Demo: traffic lights recognize

Author: Jim Xie

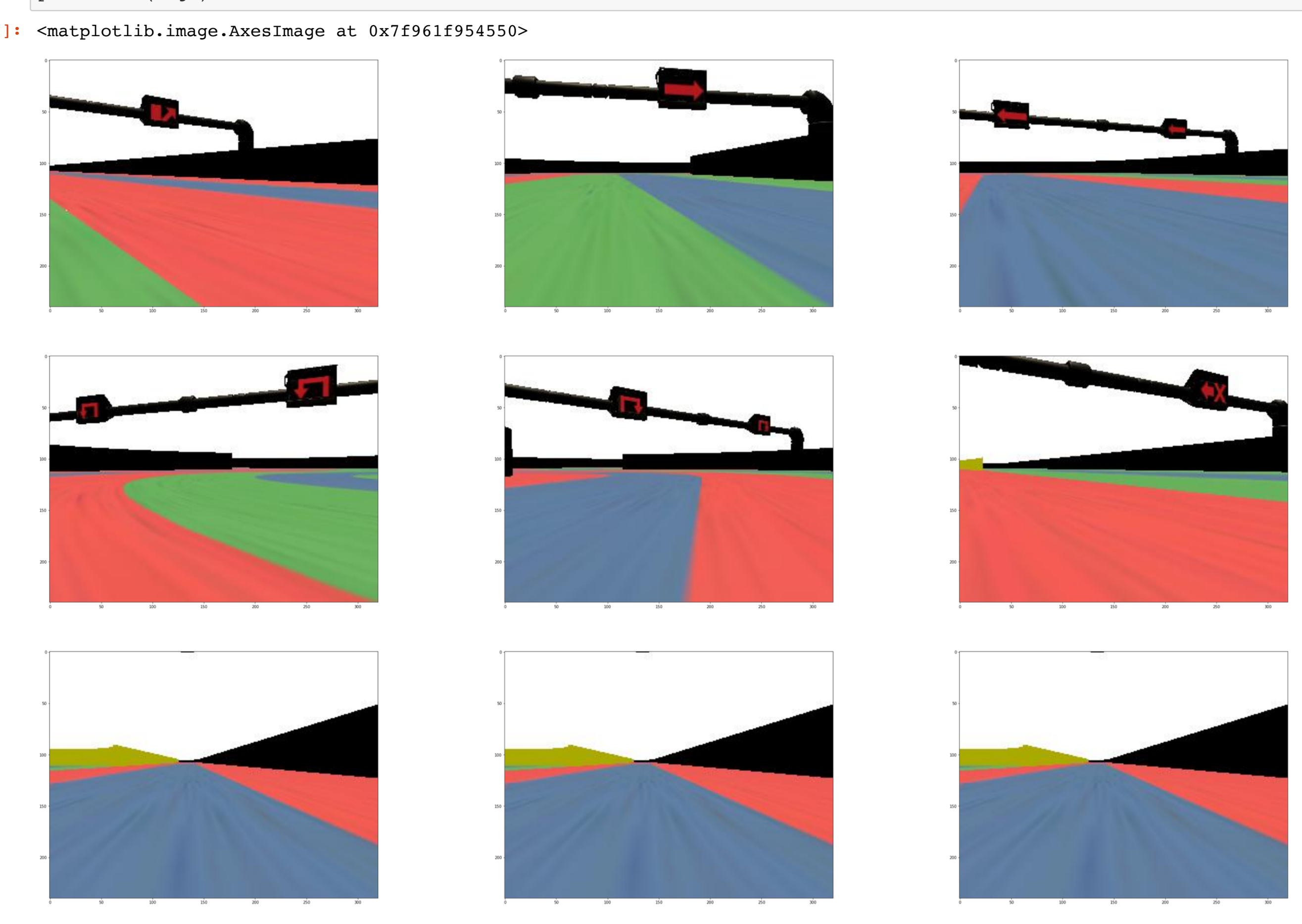
Date: 2020-08-07

In [58]:
```python
# -*- coding:utf-8 -*-
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.models import model_from_json
from CNNData import CCNNData,CCNNDataBase,g_input_shape,preprocess
import cv2
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import keras.backend as K
%matplotlib inline
from keras import optimizers
```

Data pre-process

In [59]:
```python
img1= cv2.imread('./samples/train/1/1-center_2018_08_22_09_29_24_368.jpg')
img2= cv2.imread('./samples/train/2/3-center_2018_08_23_11_04_17_054.jpg')
img3= cv2.imread('./samples/train/3/4-center_2018_08_22_15_06_12_233.jpg')
img4= cv2.imread('./samples/train/4/5-center_2018_08_23_10_41_43_916.jpg')
img5= cv2.imread('./samples/train/5/6-center_2018_08_21_18_23_11_991.jpg')
img6= cv2.imread('./samples/train/6/8-center_2018_08_21_18_16_19_502.jpg')
img0= cv2.imread('./samples/train/0/1-center_2018_08_22_14_54_39_335.jpg')
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)
img4 = cv2.cvtColor(img4, cv2.COLOR_BGR2RGB)
img5 = cv2.cvtColor(img5, cv2.COLOR_BGR2RGB)
img6 = cv2.cvtColor(img6, cv2.COLOR_BGR2RGB)
img0 = cv2.cvtColor(img0, cv2.COLOR_BGR2RGB)

fig,ax = plt.subplots(3,3,figsize=(60,40))
plt.subplot(3,3,1)
plt.imshow(img1)
plt.subplot(3,3,2)
plt.imshow(img2)
plt.subplot(3,3,3)
plt.imshow(img3)
plt.subplot(3,3,4)
plt.imshow(img4)
plt.subplot(3,3,5)
plt.imshow(img5)
plt.subplot(3,3,6)
plt.imshow(img6)
plt.subplot(3,3,7)
plt.imshow(img0)
plt.subplot(3,3,8)
plt.imshow(img0)
plt.subplot(3,3,9)
plt.imshow(img0)
```

Out[59]: <matplotlib.image.AxesImage at 0x7f961f954550>



In [60]:
```python
img1_1 = preprocess(img1)
img2_1 = preprocess(img2)
img3_1 = preprocess(img3)
img4_1 = preprocess(img4)
img5_1 = preprocess(img5)
img6_1 = preprocess(img6)
fig,ax = plt.subplots(2,3,figsize=(10,6))
plt.subplot(2,3,1)
plt.imshow(img1_1,cmap='gray')
plt.subplot(2,3,2)
plt.imshow(img2_1,cmap='gray')
plt.subplot(2,3,3)
plt.imshow(img3_1,cmap='gray')
plt.subplot(2,3,4)
plt.imshow(img4_1,cmap='gray')
plt.subplot(2,3,5)
plt.imshow(img5_1,cmap='gray')
plt.subplot(2,3,6)
plt.imshow(img6_1,cmap='gray')
```

Out[60]: <matplotlib.image.AxesImage at 0x7f960b18a4e0>

Load dataset

In [61]:
```python
g_test_train = CCNNData()
g_test_train.LoadTrain()
g_test_data = CCNNData()
g_test_data.LoadVerify()
print("Load finished")
```

```
adding train sample tag = .DS_Store folder = ./samples/Train/.DS_Store/
adding train sample tag = 0 folder = ./samples/Train/0/
adding train sample tag = 6 folder = ./samples/Train/6/
adding train sample tag = 1 folder = ./samples/Train/1/
adding train sample tag = 4 folder = ./samples/Train/4/
adding train sample tag = 3 folder = ./samples/Train/3/
adding train sample tag = 2 folder = ./samples/Train/2/
adding train sample tag = 5 folder = ./samples/Train/5/
adding verify sample tag = .DS_Store folder = ./samples/verify/.DS_Store/
adding verify sample tag = 0 folder = ./samples/verify/0/
adding verify sample tag = 6 folder = ./samples/verify/6/
adding verify sample tag = 1 folder = ./samples/verify/1/
adding verify sample tag = 4 folder = ./samples/verify/4/
adding verify sample tag = 3 folder = ./samples/verify/3/
adding verify sample tag = 2 folder = ./samples/verify/2/
adding verify sample tag = 5 folder = ./samples/verify/5/
Load finished
```

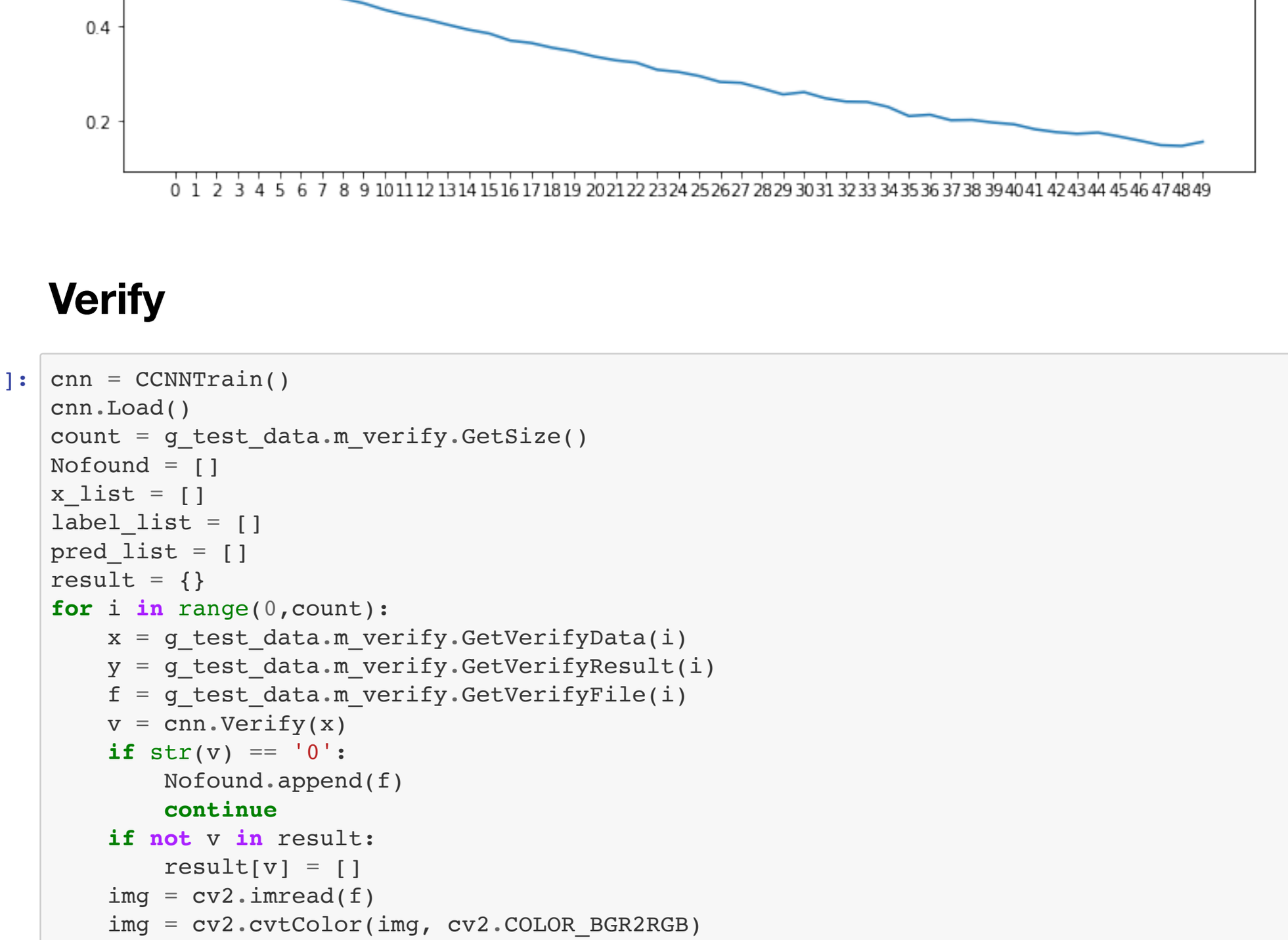
Begin Train

In [76]:
```python
class CCNNTrain:
    def __init__(self):
        self.m_model = Sequential()
        self.m_learn_rate = 0.0005 #0.5
        self.m_adam = optimizers.Adam(lr=self.m_learn_rate)

    def CreateModel(self,n_classes):
        self.m_model.add(Conv2D(28, (3, 3), activation='sigmoid', input_shape=g_input_shape))
        self.m_model.add(Conv2D(28, (3, 3), activation='sigmoid'))
        self.m_model.add(MaxPooling2D(pool_size=(2,2)))
        self.m_model.add(Dropout(0.25))
        self.m_model.add(Flatten())
        self.m_model.add(Dense(128, activation='sigmoid'))
        self.m_model.add(Dropout(0.5))
        self.m_model.add(Dense(n_classes, activation='softmax'))
        self.m_model.compile(loss='categorical_crossentropy',optimizer=self.m_adam,metrics=['accuracy'])

    def Fit(self,x_train,y_train):
        return self.m_model.fit(x_train, y_train, batch_size=32, epochs=50, verbose=1)

    def Evaluate(self,x_test,y_test):
        score = self.m_model.evaluate(x_test, y_test, verbose=1)
        return score

    def Predict(self, image):
        x = CCNNDataBase().GetPredict(image)
        if x is None :
            return 0
        predictions = self.m_model.predict(x)
        for pred in predictions.tolist():
            confidence = max(pred)
            if confidence < 0.5:
                return 0
            return pred.index(max(pred))

    def Verify(self, data):
        predictions = self.m_model.predict(data)
        for pred in predictions.tolist():
            confidence = max(pred)
            if confidence < 0.5:
                return 0
            return pred.index(max(pred))

    def Load(self):
        with open('./model/model.json', 'r') as json_file:
            loaded_model_json = json_file.read()
        self.m_model = model_from_json(loaded_model_json)
        self.m_model.load_weights("./model/model.h5")
        self.m_model.compile(loss='categorical_crossentropy',optimizer=self.m_adam,metrics=['accuracy'])
        return self.m_model

    def Save(self):
        model_json = self.m_model.to_json()
        with open("./model/model.json", "w") as json_file:
            json_file.write(model_json)
            # serialize weights to HDF5
        self.m_model.save_weights("./model/model.h5")
```

In [77]:
```python
n_train_class,y_train_data,x_train_data = g_test_train.GetTrain()
cnn = CCNNTrain()
cnn.CreateModel(n_train_class)
print("trainning")
measure = cnn.Fit(x_train_data,y_train_data)
cnn.Save()
```

```
trainning
Epoch 1/50
278/278 [==============================] - 9s 34ms/step - loss: 1.2089 - accuracy: 0.5425
Epoch 2/50
278/278 [==============================] - 9s 34ms/step - loss: 0.6283 - accuracy: 0.7681
Epoch 3/50
278/278 [==============================] - 9s 33ms/step - loss: 0.5715 - accuracy: 0.7833
Epoch 4/50
278/278 [==============================] - 10s 34ms/step - loss: 0.5415 - accuracy: 0.7906
Epoch 5/50
278/278 [==============================] - 10s 35ms/step - loss: 0.5221 - accuracy: 0.8003
Epoch 6/50
278/278 [==============================] - 9s 34ms/step - loss: 0.5026 - accuracy: 0.8109
Epoch 7/50
278/278 [==============================] - 9s 34ms/step - loss: 0.4868 - accuracy: 0.8139
Epoch 8/50
278/278 [==============================] - 9s 34ms/step - loss: 0.4719 - accuracy: 0.8234
Epoch 9/50
278/278 [==============================] - 10s 34ms/step - loss: 0.4597 - accuracy: 0.8243
Epoch 10/50
278/278 [==============================] - 9s 33ms/step - loss: 0.4495 - accuracy: 0.8309
Epoch 11/50
278/278 [==============================] - 9s 34ms/step - loss: 0.4355 - accuracy: 0.8335
Epoch 12/50
278/278 [==============================] - 9s 33ms/step - loss: 0.4243 - accuracy: 0.8404
Epoch 13/50
278/278 [==============================] - 10s 35ms/step - loss: 0.4152 - accuracy: 0.8408
Epoch 14/50
278/278 [==============================] - 9s 33ms/step - loss: 0.4040 - accuracy: 0.8440
Epoch 15/50
278/278 [==============================] - 9s 34ms/step - loss: 0.3933 - accuracy: 0.8500
Epoch 16/50
278/278 [==============================] - 9s 31ms/step - loss: 0.3852 - accuracy: 0.8556
Epoch 17/50
278/278 [==============================] - 8s 30ms/step - loss: 0.3701 - accuracy: 0.8616
Epoch 18/50
278/278 [==============================] - 10s 36ms/step - loss: 0.3650 - accuracy: 0.8626
Epoch 19/50
278/278 [==============================] - 10s 37ms/step - loss: 0.3549 - accuracy: 0.8682
Epoch 20/50
278/278 [==============================] - 10s 37ms/step - loss: 0.3475 - accuracy: 0.8701
Epoch 21/50
278/278 [==============================] - 10s 37ms/step - loss: 0.3365 - accuracy: 0.8763
Epoch 22/50
278/278 [==============================] - 10s 36ms/step - loss: 0.3285 - accuracy: 0.8775
Epoch 23/50
278/278 [==============================] - 9s 33ms/step - loss: 0.3236 - accuracy: 0.8770
Epoch 24/50
278/278 [==============================] - 9s 32ms/step - loss: 0.3083 - accuracy: 0.8842
Epoch 25/50
278/278 [==============================] - 9s 33ms/step - loss: 0.3038 - accuracy: 0.8844
Epoch 26/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2952 - accuracy: 0.8908
Epoch 27/50
278/278 [==============================] - 10s 34ms/step - loss: 0.2825 - accuracy: 0.8949
Epoch 28/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2806 - accuracy: 0.8952
Epoch 29/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2687 - accuracy: 0.8999
Epoch 30/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2560 - accuracy: 0.9066
Epoch 31/50
278/278 [==============================] - 10s 34ms/step - loss: 0.2610 - accuracy: 0.9037
Epoch 32/50
278/278 [==============================] - 10s 34ms/step - loss: 0.2481 - accuracy: 0.9099
Epoch 33/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2409 - accuracy: 0.9109
Epoch 34/50
278/278 [==============================] - 10s 34ms/step - loss: 0.2402 - accuracy: 0.9151
Epoch 35/50
278/278 [==============================] - 9s 34ms/step - loss: 0.2296 - accuracy: 0.9169
Epoch 36/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2102 - accuracy: 0.9242
Epoch 37/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2131 - accuracy: 0.9242
Epoch 38/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2014 - accuracy: 0.9318
Epoch 39/50
278/278 [==============================] - 10s 35ms/step - loss: 0.2021 - accuracy: 0.9274
Epoch 40/50
278/278 [==============================] - 9s 34ms/step - loss: 0.1965 - accuracy: 0.9313
Epoch 41/50
278/278 [==============================] - 10s 35ms/step - loss: 0.1927 - accuracy: 0.9335
Epoch 42/50
278/278 [==============================] - 9s 33ms/step - loss: 0.1823 - accuracy: 0.9349
Epoch 43/50
278/278 [==============================] - 10s 35ms/step - loss: 0.1763 - accuracy: 0.9406
Epoch 44/50
278/278 [==============================] - 10s 37ms/step - loss: 0.1727 - accuracy: 0.9403
Epoch 45/50
278/278 [==============================] - 9s 33ms/step - loss: 0.1752 - accuracy: 0.9389
Epoch 46/50
278/278 [==============================] - 9s 34ms/step - loss: 0.1670 - accuracy: 0.9373
Epoch 47/50
278/278 [==============================] - 10s 37ms/step - loss: 0.1581 - accuracy: 0.9438
Epoch 48/50
278/278 [==============================] - 10s 37ms/step - loss: 0.1485 - accuracy: 0.9480
Epoch 49/50
278/278 [==============================] - 10s 37ms/step - loss: 0.1469 - accuracy: 0.9496
Epoch 50/50
278/278 [==============================] - 10s 36ms/step - loss: 0.1558 - accuracy: 0.9474
```

In [78]:
```python
plt.style.use({'figure.figsize':(12, 6)})
df = pd.DataFrame()
df['loss'] = measure.history['loss']
df['accuracy'] = measure.history['accuracy']
#df['val_loss'] = measure.history['val_loss']
#df['val_accuracy'] = measure.history['val_accuracy']
xticks = range(len(df['accuracy']))
df.plot(xticks=xticks)
plt.ylabel('loss/accuracy')
plt.legend()
plt.show()
```



Verify

In [79]:
```python
cnn = CCNNTrain()
cnn.Load()
count = g_test_data.m_verify.GetSize()
Nofound = []
x_list = []
label_list = []
pred_list = []
result = {}
for i in range(0,count):
    x = g_test_data.m_verify.GetVerifyData(i)
    y = g_test_data.m_verify.GetVerifyResult(i)
    f = g_test_data.m_verify.GetVerifyFile(i)
    v = cnn.Verify(x)
    if str(v) == '0':
        Nofound.append(f)
        continue
    if not v in result:
        result[v] = []
    img = cv2.imread(f)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = preprocess(img)
    if len(img) > 0:
        result[v].append(img)
    label_list.append(int(y))
    pred_list.append(int(v))

label_list = np.array(label_list)
pred_list = np.array(pred_list)
```

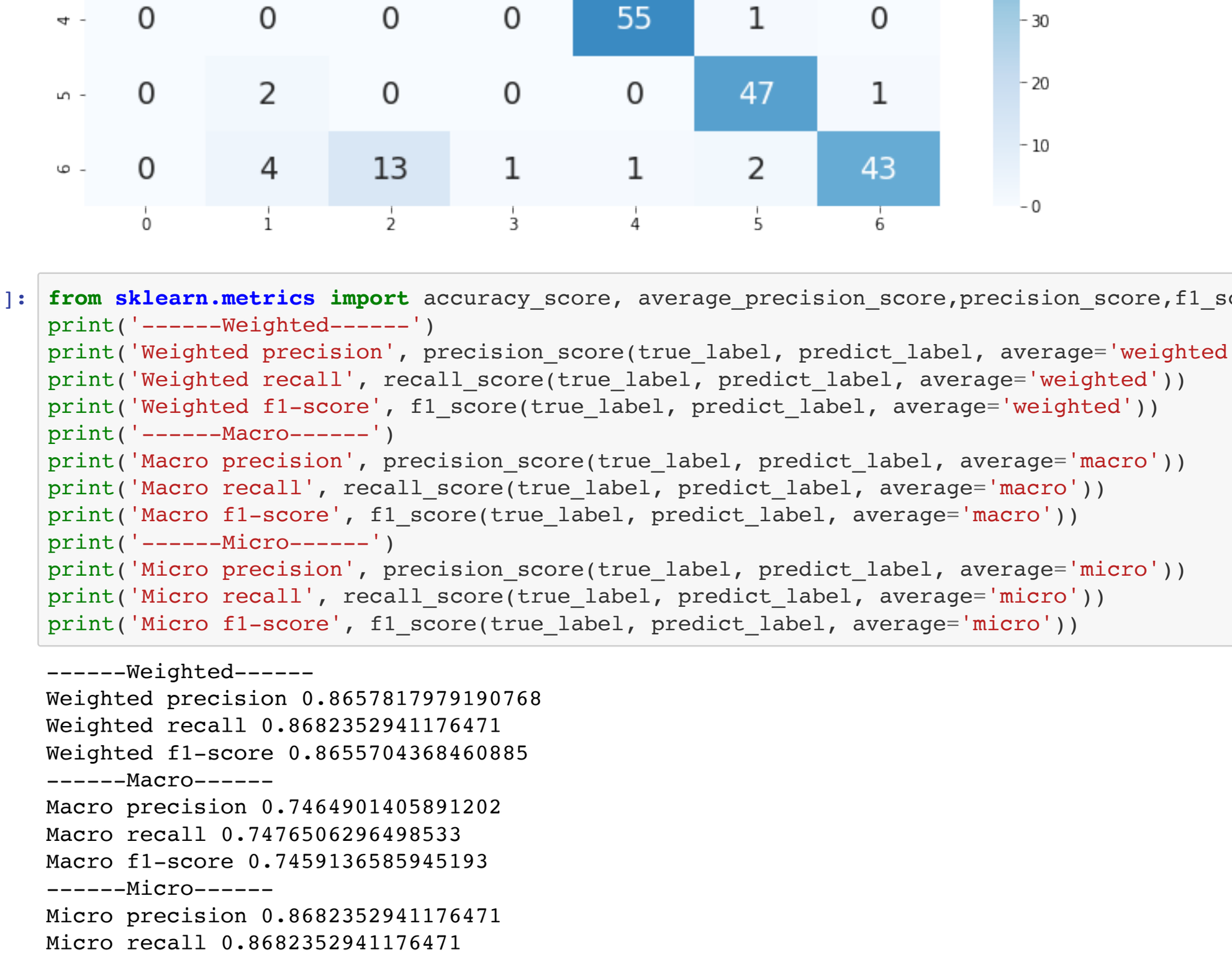
In [80]:
```python
true_label = label_list
predict_label = pred_list
pd.crosstab(true_label,predict_label,rownames=['label'],colnames=['predict'])
```

Out[80]:

predict label	1	2	3	4	5	6
0	0	2	0	0	0	0
1	83	0	2	0	2	5
2	1	84	1	0	0	5
3	0	3	57	1	1	8
4	0	0	0	55	1	0
5	2	0	0	0	47	1
6	4	13	1	1	2	43

In [87]:
```python
from sklearn import metrics

confusion_matrix = metrics.confusion_matrix(true_label, predict_label)
sns.heatmap(confusion_matrix, annot=True, annot_kws={"size": 19}, cmap="Blues")
```

Out[87]: <AxesSubplot:>



In [88]:
```python
from sklearn.metrics import accuracy_score, average_precision_score,precision_score,f1_score,recall_score
print('------Weighted------')
print('Weighted precision', precision_score(true_label, predict_label, average='weighted'))
print('Weighted recall', recall_score(true_label, predict_label, average='weighted'))
print('Weighted f1-score', f1_score(true_label, predict_label, average='weighted'))
print('------Macro------')
print('Macro precision', precision_score(true_label, predict_label, average='macro'))
print('Macro recall', recall_score(true_label, predict_label, average='macro'))
print('Macro f1-score', f1_score(true_label, predict_label, average='macro'))
print('------Micro------')
print('Micro precision', precision_score(true_label, predict_label, average='micro'))
print('Micro recall', recall_score(true_label, predict_label, average='micro'))
print('Micro f1-score', f1_score(true_label, predict_label, average='micro'))
```

```
------Weighted------
Weighted precision 0.8657817979190768
Weighted recall 0.8682352941176471
Weighted f1-score 0.8655704368460885
------Macro------
Macro precision 0.7464901405891202
Macro recall 0.7476506296498533
Macro f1-score 0.7459136585945193
------Micro------
Micro precision 0.8682352941176471
Micro recall 0.8682352941176471
Micro f1-score 0.8682352941176471
```

In [89]:
```python
def ShowImg(img_list):
    rows = 3
    if len(img_list) < 24:
        rows = len(img_list)//8
    fig,ax = plt.subplots(rows,8,figsize=(10,5))
    for j in range(0,rows):
        for i in range(1,9):
            plt.subplot(rows,8,i+j*8)
            plt.imshow(img_list[i+j*8],cmap='gray')
            plt.axis('off')
```

In [84]:
```python
ShowImg(result[3])
```

In [85]:
```python
def ShowLayer(data, num_layer=1):
    data = np.expand_dims(data, axis=0)
    layer = K.function([cnn.m_model.layers[0].input], [cnn.m_model.layers[num_layer].output])
    f1 = layer([data])[0]
    num = f1.shape[-1]
    plt.figure(figsize=(10, 10))
    for i in range(num):
        plt.subplot(8,8, i+1)
        plt.imshow(f1[0, :, :, i] * 255, cmap='gray')
        plt.axis('off')
    plt.show()
```

In [86]:
```python
ShowLayer(result[2][0],2)
```

In [ ]:

In [ ]: