

SI100B Project Face Detection and emotion classification

Lecture 6

Objectives

- Introduce the programming approach of the program written in the last class.
- Master the basic skills of plotting with Matplotlib.
- Integrate the entire process.

1 Review

Print the output data of the model first.

```
tensor([[ -5.8959,  -1.5088,   6.3008],
        [  8.7982,  -0.1693, -10.9480],
        [ -2.2721,   0.9975,   1.0200],
        [ -3.5406,  -1.7631,   4.7864],
        [ 23.0014, -13.4124, -11.7141],
        [ 11.5484,  -4.4288,  -8.6290],
        [  1.4547,   2.6202,  -5.7146],
        [  2.4613,  -4.1578,   1.5442],
        [  3.1281,  -0.1503,  -4.0557],
        [  5.1923,  -5.2262,   0.0678],
        [-10.8241,  -0.2851,   8.8197],
        [ -3.1854,   3.2595,  -2.1556],
        [ -9.0431,   1.7174,   6.0872],
        [ -3.9267,  -0.6808,   4.0678],
        [  0.4705,  -1.6097,   0.2503],
        [  5.5182,  -2.3706,  -3.9312],
        [ -1.6096,   0.5056,   0.4481],
        [ -6.0956,   2.4309,   2.4227],
        [  5.7047,  -0.0574,  -7.2277],
        [ -4.7507,   2.4462,   1.2095]])
```

Figure 1: Output

By analyzing the output data, we can see that our task is to find the index of the maximum element in each row. You can define a function to do this.

```
def FindPredLabel(array0):
    #find the index that have the max value in a two-dimension array
    pass
```

For confusion matrix, the first step is get three arrays. You can also define new functions to make program more concise.

1. Prediction. Traverse "new_label"
2. Label. Traverse "y"
3. TP(True positive). Traverse two array.

Confusion Matrix

Prediction \ True Label	Happy	Neutral	Sad	
Happy	TP[0]			Label[0]
Neutral		TP[1]		Label[1]
Sad			TP[2]	Label[2]
	Pred[0]	Pred[1]	Pred[2]	

Pred=[count(pred, 0),
count(pred, 1),
count(pred, 2)]

Label=[count(label, 0),
count(label, 1),
count(label, 2)]

TP=[Count2(label, pred, 0),
count2(label, pred, 1)
count2(label, pred, 2)]

Figure 2: Output

```
def count(array, i):
    #Count the number of elements in the array that have the value i.
    pass

def count2(array0, array1, i):
    #Count the number of positions where the elements at the same index in two arrays are both equal to i.
    pass
```

For the data in full dataset, you need to define a function to add two arrays.

```
def Add(array0, array1):
    pass

Add(TP_Full, TP)
```

After all this you can get the accuracy and recall.

$$acc_i = \frac{TP_full[i]}{Pred_full[i]}, recall_i = \frac{TP_full[i]}{Label_full[i]}$$

List Comprehension is an concise way to get an array. The basic grammar is:

```
[expression for item in iterable if condition ].
```

It is quite suitable to get "TP, Pred, Label" arrays by this way.

Another way to compute the confusion matrix is to directly record the matrix:

```
for i, j in zip(y, new_label):
    confusion_matrix[i, j] += 1
```

2 Matplotlib

It is very important to present your work results using chart visualizations. Today, we will learn the basic operations of Matplotlib and use this library to showcase the results you achieved earlier.

2.1 Plot

Plot a sine function curve using the plot function.

```
import matplotlib.pyplot as plt
import numpy as np

# Set the range of the independent variable to [-5, 5] when plotting. Sample 1,000 points
x = np.linspace(-5, 5, 1000)

# Get the value of the dependent variable at sampling points.
y0 = np.sin(x)

#set figure title
plt.title("example")
#plot curve
plt.plot(x, y0, label="sin", color = 'r')
#add curve labels
plt.legend()
#show figure
plt.show()
```

2.2 Confusion Matrix

Show a confusion matrix.

```
#Use the confusion matrix which you generated at the last class.
fig, ax = plt.subplots(figsize=(8, 6))

#Show confusion matrix
cax = ax.matshow(conf_matrix, cmap="Blues")

#set the class labels
ax.set_xticks(np.arange(num_classes))
ax.set_yticks(np.arange(num_classes))
ax.set_xticklabels([f'Class {i}' for i in range(num_classes)])
ax.set_yticklabels([f'Class {i}' for i in range(num_classes)])

#add text to graph
for i in range(num_classes):
    for j in range(num_classes):
        ax.text(j, i, str(conf_matrix[i, j]), ha='center', va='center', color='black')

#set axis name
ax.set_xlabel('Predicted Label')
```

```
ax.set_ylabel('True Label')
ax.set_title('Confusion Matrix')

plt.show()
```

2.3 Subfigure

If you want to display two plots side by side in one window, you can use the subplot feature.

```
# Set the range of the independent variable to [-5, 5] when plotting. Sample 1,000 points
x = np.linspace(-5, 5, 1000)

# There are two dependent variables.
y0 = np.sin(x)
y1 = np.cos(x)

# Create sub figure.
fig, ax = plt.subplots(2, 1)

# Add title
fig.suptitle("trigonometric functions")

ax[0].plot(x, y0, label = "sin")
ax[0].set_title("sin")
ax[0].legend() # Add label
ax[0].grid() # Grid on
ax[1].plot(x, y1, label = "cos")
ax[1].set_title("cos")
ax[1].legend() # Add label
ax[1].grid() # Grid on
plt.show()
```

Don't forget to include necessary annotations and titles when creating the plots.

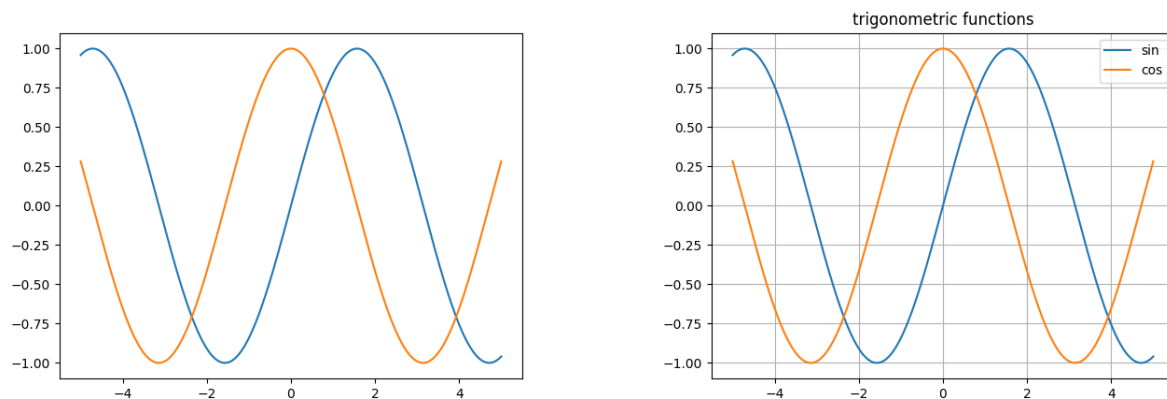


Figure 3: compare

Plot the loss curve and accuracy curve that generated in third class.



Figure 4: Output

Display the confusion matrix generated in the fourth class.

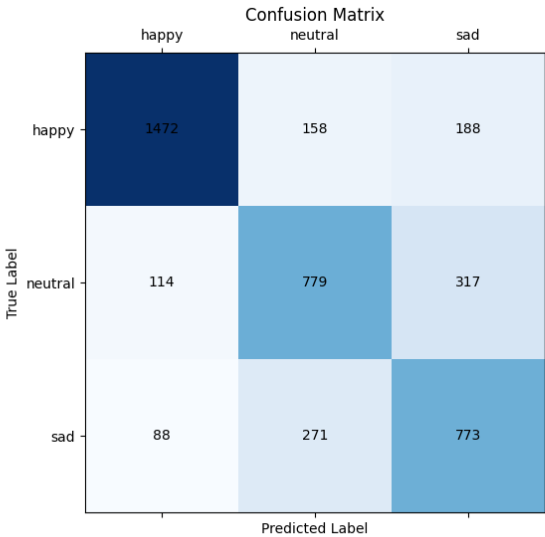


Figure 5: Output

Think about what this graph represents; do the two categories that are easy to confuse correspond with your recall and accuracy? Is it more intuitive now?

3 Integrate the process

Now that the conditions for connecting the entire process are ready, in this class, we will combine detection and classification to directly obtain the emotions of individuals in photos.

Please follow the prompts below to gradually complete your code and ultimately implement the function of detecting people's emotions.

```
import cv2
import torch
import torchvision.transforms as transforms

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

class Detector:
    def __init__(self, CascadePath, ModelPath):
        self.classes = ['happy', 'neutral', 'sad']
        #Step 1 load models
        # load a CascadeClassifier

        # load a DL model

    def process(self, img):
        #Step 2 process image
        #convert it to gray

        #detect use CascadeClassifier

        #Cropped faces from image and resize it to 48 * 48 and append it to a list

        #convert all cropped image to tensor

        #predict every tensor and draw the result to image

        return img

    def transform2tensor(self, data):
        tensor_data = transform(data)
        return tensor_data[None]

#Step 3 Create an instance of a class and use this class to process your image.
# read demo image

# init one detector

# call process

# write the ret to local and display it here
```

One more thing, we have been trying to downplay the data type during training. It is not the raw image data

that is directly read, but rather we preprocess the image through a series of normalization steps to obtain a tensor, which is then used for training and prediction. We have provided this method, and those interested can look into the purpose behind this approach and what role it plays in the model.

4 Bonus

1. A better approach would be to define the Detector class as a singleton. Understand what a singleton is, and then convert this class into a singleton.
2. Record a video clip and make sure there at least two faces in this video. Use your program to process this video and generate a new video that highlights all detected faces and labels their emotions.