Week8 assignment

Name: Tianze Zhang
Student No:19334401

(i)(a)

```
def convolution2d(image, kernel):
    m, n = kernel.shape
    if (m == n):
        y, x = image.shape
        y = y - m + 1
        x = x - m + 1
        output = np.zeros((y,x))
        for i in range(y):
            for j in range(x):
                output[i][j] = np.sum(image[i:i+m, j:j+m]*kernel)
    return output
```

The function implemented takes a n*n numpy array as image and a k*k numpy array as kernel for the function parameters. Here we assume that the stride is 1 and valid convolution (padding = 0), resulting of the output of this function is a (n-k-1) *(n-k-1) dimension numpy array. Each element of output is calculated by summing each corresponding kernel's element to the input's element of where the kernel frame slides to.

The correctness of this function was tested against week 8 lecture slide "intro to convolution" page 7, where the input matrix is [[1,2,3,4,5],[1,3,2,3,10],[3,2,1,4,5],[6,1,1,2,2],[3,2,1,5,4]] and the kernel is [[1, 0,-1],[1,0,-1],[1,0,-1]]. By using those values of parameters of the function the correct output [[-1, -4, -1 4], [6, -3, -13], [9, -6, -8]] is returned (same as result from lecture slide).

(b)The image I pick is a triangle with 20x20 pixels.

```
kernal1 = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
print("kernal1:\n")
print(convolution2d(r,kernal1))
kernal2 = np.array([[0,-1,0],[-1,8,-1],[0,-1,0]])
print("kernal2:\n")
print(convolution2d(r,kernal2))
```

The output matrix after applying the above kernels to r are (running the above code):

```
[[   0.    0.    0. ...    0.    0.    0.]
 [   0.    0.    0. ...    0.    0.    0.]
 [   0.    0.    0. ...    0.    0.    0.]
 ...
 [1020.  255.    0. ...    0.  255. 1020.]
 [ 510.    0.    0. ...    0.    0.  510.]
 [ 255.    0.    0. ...    0.    0.  255.]]
```

```
kernal2:

[[   0.    0.    0. ...    0.    0.    0.]
 [   0.    0.    0. ...    0.    0.    0.]
 [   0.    0.    0. ...    0.    0.    0.]
 ...
 [1530. 1020. 1020. ... 1020. 1020. 1530.]
 [1275. 1020. 1020. ... 1020. 1020. 1275.]
 [1020. 1020. 1020. ... 1020. 1020. 1020.]]
```

(ii)(a) In the ConvNet, there are 1 input layer, 4 convolutional layers and one dense layer (FC layer).
In the input layer 32*32*3, there are 3 input channels.
In the first Conv layer, each kernel is 3 x 3 size, 16 output channels, padding is same which means output size is same as input, stride with default value 1 (means move kernel along by one column each step). activation method is relu.
In the second Conv layer, each kernel is 3 x 3 size, 16 output channels, padding is same, and stride is 2(means move kernel along by two columns each step)
In the third conv layer, each kernel is 3 x 3 size, 32 output channels, padding is same (output size = input size), stride is 1(moving kernel 1 columns each step.)
In the fourth conv layer, each kernel is 3 x 3 size, 32 output channels, padding is same (output size = input size), stride is 2(moving kernel 2 columns each step.
In the dense/FC layer, 10 number of output classes/categories, using nonlinear SoftMax (multi-class logistic regression model) as classifier to get the final classification output. the loss function is cross entropy and use the accuracy metrics to measure model performance.

(b)(i)
From the result of keras model.summary(), it says there are total 37146 parameters in this model. The Dense/FC layer has the most parameters: 20490 in total, which is larger than the total parameters in the conv layers. The reason for this is that: with padding='same' in each of the conv layers, the input layer size(32x32) is not reduced in the conv layers, after flattening from the output of the last conv layer, the total parameter for the dense layer is then calculated with 32*32*32* number of neurons in FC layer, so it is the biggest than those of any conv layers.

The precision of the training data is 63% where the precision of the testing data is 49% this means when the model predicts positive, the misclassification on the training data is less than the testing data. The recall of the training data is 62% while the recall of the testing data is 50%, this means when the model predicts negative, the misclassification on the training data is slightly less than the testing data. The f1 score of the training data is 62% while the f1 score of the testing data is 49%, this mean the model overall accuracy on the training data is higher than the testing data. the baseline model is implemented using a dummy classifier from sklearn that predicts the most common case, it gives 1% for precision, 10% for recall and 2% for f1 score for both training and testing. This means our current model outperforms the baseline since all scores beats the baseline.

(ii)

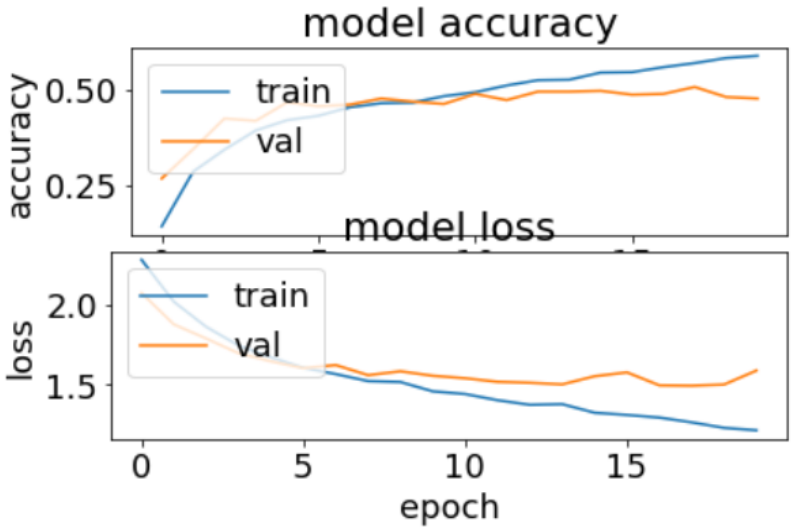

Figure 1.0

From figure 1.0, the model is underfitting because the training data is doing better than the testing data, where the training data has a much more accuracy and less loss than testing. This is because we are keeping epoch at 20 which means the model goes over the training data 20 times to train the model. However, since we are only using a small proportion of data (10%), the model will train the model based on 10% of data, which result overfit quicker.

(iii)

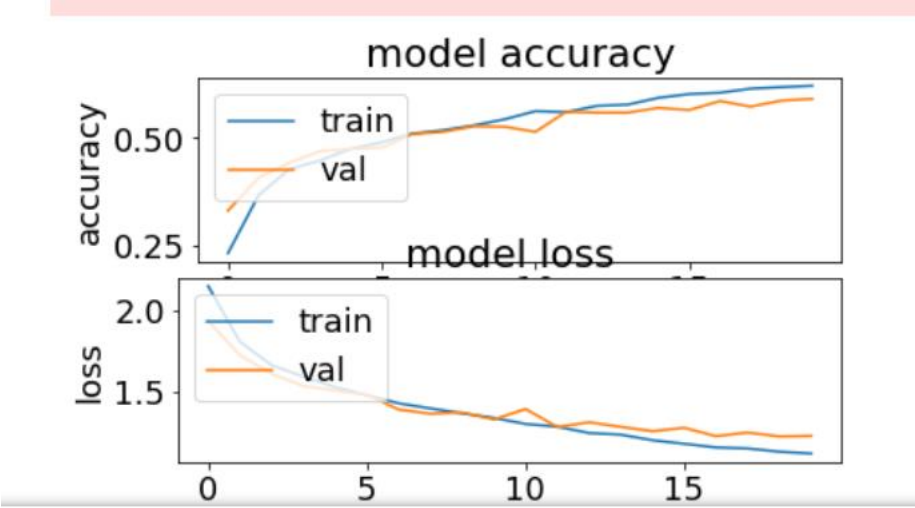

Figure 2.0

From figure 2.0, when increase the training data to 10k, in compared to 5k of training data. The performance of the training data remains similar, with a precision, recall and f1 score of 68%. However, results in a significant performance increase for the testing data, with a precision, recall and f1 score of 58%, around 10% increase. This can be due to the reduce of overfitting, since when we increase the size of the data, the benefit of doing multiple epochs outweighs the fact of going through the data 20 times, since we have larger amount of data, there are less overfitting. However, when the size of dataset increases, the time taken to train also increases.

(iv)

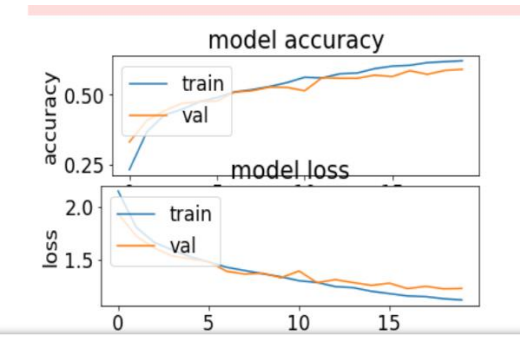

Figure 3.0: L1=0

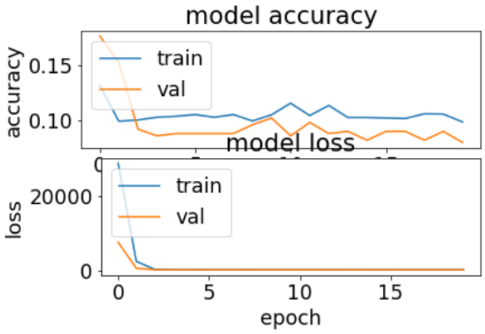

Figure 3.1: L1 = 100

As figure 3.0, when decreasing the L1 penalty to 0, the loss of model decreased and accuracy of the model increased, in compared to Figure 1.0 when L1 = 0.0001. However, when increasing L1 to a significant amount i.e., 100. The model accuracy decreased significantly, and the model loss also decreased significantly, this is due to overfitting as the penalty is too high. Therefore, lower L1 usually deals better with overfitting, however, it should only be decreased to a certain amount to avoid underfitting.

(c)(i) See code
(ii)

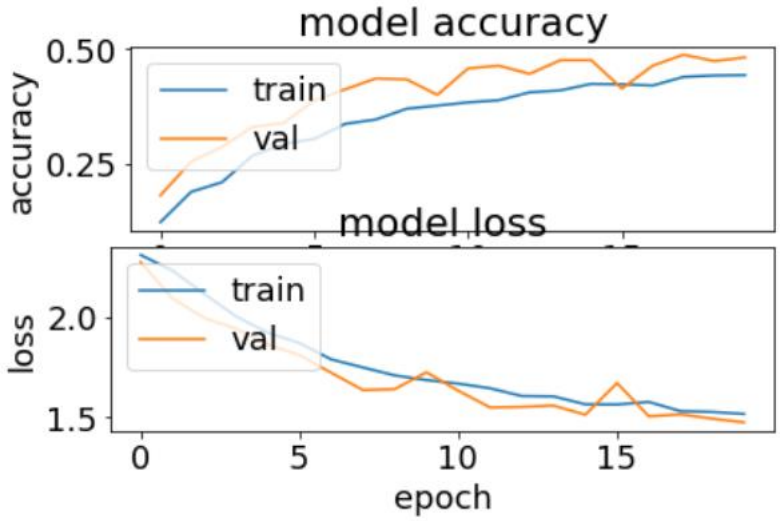

Figure 4.0

After doing the max-pooling the number of parameters reduced to 17946. Comparing to the original network, the training data's precision and recall decreased to 49% while the f1 score decreased to 50%. The testing data's precision and recall remained relatively same of 47%, and f1 score also relatively the same of 48%. However, this is preferred than the original network because the testing data is doing better than the training data as Figure 4.0, which means some overfitting is reduced by reducing some unnecessary parameters. The max-pooling also ran faster than the original one because there were less parameters, due to max-pooling downsamples and reduces the size of the matrix and calculate the max element in each block, and only keeping the max element, where reduces the process time.

(d)

In the example, as add two more conv layers(8 output channels) in front of the previous existing model(4 conv layers), the model is becoming deeper, the extra conv layers in front, which extract more features(extra generic features),combined with later conv layers extracting complex features (Feature Maps), the decision made in FC layer is more accurate in prediction, reduce the model under-fitting but the training time takes longer because it leads to a large number of parameters to be tuned. But adding more layers can lead to vanishing gradient, the skip connections (skip some layers to reach the beginner layers). The more complex model (The deeper of the models), means more parameters to learn, more training data is needed to avoid underfitting, the more layer added but to cross the optimal threshold can also lead to overfitting, more overfitting controls are needed (max pooling and dropout). There are trade-offs balancing among those factors to meet the specific performance requirements and business use cases when creating the CNN models.

Appendix

```
import numpy as np

from PIL import Image

#assume stride=1 and padding=0(valid convolution ) output=(n-k-1)*(n-k-1) dimension

def convolution2d(image, kernel):

    m, n = kernel.shape

    if (m == n):

        y, x = image.shape

        y = y - m + 1

        x = x - m + 1

        output = np.zeros((y,x))

        for i in range(y):

            for j in range(x):

                output[i][j] = np.sum(image[i:i+m, j:j+m]*kernel)

    return output

    #test with week 8 lecture slide 7 from intro to convolution

    image = np.array([[1,2,3,4,5],[1,3,2,3,10],[3,2,1,4,5],[6,1,1,2,2],[3,2,1,5,4]])

    kernal = np.array([[1,0,-1],[1,0,-1],[1,0,-1]])

    convolution2d(image,kernal)

    im = Image.open('200px-Triangle_Orange.svg.png')

    rgb = np.array(im.convert('RGB'))

    r=rgb[:,:,0]

    Image.fromarray(np.uint8(r)).show()
```

```
kernal1 = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
print("kernal1:\n")
print(convolution2d(r,kernal1))
kernal2 = np.array([[0,-1,0],[-1,8,-1],[0,-1,0]])
print("kernal2:\n")
print(convolution2d(r,kernal2))
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D, LeakyReLU
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys

# Model / data parameters
num_classes = 10
input_shape = (32, 32, 3)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
n=5000
x_train = x_train[1:n]; y_train=y_train[1:n]
#x_test=x_test[1:500]; y_test=y_test[1:500]
```

```
    # Scale images to the [0, 1] range
    x_train = x_train.astype("float32") / 255
    x_test = x_test.astype("float32") / 255
    print("orig x_train shape:", x_train.shape)

    # convert class vectors to binary class matrices
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    use_saved_model = False
    if use_saved_model:
        model = keras.models.load_model("cifar.model")
    else:
        model = keras.Sequential()
        model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train.shape[1:],activation='relu'))
        model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
        model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
        model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
        model.summary()

        batch_size = 128
        epochs = 20
        history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
        model.save("cifar.model")
        plt.subplot(211)
        plt.plot(history.history['accuracy'])
```

```
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.subplot(212)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss'); plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()

preds = model.predict(x_train)
y_pred = np.argmax(preds, axis=1)
y_train1 = np.argmax(y_train, axis=1)
print(classification_report(y_train1, y_pred))
print(confusion_matrix(y_train1,y_pred))

preds = model.predict(x_test)
y_pred = np.argmax(preds, axis=1)
y_test1 = np.argmax(y_test, axis=1)
print(classification_report(y_test1, y_pred))
print(confusion_matrix(y_test1,y_pred))
# Model / data parameters
num_classes = 10
input_shape = (32, 32, 3)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
n=5000
```

```
x_train = x_train[1:n]; y_train=y_train[1:n]
#x_test=x_test[1:500]; y_test=y_test[1:500]

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
print("orig x_train shape:", x_train.shape)

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

use_saved_model = False
if use_saved_model:
    model = keras.models.load_model("cifar.model")
else:
    model = keras.Sequential()
    model.add(Conv2D(16, (3,3), padding='same', input_shape=x_train.shape[1:],activation='relu'))
    model.add(Conv2D(16, (3,3), strides=(2,2), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
    model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
    model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
    model.summary()

    batch_size = 128
```

```
    epochs = 20
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
    model.save("cifar.model")
    plt.subplot(211)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.subplot(212)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss'); plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()

preds = model.predict(x_train)
y_pred = np.argmax(preds, axis=1)
y_train1 = np.argmax(y_train, axis=1)
print(classification_report(y_train1, y_pred))
print(confusion_matrix(y_train1,y_pred))

preds = model.predict(x_test)
y_pred = np.argmax(preds, axis=1)
y_test1 = np.argmax(y_test, axis=1)
print(classification_report(y_test1, y_pred))
print(confusion_matrix(y_test1,y_pred))
```