Color Image Processing Homework 3

> Author: M11102122 陳煜翔

> Date: 2022/12/30

[TOC]

Intorduction

Use numpy API to create a CNN model. This CNN model have fixed kernel weight to extract feature, and use linear regression to predict the result number.

Coding Detail

Envionment

I use Python environment by version 3.8, and use cv2, numpy to achieve this task, and use matplotlib package to visualize the prediction result.

utils.py

In utils.py file, I create a simple CNN model with convolution operator and maxpooling, and use forward function to extract the feature map.

```
import numpy as np
import os
import cv2

conv1_weight = np.array([[[[-1, -1, 1],
                           [-1, 0, 1],
                           [-1, 1, 1]],

                          [[-1, -1, -1],
                           [-1, 0, 1],
                           [1, 1, 1]]]])

conv2_weight = np.array([[[[-1, -1, 1],
                           [-1, 2, -1],
                           [1, 1, 1]],

                          [[1, -1, 0],
                           [-1, -1, -1],
                           [1, -1, 1]]]])


class model():
    def __init__(self) -> None:
        self.conv_weight = conv1_weight
        self.conv_weight2 = [conv1_weight, conv2_weight]
```

```
    def forward(self, x) -> np.ndarray:
        x = self.convolution(x, kernel_size=3, in_channel=1,  # Conv1
                             out_channel=2, padding=1,
kernel_weight=self.conv_weight)

        x = self.maxpool(x, kernel_size=2)  # Maxpool1

        x = self.convolution(x, kernel_size=3, in_channel=2,  # Conv2
                             out_channel=2, padding=1,
kernel_weight=self.conv_weight2)

        x = self.maxpool(x, kernel_size=2)  # Maxpool2

        x = np.reshape(x, (x.shape[0], -1))  # Flatten

        x = np.append(x, np.expand_dims(
            np.array([1] * x.shape[0]), axis=1), axis=1)
        return x

    def convolution(self, input_image, kernel_size, in_channel, out_channel,
padding, kernel_weight) -> np.ndarray:

        convolution_kernel = kernel_weight
        for batch in range(input_image.shape[0] - 1):  # Prepare Filter Size
            convolution_kernel = np.concatenate(
                (convolution_kernel, kernel_weight), axis=0)

        feature_map = np.zeros(
            (tuple([input_image.shape[0]]) + tuple([out_channel]) +
input_image.shape[2:]))

        input_image = np.pad(input_image, ((0, 0), (0, 0), (padding, padding),
                             (padding, padding)), mode='constant',
constant_values=(0, 0))  # Paddind image

        if (in_channel == 1):  # For First Layer
            for w in range(feature_map.shape[2]):
                for h in range(feature_map.shape[3]):
                    feature_map[:, :, w, h] = np.sum(np.sum(
                        input_image[:, :, w:w+kernel_size, h:h+kernel_size] *
convolution_kernel, axis=2), axis=2)
            return feature_map
        else:  # For in_channel = 2
            for channel in range(out_channel):  # Channelwise convolution
                for w in range(feature_map.shape[2]):
                    for h in range(feature_map.shape[3]):
                        feature_map[:, channel, w, h] = np.sum(np.sum(np.sum(
                            input_image[:, :, w:w+kernel_size, h:h+kernel_size] *
convolution_kernel[channel], axis=2), axis=2), axis=1)

            return feature_map

    def maxpool(self, input_image, kernel_size, stride=2) -> np.ndarray:
```

```
        feature_map = np.zeros(  # Set feature map size
            (input_image.shape[:2] + tuple(int(shape / kernel_size) for shape in 
input_image.shape[2:])))

        for w in range(feature_map.shape[2]):
            for h in range(feature_map.shape[3]):
                feature_map[:, :, w, h] = np.max(np.max(  # Calculate maxpool
                    input_image[:, :, w*stride:w*stride+kernel_size, 
h*stride:h*stride+kernel_size], axis=2), axis=2)

        return feature_map
```

main.py

In the main.py file, I use natsort package to sort file name by number, and use utils.model to extract image feature.

After we got the feature map of image, we use linear regressiion to predict the result, and calculate for confustion matrix at the same time.

Finally we use matplot package to plot the result and plus confusion matrix in the smae plot to visualize the result.

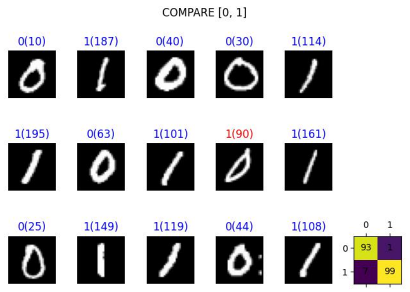
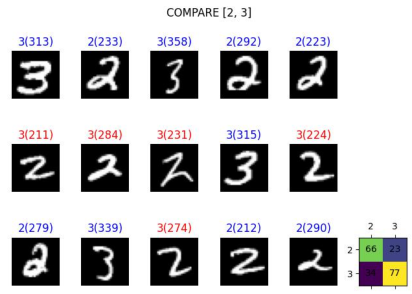
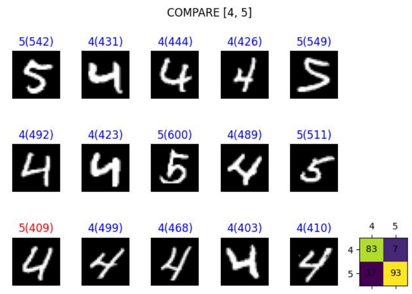
```
import config
from natsort import natsorted
import os
import utils
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

if __name__ == "__main__":

    BATCH = config.BATCH

    images = []
    labels = []

    for root, dirs, files in os.walk(config.IMAGE_PATH):
        files = natsorted(files)  # Sort by file name in the list
        for file in files:
            if (".png" in file):
                images.append(os.path.join(root, file))
                continue

            if (".txt" in file):
                file = open(os.path.join(root, file))
                while True:
                    line = file.readline()
```

```
                    if not line:
                        break
                    labels.append(int(line.strip()))
                file.close()

    for index in range(0, len(images), BATCH):
        model = utils.model()  # Read model
        label_list = []
        image_list = []

        for idx in range(BATCH):  # Prepare Data
            image = cv2.imread(images[index + idx], cv2.IMREAD_GRAYSCALE)
            image_list.append(image)  # Save original image
            # Resize image to 8x8
            image = cv2.resize(image, (8, 8), interpolation=cv2.INTER_CUBIC)
            # Expand dim to fit model
            image = np.expand_dims(np.expand_dims(image, axis=0), axis=0)
            # Prepare label list
            label = np.expand_dims(
                np.mod(np.array(labels[index + idx]), 2), axis=0)

            label_list.append(labels[index + idx])
            if idx == 0:  # First Iteration
                input_image = image
                input_label = label

            else:
                input_image = np.concatenate(
                    (input_image, image), axis=0)  # Concate image for batch
                input_label = np.concatenate((input_label, label), axis=0)
        label_list = list(set(label_list))

        prediction = model.forward(input_image)  # Get feature map

        # Transpose shape to (9, BATCH) for linear regression
        trans_prediction = np.transpose(prediction, (1, 0))
        # Prepare for linear regression
        gt_label = np.expand_dims(np.array(input_label), axis=1)

        linear_gression_function = np.dot(np.linalg.inv(  # Linear regression
            np.dot(trans_prediction, prediction)), np.dot(trans_prediction,
gt_label))

        confusion_matrix = np.zeros((2, 2))  # Set confusion matrix size
        prediction_list = []
        for idx in range(BATCH):
            prediction_number = np.where(  # Check prediction and gt is equal or
not
                np.sum(np.squeeze(prediction[idx]) *
np.squeeze(linear_gression_function)) > 0.5, 1, 0)
            prediction_list.append(prediction_number)  # Save prediction value
            confusion_matrix[int(np.mod(gt_label[idx][0], 2)),  # Confusion matrix
value plus 1
                             int(np.mod(prediction_number, 2))] += 1
```

```
        print("Confusion Matrix {label}\n{fm}\n".format(
            label=label_list, fm=confusion_matrix))

        random_list = random.sample(range(200), 15)  # Randon extract data
        # Set figure size for 15 figures and confusion matrix
        fig, ax = plt.subplots(3, 6)

        fig.suptitle("COMPARE {LABEL}".format(LABEL=label_list))

        for row in range(3):
            for column in range(5):
                idx = 5 * row + column
                color = "blue" if prediction_list[random_list[idx]  # If
predicriont == gt label: set blue color, o.w. set red color
                                                  ] == gt_label[random_list[idx]]
[0] else "red"
                ax[row, column].imshow(  # Set image for grayscale
                    np.squeeze(image_list[random_list[idx]]), cmap="gray")

                ax[row, column].set_title("{label}
({idx})".format(label=label_list[0] + input_label[random_list[idx]],  # Set title
for image number and index
                                                                  idx=index +
random_list[idx] + 1), color=color)

                ax[row, column].axis("off")

        ax[0, 5].axis("off")  # Close axis for right hand side
        ax[1, 5].axis("off")

        ax[2, 5].matshow(confusion_matrix)

        ax[2, 5].set_xticks(  # Set confusioon matrix label
            np.arange(len(label_list)), labels=label_list)
        ax[2, 5].set_yticks(
            np.arange(len(label_list)), labels=label_list)

        for i in range(len(label_list)):
            # Set confusion matrix with value of count for prediction
            for j in range(len(label_list)):
                ax[2, 5].text(i, j, str(int(confusion_matrix[i, j])),
                              va="center", ha="center")

        fig.tight_layout() # Let figure not too tight

        plt.savefig(os.path.join(config.SAVE_PATH, # Save result figure
                    str(int(index/BATCH)) + ".jpg"))
        plt.clf()
```

Result

In the result for this task, I choose 5 pairs to prediction in the same model and use linear regression to prediction the value.

COMPARE 0, 1	COMPARE 2, 3	COMPARE 4, 5
		

COMPARE 6, 7	COMPARE 8, 9