

report

by Tao Chen

FILE	25396_TAO_CHEN_REPORT_316598_430111777.DOCX (1.92M)		
TIME SUBMITTED	18-JAN-2021 10:36PM (UTC+0000)	WORD COUNT	1533
SUBMISSION ID	142619581	CHARACTER COUNT	7747

Report

Module Applied Machine Learning

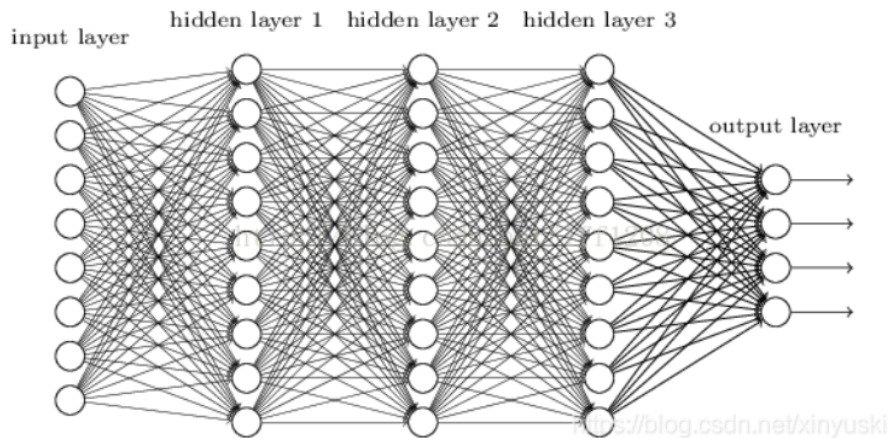
Lecturer Gordon Dickers

Student ID NO. 2006607

Student Name TAO CHEN

1. Introduction

Let me talk about DNN. Structurally, it is no different from the traditional NN (Neural Network). The main difference between DNN and NN is that the sigmoid function is replaced with ReLU and maxout, which overcomes the problem of gradient disappearance.



We see that in the structure of the fully connected DNN, the lower layer neurons and all upper layer neurons can form connections, which leads to the expansion of the number of parameters. This is not only easy to overfit, but also very easy to fall into the local optimum.

Because there are inherent local patterns in the image (such as the eyes, nose, mouth, etc.) in the face, the combination of image processing and neural network leads to the convolutional neural network CNN. CNN links the upper and lower layers through the convolution kernel. The same convolution kernel is shared in all images. The image still retains the original positional relationship after the convolution operation.

2. Experiment

This part will describe the experimental process in detail to realize the facial expression recognition function.

2.1 Examine the data

After examining the data, it is easy to find that the number of images in each class is

about 36000, their format obviously is greyscale, and image size is 48*48. This data has been divided into a training set and a test set. The test set accounts for about 20% of the total number of data sets, and the training set accounts for about 80% of the total number of data sets.

2.2 Network model selection

In order to realize the facial expression recognition function, after comparing DNN and CNN, choose CNN to realize this function. This report uses two CNN network models and compares their performance.

2.2.1 AlexNet model^[1]

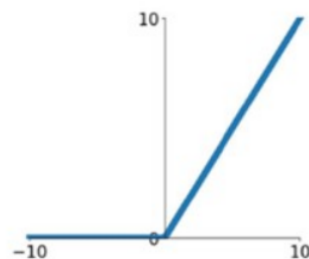
The first typical CNN is the LeNet5 network structure, but the first network that caught everyone's attention was AlexNet.

AlexNet deepens the structure of the network on the basis of LeNet, and learns richer and higher-dimensional image features. Features of AlexNet:

- Deeper network structure.
- Use stacked convolutional layers, that is, convolutional layer + convolutional layer + pooling layer to extract image features.
- Use Dropout to suppress overfitting.
- Use Data Augmentation to Suppress Overfitting.
- Use Relu function to replace the previous sigmoid as the activation function.
- Multi-GPU training.

Advantages of using relu function as an activation function in Alexnet:

$$\text{ReLU} \\ \max(0, x)$$

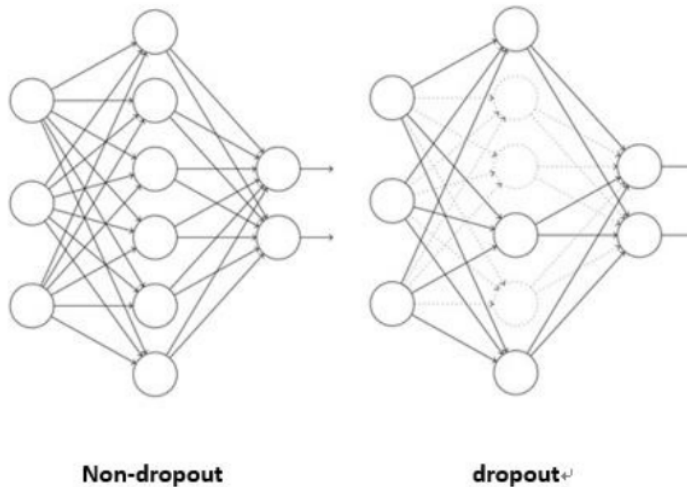


- sigmoid and tanh function have a saturation region, the derivative of the relu function is always 1 when $x > 0$, because the continuous multiplication expression of the gradient includes the derivative of the activation function of each layer and the weight of each layer, relu function solves the derivative problem of the activation function, so there is It helps alleviate the disappearance of the gradient, and can also solve the gradient explosion to a certain extent, thereby speeding up the training speed.
- Whether it is forward propagation or backward propagation, the amount of calculation is significantly less than sigmoid and tanh.

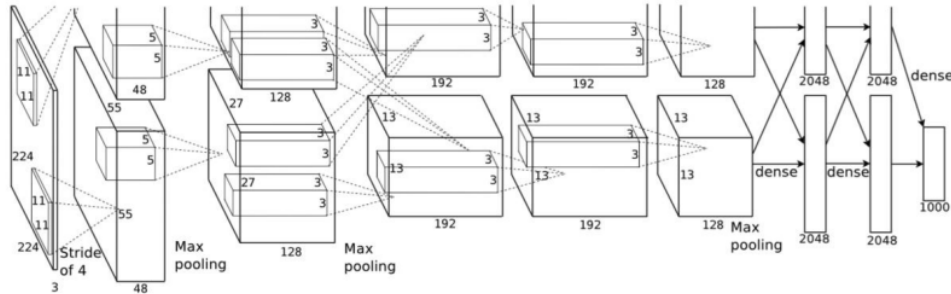
Dropout

This is a more commonly used method to suppress overfitting.

The main reason for introducing Dropout is to prevent overfitting. In the neural network, Dropout is realized by modifying the structure of the neural network itself. For a neuron of a certain layer, the neuron is set to 0 by a defined probability, and this neuron does not participate in forward and backward propagation, just like in the network is deleted, while keeping the number of neurons in the input layer and output layer unchanged, and then update the parameters according to the neural network learning method. In the next iteration, some neurons are randomly deleted again (set to 0) until the end of training.



AlexNet network structure



The network contains 8 layers with weights; the first 5 layers are convolutional layers, and the remaining 3 layers are fully connected layers. The output of the last fully connected layer is the input of 7-dimensional softmax, and softmax will generate a distribution of 7 types of labels.

- **Convolutional layer C1**

There are 96 convolution kernels of $11 * 11$; padding='same'; stride=2; activation is relu; For maxpooling, the pooling core size is $3 * 3$, and the stride is 2;

- **Convolutional layer C2**

There are 256 convolution kernels of $5 * 5$; padding='same'; stride=1; activation is relu; For maxpooling, the pooling core size is $3 * 3$, and the stride is 2;

- **Convolutional layer C3**

There are 384 convolution kernels of $3 * 3$; padding='same'; stride=1; activation is relu;

- **Convolutional layer C4**

Almost exactly the same as the convolutional layer C3

- **Convolutional layer C5**

There are 256 convolution kernels of $3 * 3$; padding='same'; stride=1; activation is relu; For maxpooling, the pooling core size is $2 * 2$, and the stride is 2;

- **Fully connected layer FC6**

A total of 2048 neurons, the activation function is Relu, and a dropout of 0.5 is performed.

- **Fully connected layer FC7**

Almost exactly the same as the fully connected layer FC7

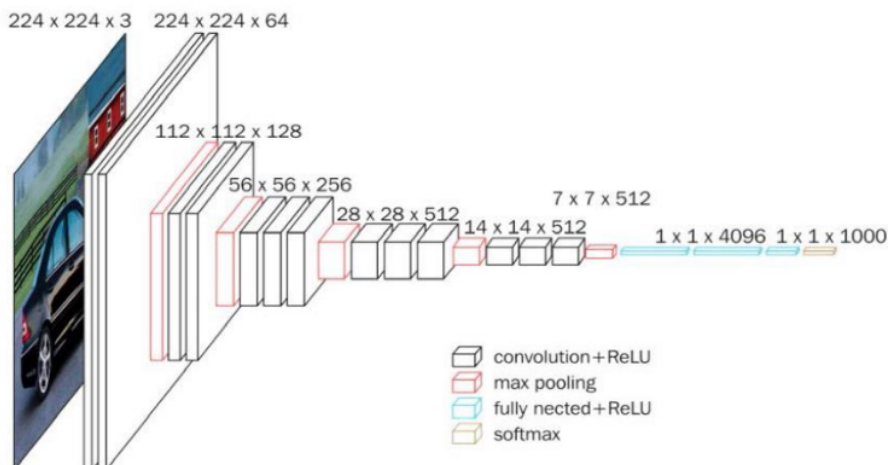
- **Output layer**

A total of 7 neurons, 7 classifications

2.2.2 VGG16 model^[2]

The biggest improvement of the VGG16 network is in the depth of the network, from the 8 layers of AlexNet to the 16 layers. A deeper network means stronger expression ability, which benefits from the support of strong computing power. Another notable feature of VGG16 is that it only uses a single size of 3×3 convolution kernel. In fact, 3×3 small convolution kernels are widely used in many convolutional networks. This is due to the same receptive field. In this case, the stacking effect of the small convolution kernel is better than that of the large convolution kernel, and the amount of parameters is also less. VGG16 uses a 3×3 convolution kernel to replace the large convolution kernel (11×11 , 5×5) in AlexNet, and has achieved better results.

VGG16 network structure



- **Convolutional layer C1**

Two convolutions, each using 64 3×3 convolution kernels; padding='same'; stride=1; activation is relu; For maxpooling, the pooling core size is 2×2 , and the stride is 2;

- **Convolutional layer C2**

Two convolutions, each using 128 3×3 convolution kernels, the others are the same as C1

- **Convolutional layer C3**

Three convolutions, each using 256 3×3 convolution kernels, the others are the same as C1

- **Convolutional layer C4, C5**

Three convolutions, each using 512 3×3 convolution kernels, the others are the same as C1

- **Fully connected layer FC6**

A total of 4096 neurons, the activation function is Relu, and a dropout of 0.5 is performed.

- **Fully connected layer FC7**

A total of 512 neurons, the activation function is Relu, and a dropout of 0.5 is performed.

- **Output layer**

A total of 7 neurons, 7 classifications

2.3 initial choice of parameters/hyper-parameters.

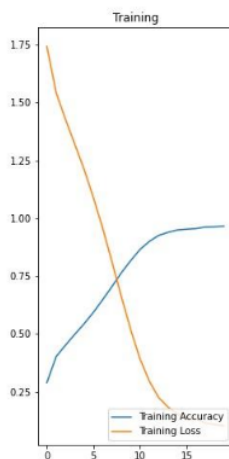
- **How many categories to recognize** class_num = 7
- **Image size** image_size = 48

- **Image channels** img_channels = 1
- **Learning rate** learning_rate = 0.00001
Excessive learning rate can easily lead to divergence of the model, so choose a smaller learning rate
- **Batch size** batch_size = 4
When the data cannot be passed through the neural network at one time, the data set needs to be divided into several batches
- **Dropout rate** dropout_rate = 0.5
When the dropout rate is 0.5, the generalization ability of the model is the best, but it can be adjusted appropriately according to actual needs.
- **The number of epoch** epoch_num = 20
As the number of epochs increases, the number of weight updates in the neural network also increases, and the curve changes from underfitting to overfitting. So, how many epochs are appropriate? Unfortunately, there is no correct answer to this question. For different data sets, the answer is different.

2.3 The evaluation of your experiments

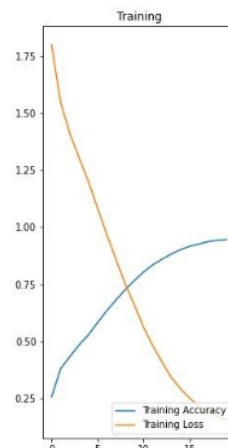
AlexNet

Loss and accuracy



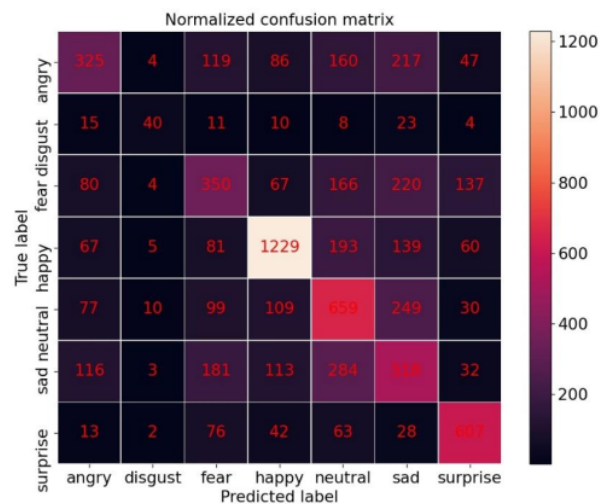
VGGNet

Loss and accuracy

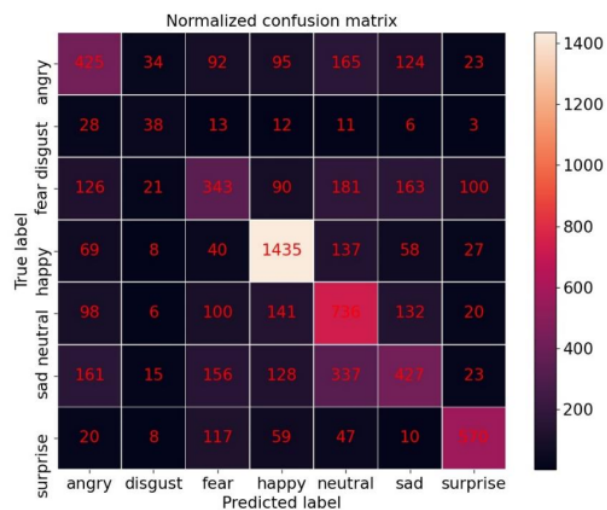


By comparing the two figures, it can be concluded that the accuracy of the AlexNet model is higher after training for 20 epochs. Compared with VGG16, the loss function value is also reduced faster. Therefore, the AlexNet model converges faster than the VGG16 model.

Confusion matrix of AlexNet



Confusion matrix of VGG16



It can be seen from the confusion matrix that the two model pairs are easier to identify the happy category

AlexNet

	angry	disgust	fear	happy	neutral	sad	surprise
precision	0.47	0.59	0.38	0.74	0.43	0.37	0.66
recall	0.34	0.36	0.34	0.69	0.53	0.42	0.73
f1-score	0.39	0.45	0.36	0.72	0.48	0.39	0.69
support	958.0	111.0	1024.0	1774.0	1233.0	1247.0	831.0

VGG16

	angry	disgust	fear	happy	neutral	sad	surprise
precision	0.46	0.29	0.4	0.73	0.46	0.46	0.74
recall	0.44	0.34	0.33	0.81	0.6	0.34	0.69
f1-score	0.45	0.32	0.36	0.77	0.52	0.39	0.71
support	958.0	111.0	1024.0	1774.0	1233.0	1247.0	831.0

Test :

```
C:\Users\Administrator\anaconda3\envs\tensorflow_gpuenv\python.exe C:/Users/Administrator/Pychar
2021-01-18 20:05:17.227705: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
3
```

The result is 3, so this is a picture of happy classification

Comment

Through the above chart, we can know that AlexNet is better than VGGNet under most performance indicators

Concussion^[4]

Compared with large networks (such as VGG), the AlexNet deep convolutional neural network has low complexity and fewer parameters; compared with the existing lightweight networks, the training is not very difficult, and the representation ability is strong. It saves a lot of model training time, makes fast predictions, and is less prone to overfitting. Therefore, it is more suitable for real-time face recognition applications. At present, most of the novel networks are based on classic networks such as AlexNet, VGG, and GoogleNet, ^[3] and their structures are more complex than AlexNet. Therefore, you can choose AlexNet to realize facial expression recognition

Reference

- [1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In NIPS, 2012. ↵
- [2] K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition In ICLR, 2015. ↵
- [3] Szegedy C, Liu W, Jia Y, et al. Going Deeper with Convolutions. In CVPR, 2015.↵
- [4] Kaiming He, Xiangyu Zhang, Shaoqing Ren. Deep Residual Learning for Image Recognition. In↵

Appendix

Load dataset:

```
import tensorflow as tf
import numpy as np
import time
import random
import pickle
import math
import datetime

import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import ImageEnhance
from tqdm import tqdm
import itertools
from tensorflow.keras.utils import to_categorical

###获取所有图片的路径
def get_img_label_paths(file_name, classes):
    res = []
    for index, name in enumerate(classes): # enumerate()可以同时获得索引和元素
        class_path = file_name + classes[index] + '/'
        for img_name in os.listdir(class_path): # os.listdir 返回指定目录下所有的文件和目录
            img_path = class_path + img_name # 每一个图片的地址
            res.append((img_path, index))
    return res

# #####归一化
def get_image_array(img):
    return np.float32(img) / 127.5 - 1

def image_segmentation_generator(images_path, batch_size, classes):
    img_seg_pairs = get_img_label_paths(images_path, classes)
    random.shuffle(img_seg_pairs)
    zipped = itertools.cycle(img_seg_pairs)
    while True:
        x = []
        y = []
        for _ in range(batch_size):
            img, lab = next(zipped)
            # print(img, lab)
```

```
img = mpimg.imread(img)
img.resize(48,48,1) #裁剪为模型能输入的大小
img2 = tf.image.resize(img, [360, 360]) # 裁剪为模型能输入的大小
npdata = img.numpy()
X.append(get_image_array(img))

encoded = to_categorical(lab, 7)

Y.append(encoded)

yield np.array(X), np.array(Y)
```

Train:

```
import tensorflow as tf
print(tf.__version__)
print(tf.__version__)
print('GPU', tf.test.is_gpu_available())

import numpy as np

from tensorflow.keras import layers, optimizers, datasets, Sequential, metrics, models, regularizers, Input, Model
from tensorflow.keras.layers import Conv2D, LayerNormalization, MaxPooling2D

from tensorflow.keras.layers import Dropout, Flatten, Dense
from load_dataset import image_segmentation_generator
import matplotlib.pyplot as plt
```

```
class_num = 7
image_size = 48
img_channels = 1
learning_rate = 0.00001

batch_size = 4

dropout_rate = 0.5

epoch_num = 20
# 预先定义的变量

classes = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
```

```

file_name = "./dataset/train/"

def VGG16(nb_classes, input_shape):
    input_tensor = Input(shape=input_shape)
    # 1st block
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='conv1a')(input_tensor)
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='conv1b')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='pool1')(x)
    # 2nd block
    x = Conv2D(128, (3, 3), activation='relu', padding='same', name='conv2a')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same', name='conv2b')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='pool2')(x)
    # 3rd block
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='conv3a')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='conv3b')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='conv3c')(x)

    x = MaxPooling2D((2, 2), strides=(2, 2), name='pool3')(x)
    # 4th block
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='conv4a')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='conv4b')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='conv4c')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='pool4')(x)
    # 5th block
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='conv5a')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='conv5b')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='conv5c')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='pool5')(x)
    # full connection
    x = Flatten()(x)
    x = Dense(4096, activation='relu', name='fc6')(x)
    x = Dropout(0.5)(x)

    x = Dense(512, activation='relu', name='fc7')(x)
    x = Dropout(0.5)(x)
    output_tensor = Dense(nb_classes, activation='softmax', name='fc8')(x)

    model = Model(input_tensor, output_tensor)
    return model

def AlexNet(in_shape):
    model = Sequential(name='AlexNet')

    model.add(layers.Conv2D(96, (11, 11), strides=(2, 2), input_shape=(in_shape[1], in_shape[2], in_shape[3]),

```

```
padding='same',activation='relu',kernel_initializer='uniform'))
model.add(layers.MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(layers.Conv2D(256,(5,5),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(layers.MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(layers.Conv2D(384,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(layers.Conv2D(384,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(layers.Conv2D(256,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(layers.MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(2048,activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2048,activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(7,activation='softmax'))
```

```
return model

class BasicBlock(layers.Layer):
    def __init__(self,filter_num,stride=1):
        super(BasicBlock, self).__init__()
        self.conv1=layers.Conv2D(filter_num,(3,3),strides=stride,padding='same')
        self.bn1=layers.BatchNormalization()
        self.relu=layers.Activation('relu')

        self.conv2=layers.Conv2D(filter_num,(3,3),strides=1,padding='same')
        self.bn2 = layers.BatchNormalization()

        if stride!=1:
            self.downsample=Sequential()
            self.downsample.add(layers.Conv2D(filter_num,(1,1),strides=stride))
        else:
            self.downsample=lambda x:x
```

```
def call(self,input,training=None):
    out=self.conv1(input)
    out=self.bn1(out)
    out=self.relu(out)

    out=self.conv2(out)
    out=self.bn2(out)

    identity=self.downsample(input)
    output=layers.add([out,identity])
    output=tf.nn.relu(output)
    return output
```

```
# there are [b,512,h,w]
# 自适应
self.avgpool=layers.GlobalAveragePooling2D()
self.fc=layers.Dense(500)
self.fc1=Dense(num_classes, activation='softmax', name='fc8')

def call(self, input, training=None):
    x=self.stem(input)
    x=self.layer1(x)
    x=self.layer2(x)
    x=self.layer3(x)
    x=self.layer4(x)
    # [b,c]
    x=self.avgpool(x)
```

```
class ResNet(tf.keras.Model):
    def __init__(self, layer_dims, num_classes=7):
        super(ResNet, self).__init__()
        # 预处理层
        self.stem=Sequential([
            layers.Conv2D(64, (3,3), strides=(1,1)),
            layers.BatchNormalization(),
            layers.Activation('relu'),
            layers.MaxPool2D(pool_size=(2,2), strides=(1,1), padding='same')
        ])
        # resblock
        self.layer1=self.build_resblock(64, layer_dims[0])
        self.layer2 = self.build_resblock(128, layer_dims[1], stride=2)
        self.layer3 = self.build_resblock(256, layer_dims[2], stride=2)
        self.layer4 = self.build_resblock(512, layer_dims[3], stride=2)
```

```
        x=self.fc(x)
        x=self.fc1(x)
        return x

    def build_resblock(self, filter_num, blocks, stride=1):
        res_blocks= Sequential()
        # may down sample
        res_blocks.add(BasicBlock(filter_num, stride))
        # just down sample one time
        for pre in range(1, blocks):
            res_blocks.add(BasicBlock(filter_num, stride=1))
        return res_blocks

def resnet18():
    return ResNet([2, 2, 2, 2])
```

```
#model = AlexNet([None, 48, 48, 1])
#model = VGG16(7, (48, 48, 1))
model=resnet18()
model.build(input_shape=(None, 48, 48, 1))

model.summary()

def scheduler(epoch):
    if epoch < epoch_num * 0.4:
        return learning_rate
    if epoch < epoch_num * 0.8:
```

```
        return learning_rate * 0.1
    return learning_rate * 0.01

class DisplayCallback(tf.keras.callbacks.Callback):

    def on_epoch_end(self, Epoch, logs=None):

        global learning_rate
        global epoch_num
        if Epoch < epoch_num * 0.4:
            return learning_rate
        if Epoch < epoch_num * 0.8:
            return learning_rate * 0.1
        return learning_rate * 0.01
```

```
        # print("\n")
        print("learning_rate:" + str(learning_rate))

# print("\n")
# train

# train_gen = image_segmentation_generator(
#     save_dir, batch_size, classes)
# for X,Y in train_gen:
#     print(np.shape(X))
# break
```

```
if __name__ == "__main__":  
    # change_lr = LearningRateScheduler(scheduler)  
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate),  
                  loss='categorical_crossentropy', metrics=['accuracy'])  
    train_gen = image_segmentation_generator(  
        file_name, batch_size, classes)  
  
    history = model.fit_generator(train_gen,  
                                steps_per_epoch=35000 / batch_size,  
                                epochs=epoch_num,  
                                callbacks=[DisplayCallback()])  
  
    model.save_weights("RESNET.h5")  
    # model.save("RESNET.h5")  
    acc = history.history['accuracy']
```

```
loss = history.history['loss']  
  
epochs_range = range(epoch_num)  
  
plt.figure(figsize=(8, 8))  
plt.subplot(1, 2, 1)  
plt.plot(epochs_range, acc, label='Training Accuracy')  
plt.plot(epochs_range, loss, label='Training Loss')  
plt.legend(loc='lower right')  
plt.title('Training')  
plt.savefig("RESNET_acc_loss.jpg")  
plt.show()
```

Prediction:

```
import tensorflow as tf
import numpy as np
import os
from PIL import Image
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
#model=tf.keras.models.load_model('VGG16.h5')

#model.load_weights('RESNET.h5')
model=tf.keras.models.load_model('AlexNET.h5')

classes = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

file_name="./dataset/test/"
```

```
def get_test_img_label_paths(file_name, classes):

    test_dataset=[]
    for index, name in enumerate(classes): # enumerate()可以同时获得索引和元素
        class_path = file_name + classes[index] + '/'
        for img_name in os.listdir(class_path): # os.listdir 返回指定目录下所有的文件和目录

            img_path = class_path + img_name # 每一个图片的地址

            test_dataset.append((img_path, index))

    return test_dataset

test_dataset=get_test_img_label_paths(file_name, classes)
```

```
def pred(img, model):
    img = Image.open(img) #加载图片
    # img = img.crop((80, 0, 80+480, 480))
    sample_image = np.array(img) #图片转np
    sample_image.resize([48, 48, 1])
    input_image = tf.cast(sample_image, tf.float32) / 127.5 - 1 #标准化
    img=(np.expand_dims(input_image, 0)) # 扩充一个维度
    prob=model.predict(img) #用模型进行预测
    pred_mask = tf.argmax(prob, axis=-1) #取最大值
    pred_mask1 = pred_mask[..., tf.newaxis] #缩小一个维度
    pred_ = np.array(pred_mask1)
    pred_=np.squeeze(pred_)
    return int(pred_)

print(pred(test_dataset[0][0], model))
```

```
def get_pred_true(test_dataset,model):
    pred_value=[]
    true_value=[]
    true_=None
    for i in range(len(test_dataset)):
        pred_=pred(test_dataset[i][0],model)
        pred_value.append(pred_)
        true_=test_dataset[i][1]
        true_value.append(true_)
    return pred_value,true_value

pred_value,true_value=get_pred_true(test_dataset,model)

def crate_confusion_matrix(y_true,y_pred,labels,save_dir="confusion_matrix.jpg"):
    title='Normalized confusion matrix'
```

```
cm = confusion_matrix(y_true, y_pred)
#fig=plt.figure(figsize=(10,8))
plt.figure(figsize=(10, 8), dpi=120)
h=sns.heatmap(cm,linewidths=0.5,cbar=False) #设置不使用其默认自带的colorbar
cb=h.figure.colorbar(h.collections[0]) #显示colorbar
cb.ax.tick_params(labelsize=16) #设置colorbar刻度字体大小。

ind_array = np.arange(len(labels))
x, y = np.meshgrid(ind_array, ind_array)
for x_val, y_val in zip(x.flatten(), y.flatten()):
    c = cm[y_val][x_val]
    if c > 0:
        plt.text(x_val+0.5, y_val+0.5, "%d" % (c,), color='red', fontsize=16, va='center', ha='center')
# offset the tick
```

```
plt.title(title,fontsize=16)
xlocations = np.array(range(len(labels)))
plt.xticks(xlocations+0.5,labels,fontsize=16)
plt.yticks(xlocations+0.5, labels,fontsize=16)
plt.ylabel('True label',fontsize=16)
plt.xlabel('Predicted label',fontsize=16)
plt.savefig(save_dir, format='jpg')
plt.show()

crate_confusion_matrix(true_value,pred_value,classes)

dict1 = {}
for key in true_value:
    dict1[key] = dict1.get(key, 0) + 1
print(dict1)
```

```
sppuert.append(dict1[key])
val=[]

val.append([precision,recall,f1,sppuert])
B = np.squeeze(val)

col_labels = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
row_labels = ['precision', 'recall', 'f1-score', 'support']
```

```
fig = plt.figure(figsize=(9,4))

ax = fig.add_subplot(111, frameon=True, xticks=[], yticks=[])
the_table=plt.table(cellText=B, rowLabels=row_labels, colLabels=col_labels,
                    colWidths=[0.1, 0.1, 0.1, 0.1,0.1, 0.1, 0.1], loc='center',cellLoc='center')
the_table.set_fontsize(20)

the_table.scale(_1.43,3.6)
```

report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

50/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11



Comment 1

conclusion

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

GRADING FORM: SMU GENERIC_GJ

TAO CHEN

50

RESEARCH-INFORMED LITERATURE



Refs are quite light here. Check their formatting too. Use the School's preferred referencing method please.

4

KNOWLEDGE AND UNDERSTANDING



Essential ideas identified and discussed but lacking sufficient detail in places. I was expecting you to develop and modify your own networks, however you do compare two differing networks and discuss these.

7

ANALYSIS



You have compared two networks though modification of hyper parameters and their effects on accuracy is poorly documented. How many modifications did you make? Evaluation results covers the required areas but you do not analyse/discuss these in depth. Biases in dataset not addressed.

16.5

PRACTICAL APPLICATION AND DEP.



Two existing networks used - what about your design? Trained and evaluated - but validation, splitting of training and test sets not explained. Confusion matrices and evaluation reports generated. Biases in datasets not addressed.

20.5

SKILLS FOR PROFESSIONAL PRAC.



Report could be better sections. Diagrams included - good. More please. Refs poor.

2