Esha Adhawade

Homework 3

CSCE 421 500

November 15, 2022

Question 1: Machine learning for object recognition

(a) (1 point) Visualization: Randomly select and visualize 5-6 images (no need to include all the classes).

Code

Libraries

```
from tensorflow import keras
import sys
from matplotlib import pyplot
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
```

Load CIFAR10 data and print images

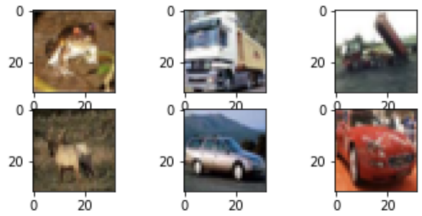
```
# load dataset
(trainX, trainY), (testX, testY) = cifar10.load_data()

# convert to grayscale images
from skimage.color import rgb2gray
#trainX = rgb2gray(trainX)
#testX = rgb2gray(testX)

# train / test dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainY.shape))
print('Test: X=%s, y=%s' % (testX.shape, testY.shape))
# plot images
for i in range(6):
    pyplot.subplot(330 + 1 + i)
    # plot data
    pyplot.imshow(trainX[i])
pyplot.show()
```

Output

```
Train: X=(50000, 32, 32, 3), y=(50000, 1)
Test: X=(10000, 32, 32, 3), y=(10000, 1)
```



Reflections

In the code above, I removed the grayscale and kept the images in color because the accuracy for the convolutional neural network was slightly better, however, I did originally test it in grayscale.

The data is also split where 50000 is for the train set and 10000 is for the test set.

(b) (1 point) Data exploration: Count the number of samples per class in the training data.

Code

Libraries

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import os
from six.moves import cPickle
```

Data loaded in batches

```
#data loaded in batches
files = {'batch1': 'data_batch_1',
             'batch2': 'data_batch_2',
             'batch3': 'data_batch_3',
             'batch4': 'data_batch_4',
             'batch5': 'data_batch_5'
             }

#display the batches
def stats(filename,sample_id):
    file = open(filename,'rb')
    data = cPickle.load(file, encoding='latin')
    file.close()
    resX=data['data']
    resY=data['labels']
    print(len(resY)) #resY - list
    for y in set(resY):
        print(y,resY.count(y), end = '  ')
```

Print

```
print("Batch 1")
stats(files["batch1"],4)
print("Batch 2")
stats(files["batch2"],4)
print("Batch 3")
stats(files["batch3"],4)
print("Batch 4")
stats(files["batch4"],4)
print("Batch 5")
stats(files["batch5"],4)
```

Output

```
Batch 1
10000
0 1005  1 974  2 1032  3 1016  4 999  5 937  6 1030  7 1001  8 1025  9 981

Batch 2
10000
0 984  1 1007  2 1010  3 995  4 1010  5 988  6 1008  7 1026  8 987  9 985

Batch 3
10000
0 994  1 1042  2 965  3 997  4 990  5 1029  6 978  7 1015  8 961  9 1029

Batch 4
10000
0 1003  1 963  2 1041  3 976  4 1004  5 1021  6 1004  7 981  8 1024  9 983

Batch 5
10000
0 1014  1 1014  2 952  3 1016  4 997  5 1025  6 980  7 977  8 1003  9 1022
```

Reflections

For this case, I loaded the data based on the batches it was given in and the training set has a total of 50000 data points, and above is a display of the number of data per class.

(c) (4 points) Image classification with FNNs: In this part, you will use a feedforward neural network (FNN) (also called "multilayer perceptron") to perform the object classification task. The input of the FNN comprises of all the pixels of the image. Use one of the five batches of the training data as a validation set.

c.i) (3 points) Experiment on the validation set with different FNN hyper-parameters, e.g. #layers, #nodes per layer, activation function, dropout, weight regularization, etc. Choose 3 hyper-parameter combinations and for each combination, please do the following:

Code

Libraries

```
from tensorflow import keras
import tensorflow as tf
import sys
from matplotlib import pyplot
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
```

Load and Normalize data

```
def load_dataset():
    #load dataset
    (trainX, trainY), (testX, testY) = cifar10.load_data()
    #encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

```
def normalize(train, test):
    # convert to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize- range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    return train_norm, test_norm
```

Training and Testing sets

```
num_classes = len(np.unique(trainY))
# training/validation sets
(x_train, x_valid) = x_train[10000:], x_train[:10000]
(y_train, y_valid) = y_train[5000:], y_train[:10000]

# print shape
print('x_train shape:', x_train.shape)

# training, validation, and test
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print(x_valid.shape[0], 'validation samples')
```

```
x_train shape: (40000, 32, 32, 3)
40000 train samples
10000 test samples
10000 validation samples
```

(1) monitor the loss on the train and validation set across the epochs of the FNN training;
(2) report the final classification accuracy on the training and validation sets;
(3) report the running time for training the FNN;
(4) report the # parameters that are learned for each FNN.
Note: If running the FNN takes a long time, you can subsample the training data (i.e., choose a random set of samples from training) or sub-sample the input images to a smaller size (e.g., 24 ×24).

Code

FNN model including activation function, dropout, and two hidden layers

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten

#fnn  model
def fnn_model():
    model = Sequential()
    model.add(Flatten(input_shape = x_train.shape[1:]))
    model.add(Dense(1000, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
    model.summary()
    return model


def run_test():
    # load dataset
    trainX, trainY, testX, testY = load_dataset()
    trainX, testX = normalize(trainX, testX)
    # define model
    model = fnn_model()
    history = model.fit(trainX, trainY, epochs=10, batch_size=64, validation_data=(testX, testY))
    _, acc = model.evaluate(testX, testY, verbose=0)
    print(history)
    print('accuracy %.3f' % (acc * 100.0))

run_test()
```

Output

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              3073000
_________________________________________________________________
dropout_1 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               512512
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 3,590,642
Trainable params: 3,590,642
Non-trainable params: 0
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 19s 371us/step - loss: 1.9573 - accuracy: 0.2958 - val_loss: 1.7900 -
val_accuracy: 0.3763
Epoch 2/10
50000/50000 [==============================] - 20s 398us/step - loss: 1.7778 - accuracy: 0.3689 - val_loss: 1.6823 -
val_accuracy: 0.4053
Epoch 3/10
50000/50000 [==============================] - 22s 442us/step - loss: 1.7002 - accuracy: 0.3974 - val_loss: 1.6228 -
val_accuracy: 0.4281
Epoch 4/10
50000/50000 [==============================] - 23s 457us/step - loss: 1.6467 - accuracy: 0.4194 - val_loss: 1.5762 -
val_accuracy: 0.4434
Epoch 5/10
50000/50000 [==============================] - 24s 483us/step - loss: 1.6052 - accuracy: 0.4356 - val_loss: 1.5541 -
val_accuracy: 0.4519
Epoch 6/10
50000/50000 [==============================] - 26s 524us/step - loss: 1.5739 - accuracy: 0.4454 - val_loss: 1.5301 -
val_accuracy: 0.4596
Epoch 7/10
50000/50000 [==============================] - 28s 561us/step - loss: 1.5409 - accuracy: 0.4551 - val_loss: 1.4956 -
val_accuracy: 0.4736
Epoch 8/10
50000/50000 [==============================] - 29s 575us/step - loss: 1.5120 - accuracy: 0.4677 - val_loss: 1.4864 -
val_accuracy: 0.4684
Epoch 9/10
50000/50000 [==============================] - 28s 566us/step - loss: 1.4923 - accuracy: 0.4743 - val_loss: 1.4581 -
val_accuracy: 0.4818
Epoch 10/10
50000/50000 [==============================] - 28s 570us/step - loss: 1.4697 - accuracy: 0.4814 - val_loss: 1.4418 -
val_accuracy: 0.4922
<keras.callbacks.callbacks.History object at 0x7f8dd3e6cd10>
accuracy 49.220
```

Reflections

For this part, I chose 3 combinations which are the activation, hidden layers, and dropout, however I added each variant to the basic FNN model and got the overall accuracy of 49.22 which is the highest accuracy.

(c.ii) (1 point) Run the best model that was found based on the validation set from question (c.i) on the testing set. Report the classification accuracy on the testing set. Report the confusion matrix for each class.

Note: The confusion matrix is a 10×10 matrix; its rows correspond to the actual labels for each class, while its columns correspond to the predicted classes. Element (i, j) includes the number of samples that belonged to the ith class and were predicted as the jth class. In a perfect classification task, the non-diagonal elements of the matrix will be all non-zero.

Code

Model Saved

```
from keras.datasets import cifar10
from keras.models import load_model
from keras.utils import to_categorical

def run_test():
    trainX, trainY, testX, testY = load_dataset()
    trainX, testX = normalize(trainX, testX)
    # load model
    model = load_model('final_model1.h5')
    # evaluate model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('accuracy %.3f' % (acc * 100.0))
```

Confusion matrix

```
from sklearn.metrics import confusion_matrix
import itertools
p_test = model.predict(testX).argmax(axis=1)
cm = confusion_matrix(testY, p_test)
plot_confusion_matrix(cm, list(range(10)))
```

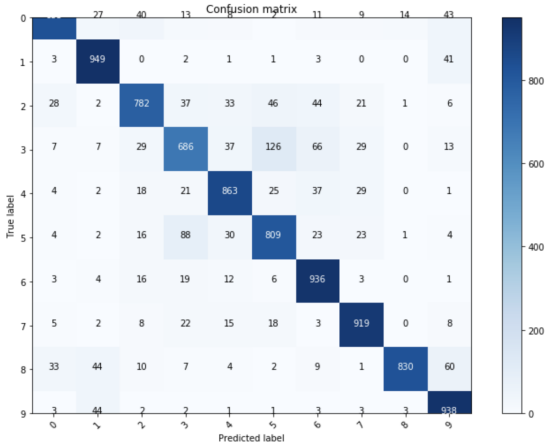
Output

Accuracy

```
run_test()
```

```
accuracy 48.920
```

Confusion matrix



Reflections

The model I saved is the same as the one above which had the accuracy for 49.2.

(d) (4 points) Image classification with CNNs: In this part, you will use a convolutional neural network (CNN) to perform the object classification task.
(d.i) (3 points) Experiment on the validation set with different CNN hyper-parameters, e.g. #layers, filter size, stride size, activation function, dropout, weight regularization, etc. Choose 3 hyper-parameter combinations and for each combination, please do the following:

Code

Load the data

```
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = cifar10.load_data()
    #encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

Normalize Images

```
# scale pixels
def normalize(train, test):
    # convert to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize- range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    return train_norm, test_norm
```

(1) monitor the loss on the train and validation set across the epochs of the CNN training;
(2) report the final classification accuracy on the training and validation sets;
(3) report the running time for training the CNN;
(4) report the # parameters that are learned for each CNN.

Code

Model Type: 1 Layer

```
#1 layer
# cnn model
def cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=
                     (32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
def run_test():
    # load data
    trainX, trainY, testX, testY = load_dataset()
    # normalize
    trainX, testX = normalize(trainX, testX)
    # model
    model = cnn_model()
    # fit model
    history = model.fit(trainX, trainY, epochs=10, batch_size=64, validation_data=(testX, testY))
    # eval model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print(history)
    print('accuracy %.3f' % (acc * 100.0))
```

Model Type: 3 Layers

```
#3 layers
# define cnn model
def cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     input_shape=(32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Model Type: Dropout Regularization

```
#dropout
import sys
from matplotlib import pyplot
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD

# define cnn model
def cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     input_shape=(32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Model Type: Weight Regularization

```
#weight
import sys
from matplotlib import pyplot
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.regularizers import l2

# define cnn model
def cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001), input_shape=(32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform', kernel_regularizer=l2(0.001)))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Output

Model Type: 1 Layer

This shows the loss on train and validation sets, total accuracy, and running time

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 51s 1ms/step - loss: 1.7536 - accuracy: 0.3747 - val_loss: 1.5227 - va
l_accuracy: 0.4539
Epoch 2/10
50000/50000 [==============================] - 49s 983us/step - loss: 1.4201 - accuracy: 0.4949 - val_loss: 1.3407 -
val_accuracy: 0.5160
Epoch 3/10
50000/50000 [==============================] - 52s 1ms/step - loss: 1.2678 - accuracy: 0.5509 - val_loss: 1.2277 - va
l_accuracy: 0.5591
Epoch 4/10
50000/50000 [==============================] - 54s 1ms/step - loss: 1.1650 - accuracy: 0.5914 - val_loss: 1.1548 - va
l_accuracy: 0.5912
Epoch 5/10
50000/50000 [==============================] - 52s 1ms/step - loss: 1.0885 - accuracy: 0.6166 - val_loss: 1.1168 - va
l_accuracy: 0.6068
Epoch 6/10
50000/50000 [==============================] - 51s 1ms/step - loss: 1.0238 - accuracy: 0.6428 - val_loss: 1.1007 - va
l_accuracy: 0.6126
Epoch 7/10
50000/50000 [==============================] - 50s 1ms/step - loss: 0.9676 - accuracy: 0.6631 - val_loss: 1.0881 - va
l_accuracy: 0.6206
Epoch 8/10
50000/50000 [==============================] - 50s 1ms/step - loss: 0.9159 - accuracy: 0.6824 - val_loss: 1.0527 - va
l_accuracy: 0.6350
Epoch 9/10
50000/50000 [==============================] - 52s 1ms/step - loss: 0.8665 - accuracy: 0.7000 - val_loss: 1.0086 - va
l_accuracy: 0.6471
Epoch 10/10
50000/50000 [==============================] - 52s 1ms/step - loss: 0.8200 - accuracy: 0.7166 - val_loss: 0.9999 - va
l_accuracy: 0.6487
<keras.callbacks.callbacks.History object at 0x7fb168108ad0>
accuracy 64.870
```

Model Summary

```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_121 (Conv2D)          (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_122 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_61 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
flatten_24 (Flatten)         (None, 8192)              0
_________________________________________________________________
dense_47 (Dense)             (None, 128)               1048704
_________________________________________________________________
dense_48 (Dense)             (None, 10)                1290
=================================================================
Total params: 1,060,138
Trainable params: 1,060,138
Non-trainable params: 0
```

Model Type: 3 Layers

This shows the loss on train and validation sets, total accuracy, and running time

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 124s 2ms/step - loss: 1.7359 - accuracy: 0.3751 - val_loss: 1.4599 - v
al_accuracy: 0.4669
Epoch 2/10
50000/50000 [==============================] - 94s 2ms/step - loss: 1.3855 - accuracy: 0.5031 - val_loss: 1.3731 - va
l_accuracy: 0.5054
Epoch 3/10
50000/50000 [==============================] - 118s 2ms/step - loss: 1.2128 - accuracy: 0.5714 - val_loss: 1.1713 - v
al_accuracy: 0.5822
Epoch 4/10
50000/50000 [==============================] - 89s 2ms/step - loss: 1.0840 - accuracy: 0.6201 - val_loss: 1.0623 - va
l_accuracy: 0.6279
Epoch 5/10
50000/50000 [==============================] - 89s 2ms/step - loss: 0.9879 - accuracy: 0.6534 - val_loss: 1.0398 - va
l_accuracy: 0.6384
Epoch 6/10
50000/50000 [==============================] - 88s 2ms/step - loss: 0.9049 - accuracy: 0.6854 - val_loss: 0.9569 - va
l_accuracy: 0.6658
Epoch 7/10
50000/50000 [==============================] - 90s 2ms/step - loss: 0.8312 - accuracy: 0.7134 - val_loss: 0.9454 - va
l_accuracy: 0.6718
Epoch 8/10
50000/50000 [==============================] - 94s 2ms/step - loss: 0.7696 - accuracy: 0.7332 - val_loss: 0.8921 - va
l_accuracy: 0.6946
Epoch 9/10
50000/50000 [==============================] - 88s 2ms/step - loss: 0.7071 - accuracy: 0.7536 - val_loss: 0.8966 - va
l_accuracy: 0.6973
Epoch 10/10
50000/50000 [==============================] - 88s 2ms/step - loss: 0.6535 - accuracy: 0.7735 - val_loss: 0.8978 - va
l_accuracy: 0.6987
<keras.callbacks.callbacks.History object at 0x7fafd09d4d50>
accuracy 69.870
```

Model Summary

```
Model: "sequential_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_123 (Conv2D)          (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_124 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_62 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_125 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_126 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_63 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_127 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_128 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_64 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
flatten_25 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_49 (Dense)             (None, 128)               262272
_________________________________________________________________
dense_50 (Dense)             (None, 10)                1290
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
```

Model Type: Dropout Regularization

This shows the loss on train and validation sets, total accuracy, and running time

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 89s 2ms/step - loss: 2.0290 - accuracy: 0.2472 - val_loss: 1.7801 - va
l_accuracy: 0.3791
Epoch 2/10
50000/50000 [==============================] - 106s 2ms/step - loss: 1.6870 - accuracy: 0.3769 - val_loss: 1.5217 - v
al_accuracy: 0.4553
Epoch 3/10
50000/50000 [==============================] - 98s 2ms/step - loss: 1.5177 - accuracy: 0.4424 - val_loss: 1.3852 - va
l_accuracy: 0.5013
Epoch 4/10
50000/50000 [==============================] - 93s 2ms/step - loss: 1.4120 - accuracy: 0.4871 - val_loss: 1.2814 - va
l_accuracy: 0.5386
Epoch 5/10
50000/50000 [==============================] - 93s 2ms/step - loss: 1.3344 - accuracy: 0.5171 - val_loss: 1.2081 - va
l_accuracy: 0.5684
Epoch 6/10
50000/50000 [==============================] - 108s 2ms/step - loss: 1.2696 - accuracy: 0.5431 - val_loss: 1.1768 - v
al_accuracy: 0.5797
Epoch 7/10
50000/50000 [==============================] - 94s 2ms/step - loss: 1.2041 - accuracy: 0.5665 - val_loss: 1.1202 - va
l_accuracy: 0.5974
Epoch 8/10
50000/50000 [==============================] - 101s 2ms/step - loss: 1.1448 - accuracy: 0.5912 - val_loss: 1.0251 - v
al_accuracy: 0.6346
Epoch 9/10
50000/50000 [==============================] - 109s 2ms/step - loss: 1.0935 - accuracy: 0.6083 - val_loss: 0.9986 - v
al_accuracy: 0.6483
Epoch 10/10
50000/50000 [==============================] - 105s 2ms/step - loss: 1.0487 - accuracy: 0.6263 - val_loss: 0.9609 - v
al_accuracy: 0.6577
<keras.callbacks.callbacks.History object at 0x7fb1b15bcbd0>
accuracy 65.770
```

Model Summary

```
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_135 (Conv2D)          (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_136 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_68 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_33 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_137 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_138 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_69 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_34 (Dropout)         (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_139 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_140 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_70 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_35 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_27 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_53 (Dense)             (None, 128)               262272
_________________________________________________________________
dropout_36 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_54 (Dense)             (None, 10)                1290
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
```

Model Type: Weight Regularization

This shows the loss on train and validation sets, total accuracy, and running time

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 89s 2ms/step - loss: 2.8981 - accuracy: 0.3646 - val_loss: 2.5919 - va
l_accuracy: 0.4689
Epoch 2/10
50000/50000 [==============================] - 88s 2ms/step - loss: 2.5147 - accuracy: 0.4966 - val_loss: 2.4013 - va
l_accuracy: 0.5313
Epoch 3/10
50000/50000 [==============================] - 87s 2ms/step - loss: 2.3036 - accuracy: 0.5651 - val_loss: 2.2680 - va
l_accuracy: 0.5711
Epoch 4/10
50000/50000 [==============================] - 88s 2ms/step - loss: 2.1615 - accuracy: 0.6092 - val_loss: 2.1373 - va
l_accuracy: 0.6114
Epoch 5/10
50000/50000 [==============================] - 91s 2ms/step - loss: 2.0412 - accuracy: 0.6460 - val_loss: 2.0516 - va
l_accuracy: 0.6395
Epoch 6/10
50000/50000 [==============================] - 92s 2ms/step - loss: 1.9341 - accuracy: 0.6754 - val_loss: 2.0636 - va
l_accuracy: 0.6300
Epoch 7/10
50000/50000 [==============================] - 92s 2ms/step - loss: 1.8396 - accuracy: 0.7017 - val_loss: 1.8883 - va
l_accuracy: 0.6832
Epoch 8/10
50000/50000 [==============================] - 90s 2ms/step - loss: 1.7568 - accuracy: 0.7217 - val_loss: 1.8297 - va
l_accuracy: 0.6940
Epoch 9/10
50000/50000 [==============================] - 90s 2ms/step - loss: 1.6782 - accuracy: 0.7435 - val_loss: 1.8052 - va
l_accuracy: 0.7002
Epoch 10/10
50000/50000 [==============================] - 91s 2ms/step - loss: 1.6085 - accuracy: 0.7630 - val_loss: 1.7695 - va
l_accuracy: 0.7087
<keras.callbacks.callbacks.History object at 0x7fb1d13bd750>
accuracy 70.870
```

Model Summary

```
Model: "sequential_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_147 (Conv2D)          (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_148 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_74 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_149 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_150 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_75 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_151 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_152 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_76 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
flatten_29 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_57 (Dense)             (None, 128)               262272
_________________________________________________________________
dense_58 (Dense)             (None, 10)                1290
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
```

Reflections

Possibly increasing the number of epochs could improve the accuracy of models.

How do these metrics compare to the FNN?

CNN accuracies are much higher than FNN. Through the process, FNN is much harder to train even with hyperparamenter tuning in comparison to CNN, which generally improved in accuracy with hyperprameter tuning.

(d.ii) (1 point) Run the best model that was found based on the validation set from question (d.i) on the testing set. Report the classification accuracy on the testing set.

Code

From above the weight regularization had the best accuracy.

```
# define cnn model
def cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001), input_shape=(32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     kernel_regularizer=l2(0.001)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform', kernel_regularizer=l2(0.001)))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

def run_test():
    trainX, trainY, testX, testY = load_dataset()
    trainX, testX = normalize(trainX, testX)
    model = cnn_model()
    model.fit(trainX, trainY, epochs=10, batch_size=64)
    model.save('final_model.h5')
```

Output

```
from keras.datasets import cifar10
from keras.models import load_model
from keras.utils import to_categorical

def run_test():
    trainX, trainY, testX, testY = load_dataset()
    trainX, testX = normalize(trainX, testX)
    # load model
    model = load_model('final_model.h5')
    # evaluate model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('accuracy %.3f' % (acc * 100.0))
```

```
run_test()
```

```
accuracy 69.870
```

The accuracy is 69.870

Reflections

The accuracy is 69.870. This could be higher with an increased number of epochs.

How does this metric compare to the FNN?

The best model in CNN had a significantly better accuracy rate than FNN. CNN accuracy couldve been higher with more epochs. Nonetheless CNN accuracy was nearly 70% whereas FNN accuracy was < 50%, which is pretty low.

(e) (Bonus - 1.5 point) Bayesian optimization for hyper-parameter tuning: Instead of performing grid or random search to tune the hyper-parameters of the CNN, we can also try a model-based method for finding the optimal hyper-parameters through Bayesian optimization. This method performs a more intelligent search on the hyper-parameter space in order to estimate the best set of hyper-parameters for the data. Use publicly available libraries (e.g., hyperopt in Python) to perform a Bayesian optimization on the hyper-parameter space using the validation set. Report the emotion classification accuracy on the testing set. Hint: Check this and this source.

Code

Model definition using hyperparameters

```
def optimize_cnn(hyperparameter):
  # Define model using hyperparameters
  cnn_model = Sequential([Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], activation='relu',
                                 input_shape=(32,32,3)),
            MaxPooling2D(pool_size=(2,2)),
            Dropout(hyperparameter['dropout_prob']),
            Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu'),
            MaxPooling2D(pool_size=(2,2)),
            Dropout(hyperparameter['dropout_prob']),
            Flatten(),
            Dense(32, activation='relu'),
            Dense(10, activation='softmax'),])

    cnn_model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentropy', metrics=['accuracy'])

    cnn_model.fit(train_X, to_categorical(train_y), epochs=2, batch_size=256, verbose=0)

  performance = cnn_model.evaluate(valid_X, to_categorical(valid_y), verbose=0)
```

Utilizing hyperopt

```
space = {

    'conv_kernel_size': hp.choice('conv_kernel_size', [1, 3, 5]),

    'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.35),

    'optimizer': hp.choice('optimizer', ['Adam', 'sgd']),
}

trials = Trials()
best = fmin(
        optimize_cnn,
        space,
        algo=tpe.suggest,
        trials=trials,
        max_evals=10,
    )
```

Output

```
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.15884064221945612, 'optimizer': 'Adam'}
Accuracy:
0.10159999877214432
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.18858814375613153, 'optimizer': 'sgd'}
Accuracy:
0.10109999775886536
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2954079963786884, 'optimizer': 'Adam'}
Accuracy:
0.10249999910593033
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.13003919268397987, 'optimizer': 'sgd'}
Accuracy:
0.09790000319480896
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.12806301031114153, 'optimizer': 'sgd'}
Accuracy:
0.17020000517368317
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.1171055327264785, 'optimizer': 'sgd'}
Accuracy:
0.10010000318288803
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.31897005402662215, 'optimizer': 'Adam'}
Accuracy:
0.10029999911785126
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2605408224876211, 'optimizer': 'Adam'}
Accuracy:
0.10159999877214432
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.2518225357800308, 'optimizer': 'sgd'}
Accuracy:
0.09989999979734421
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.1765014323742451, 'optimizer': 'Adam'}
Accuracy:
```

```
Best Hyperparameters {'conv_kernel_size': 0, 'dropout_prob': 0.12806301031114153, 'optimizer': 1}
50000/50000 [==============================] - 27s 541us/step
==================================
Test Accuracy:  0.16553999483585358
```

Reflections

I utlized the code demonstration shown in class using the Cifar10 dataset, which brings me to around 0.17 test accuracy, which is lower than I expected.

f) (Bonus - 1.5 point) Fine-tuning: Use a pre-trained CNN (e.g., the pre-trained example of the MNIST dataset that we saw in class, or any other available pre-trained CNN) and fine- tune it on the CIFAR-10 data. Please experiment with different fine-tuning hyper-parameters (e.g., #layers to fine-tune, regularization during fine-tuning) on the validation set. Report the classification accuracy for all hyper-parameter combinations on the validation set. Also report the classification accuracy with the best hyper-parameter combination on the testing set.

Code

Batch Normalization added as another hyperparameter to fintune.

```
# define cnn model
def cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same',
                     input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.4))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Output

```
Epoch 1/10
781/781 [==============================] - 202s 258ms/step - loss: 2.1222 - accuracy: 0.2966 - val_loss: 1.5170 - val
_accuracy: 0.4468
Epoch 2/10
781/781 [==============================] - 196s 251ms/step - loss: 1.6311 - accuracy: 0.4074 - val_loss: 1.4228 - val
_accuracy: 0.4763
Epoch 3/10
781/781 [==============================] - 194s 249ms/step - loss: 1.4848 - accuracy: 0.4592 - val_loss: 1.5538 - val
_accuracy: 0.4381
Epoch 4/10
781/781 [==============================] - 197s 252ms/step - loss: 1.3960 - accuracy: 0.4890 - val_loss: 1.4756 - val
_accuracy: 0.4529
Epoch 5/10
781/781 [==============================] - 199s 255ms/step - loss: 1.3238 - accuracy: 0.5191 - val_loss: 1.2942 - val
_accuracy: 0.5188
Epoch 6/10
781/781 [==============================] - 193s 247ms/step - loss: 1.2749 - accuracy: 0.5393 - val_loss: 1.2025 - val
_accuracy: 0.5600
Epoch 7/10
781/781 [==============================] - 191s 244ms/step - loss: 1.2238 - accuracy: 0.5584 - val_loss: 1.2078 - val
_accuracy: 0.5601
Epoch 8/10
781/781 [==============================] - 190s 243ms/step - loss: 1.1822 - accuracy: 0.5763 - val_loss: 1.3467 - val
_accuracy: 0.5211
Epoch 9/10
781/781 [==============================] - 189s 242ms/step - loss: 1.1441 - accuracy: 0.5895 - val_loss: 1.1879 - val
_accuracy: 0.5679
Epoch 10/10
781/781 [==============================] - 196s 251ms/step - loss: 1.1118 - accuracy: 0.6018 - val_loss: 1.2002 - val
_accuracy: 0.5691
<keras.callbacks.callbacks.History object at 0x7fb155158650>
accuracy 56.910
```

Reflections

For both classification accuracy and the best combination of hyperparamter the accuracy is 56.9.