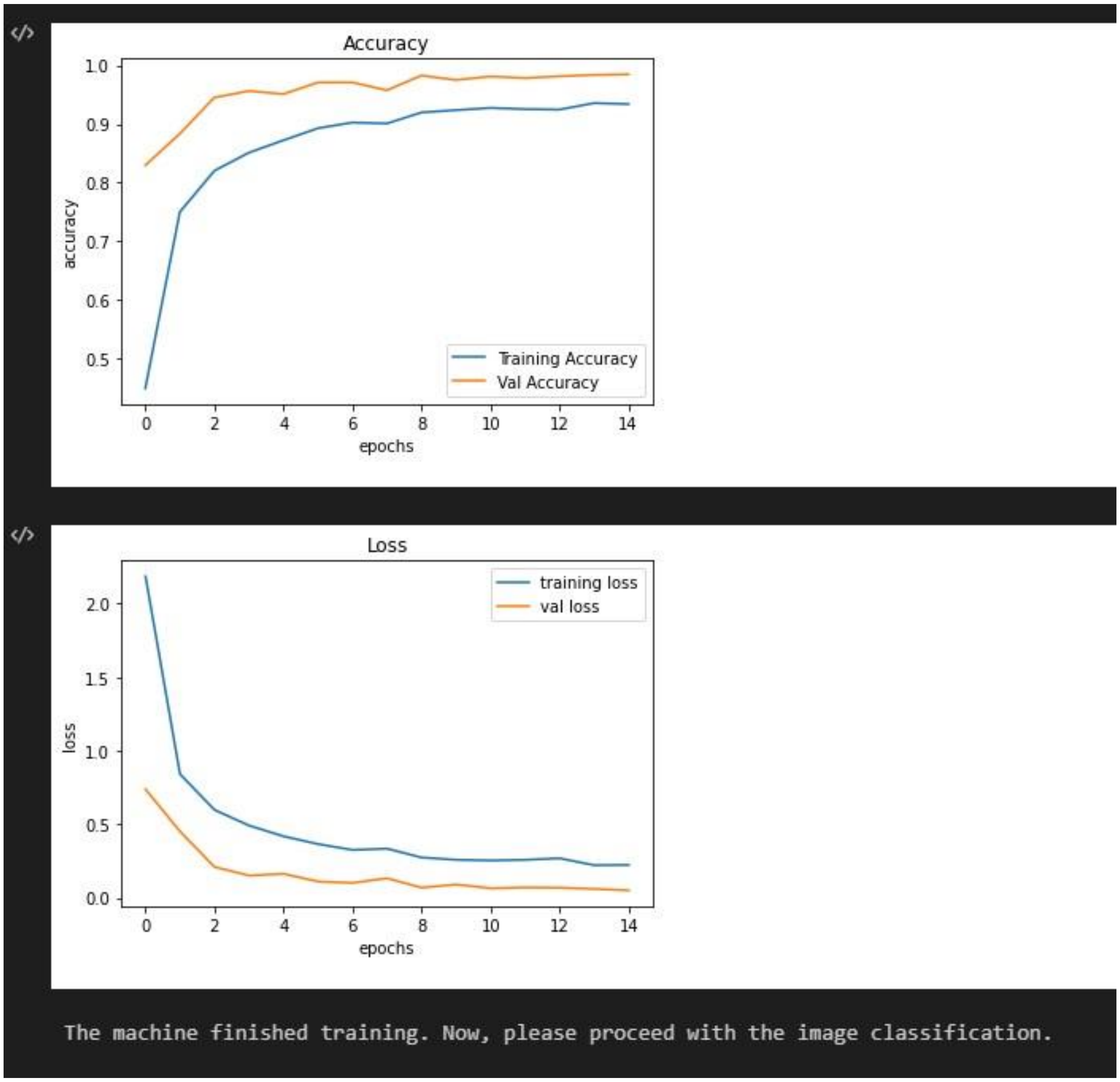
As you can see, at epoch 15 accuracy is surely incresed. Almost up to 98%.

Machine also plot its accuary-plot, and loss-plot.

Look at below figures.

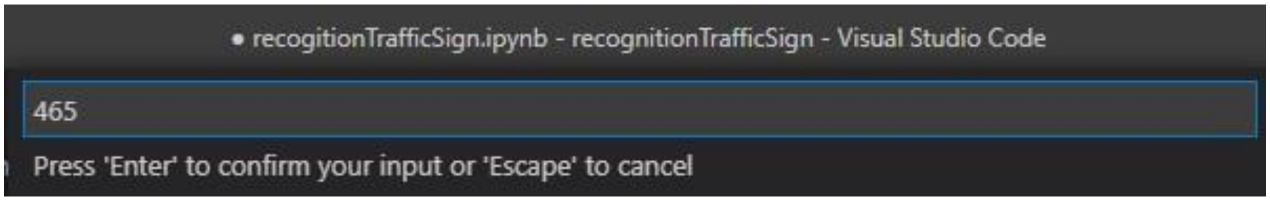


```
Epoch 1/15
981/981 [==============================] - 10s 9ms/step - loss: 3.7999 - accuracy: 0.2677 - val_loss: 0.7373 - val_accuracy: 0.8296
Epoch 2/15
981/981 [==============================] - 9s 9ms/step - loss: 0.9547 - accuracy: 0.7190 - val_loss: 0.4508 - val_accuracy: 0.8833
Epoch 3/15
981/981 [==============================] - 10s 10ms/step - loss: 0.6519 - accuracy: 0.8051 - val_loss: 0.2090 - val_accuracy: 0.9450
Epoch 4/15
981/981 [==============================] - 10s 10ms/step - loss: 0.5157 - accuracy: 0.8452 - val_loss: 0.1508 - val_accuracy: 0.9561
Epoch 5/15
981/981 [==============================] - 11s 11ms/step - loss: 0.4444 - accuracy: 0.8645 - val_loss: 0.1624 - val_accuracy: 0.9509
Epoch 6/15
981/981 [==============================] - 9s 9ms/step - loss: 0.3626 - accuracy: 0.8912 - val_loss: 0.1103 - val_accuracy: 0.9707
Epoch 7/15
981/981 [==============================] - 8s 9ms/step - loss: 0.3224 - accuracy: 0.9035 - val_loss: 0.1010 - val_accuracy: 0.9707
Epoch 8/15
981/981 [==============================] - 8s 9ms/step - loss: 0.3133 - accuracy: 0.9049 - val_loss: 0.1323 - val_accuracy: 0.9574
Epoch 9/15
981/981 [==============================] - 8s 9ms/step - loss: 0.2855 - accuracy: 0.9160 - val_loss: 0.0685 - val_accuracy: 0.9825
Epoch 10/15
981/981 [==============================] - 8s 9ms/step - loss: 0.2607 - accuracy: 0.9240 - val_loss: 0.0899 - val_accuracy: 0.9749
Epoch 11/15
981/981 [==============================] - 9s 9ms/step - loss: 0.2477 - accuracy: 0.9267 - val_loss: 0.0646 - val_accuracy: 0.9809
Epoch 12/15
981/981 [==============================] - 9s 9ms/step - loss: 0.2695 - accuracy: 0.9224 - val_loss: 0.0699 - val_accuracy: 0.9782
Epoch 13/15
981/981 [==============================] - 13s 14ms/step - loss: 0.2769 - accuracy: 0.9225 - val_loss: 0.0682 - val_accuracy: 0.9811
Epoch 14/15
981/981 [==============================] - 13s 13ms/step - loss: 0.2180 - accuracy: 0.9366 - val_loss: 0.0599 - val_accuracy: 0.9836
Epoch 15/15
981/981 [==============================] - 10s 10ms/step - loss: 0.2319 - accuracy: 0.9319 - val_loss: 0.0500 - val_accuracy: 0.9846
```



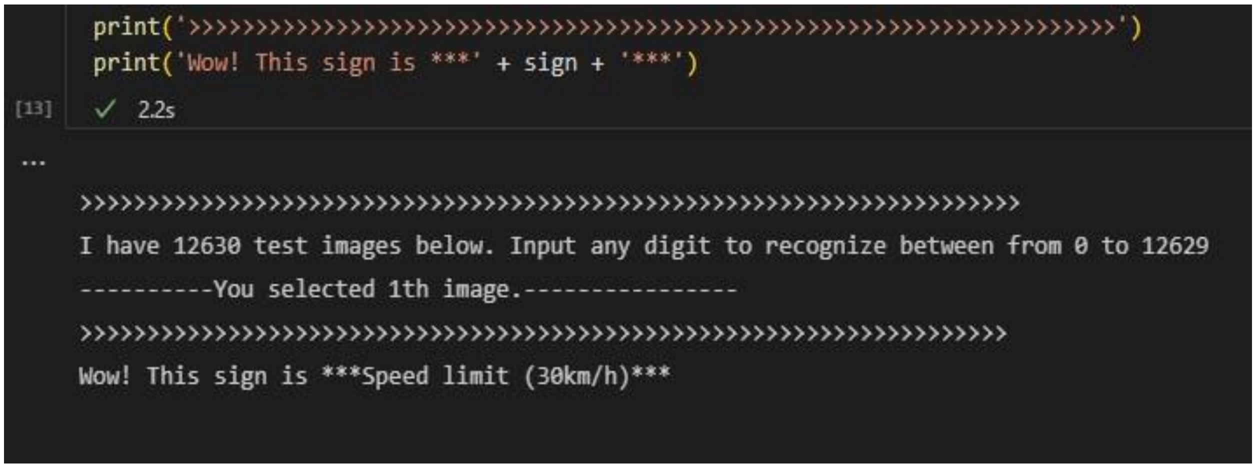
```
The machine finished training. Now, please proceed with the image classification.
```

Input image file number your want


• recogitionTrafficSign.ipynb - recognitionTrafficSign - Visual Studio Code

```
465
Press 'Enter' to confirm your input or 'Escape' to cancel
```

Then the result be showed.



```
print('>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>')
print('Wow! This sign is ***' + sign + '***')
[13]  ✓ 2.2s
...
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
I have 12630 test images below. Input any digit to recognize between from 0 to 12629
----------You selected 1th image.----------------
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Wow! This sign is ***Speed limit (30km/h)***
```

Concluding remarks

So far, you have conducted an experiment on the traffic sign identification system based on model creation using deep learning.

Python has a lot of useful libraries and functions that make it easy to implement deep learning algorithms for machine learning.

⁕ Focus in the experiment

1. Converting images sorted by class into a NumPy array.
2. One-hot encoding this.
3. Creating and storing Keras sequential models.
4. Import the model and classify the image into the corresponding class with predict_classes() function.