```
# DongKyu Kim
# ECE 471 CGML Assignment 4
# CIFAR-10
# Professor Curro

# library imports
import numpy as np
from tqdm import tqdm
import keras
import tensorflow as tf
from keras import optimizers, regularizers
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.layers import BatchNormalization, Activation, Input
from keras.callbacks import LearningRateScheduler

# Parameters
r = 0.9 # Ratio of training data to (training + validation)
batch_size = 128
epochs = 200
numclass = 10
learning_rate = 0.005

# Useful functions
def one_hot_encoding(data,numclass):
    targets = np.array(data).reshape(-1)
    return np.eye(numclass)[targets]

def lr_adaptive(self,epoch):
    lr = learning_rate
    if epoch > 20:
        lr = learning_rate/10
    elif epoch > 40:
        lr = learning_rate/20
    return lr

# Data Generation
class Data(object):
    def __init__(self,r,numclass):
        (xtemp,ytemp),(self.X_test,self.Y_test) = cifar10.load_data()
        mask = np.full(xtemp.shape[0],True)
        mask[np.random.choice(xtemp.shape[0],
            int(xtemp.shape[0]*(1-r)),replace=False)] = False
        self.numclass = numclass
        self.X_train = xtemp[mask].astype('float32')
        self.Y_train = ytemp[mask]
        self.X_val = xtemp[~mask].astype('float32')
        self.Y_val = ytemp[~mask]
        self.X_test = self.X_test.astype('float32')
        self.Y_test = self.Y_test
        self.X_train = self.normalize(self.X_train)
        self.X_val = self.normalize(self.X_val)
        self.X_test = self.normalize(self.X_test)
        self.Y_train = one_hot_encoding(self.Y_train,self.numclass)
        self.Y_val = one_hot_encoding(self.Y_val,self.numclass)
        self.Y_test = one_hot_encoding(self.Y_test,self.numclass)

    def normalize(self,X):
        X /= 255
        # These values are from
        # https://github.com/kuangliu/pytorch-cifar/issues/19
        mu = [0.4914,0.4822,0.4465]
```

```
        std = [0.2023,0.1994,0.2010]
        for i in range(0,2):
            X[:,:,:,i] = (X[:,:,:,i]-mu[i])/std[i]
        return X

# Model
class My_Model(object):
    def __init__(self,data,batch_size,epochs):
        self.data = data
        self.batch_size = batch_size
        self.epochs = epochs

    def convconv(self,filters,kernel_size,strides):
        return Conv2D(filters,kernel_size,strides =strides,
            padding='same',activation='relu')

    #The below block is from [1], Second branch is motivated by [3]
    #The below function is ultimately unused,
    #however I left this here for future usage.
    def residual_block(self,input,filter):
        x = BatchNormalization()(input)
        x = Activation('relu')(x)
        x = self.convconv(filter,[3,3],1)(x)
        x = BatchNormalization()(input)
        x = Activation('relu')(x)
        x = self.convconv(filter,[3,3],1)(x)

        y = BatchNormalization()(input)
        y = Activation('relu')(y)
        y = self.convconv(filter,[3,3],1)(y)
        y = BatchNormalization()(y)
        y = Activation('relu')(y)
        y = self.convconv(filter,[3,3],1)(y)
        return keras.layers.add([input,x,y])

    #This is a combination of residual block without the identity connection.
    def all_conv_block(self,filters,dropout,input):
        x = self.convconv(filters,[3,3],1)(input)
        x = BatchNormalization()(x)
        x = self.convconv(filters,[3,3],1)(x)
        x = BatchNormalization()(x)
        x = MaxPooling2D(2,strides = 2)(x)
        x = Dropout(dropout)(x)
        return x

    def build_model(self):
        input = Input(shape = self.data.X_train.shape[1:])
        x = self.all_conv_block(32,0.25,input)
        x = self.all_conv_block(64,0.35,x)
        x = self.all_conv_block(128,0.45,x)
        x = self.all_conv_block(256,0.35,x)
        x = Flatten()(x)
        x = Dense(1024,activation = 'relu')(x)
        y = Dense(self.data.numclass,activation='softmax')(x)
        self.model = Model(inputs = input, outputs = y)

    def train(self):
        self.optim = optimizers.Adam(learning_rate)
        self.model.compile(self.optim,'categorical_crossentropy',['accuracy'])
        self.datagen = keras.preprocessing.image.ImageDataGenerator(
            horizontal_flip = True,fill_mode = 'constant',
            width_shift_range = 4, height_shift_range = 4
            )
```

```
        self.datagen.fit(self.data.X_train)
        self.model.fit_generator(self.datagen.flow(self.data.X_train,
            self.data.Y_train,batch_size=self.batch_size),
            steps_per_epoch=len(self.data.X_train)/self.batch_size,
            epochs=self.epochs,
            validation_data=(self.data.X_val,self.data.Y_val),
            callbacks=[LearningRateScheduler(lr_adaptive)],verbose=2)

print('This is my cifar_10 case')
data = Data(r,numclass)
My_Model = My_Model(data,batch_size,epochs)
My_Model.build_model()
My_Model.train()
scores = My_Model.model.evaluate(data.X_test,data.Y_test,verbose=2)
print('Test loss:', scores[0])
print('Test accuracy:',scores[1])

# The state of the art cifar10 is 98.52% from Wikipedia achieved by
# AutoAugment: Learning Augmentation Policies from Data
# I started with my MNIST model, it didn't work well. Then I experimented with
# a deeper version of the MNIST model, but it just achieved 50% top-1 accuracy.
# I moved into residual neural network based on [1], and [2]. After 32 epochs,
# it converges at validation accuracy of 0.6479, and train set accuracy of
# 0.9967, with best validation accuracy of 0.6513. This model overfits, so I
# added a simple data augmentation scheme. With this augmentation method, I
# reach a validation accuracy of 0.6821, which is slightly better, but not that
# good. Then I did the data normalization off of a already known values because
# I didn't want to waste computational time, and result improved a lot to
# 0.8042 validation error.
# I tried elu, rotation_range in data augmentation, added another branch in
# residual network played around with learning rate, but all of these methods
# did not really help. Then I discussed with classmates and heard that simple
# convolutional neural network might work better. This made sense to me because
# even in the residual network paper, they mention for the deep versions of the
# model, they started with a simple conv net layers, and then add the residual
# network, and all the state of the art papers ran the model for 6 weeks, so
# maybe with my device and time, residual network wasn't effective. So I
# changed back to a deep 12 layer network with conv-net blocks, with adaptive
# learning rate, I achieved validation accuracy of 0.9120 and test accuracy of
# 0.9029.


# [1] Identity Mappings in Deep Residual Networks
#   https://arxiv.org/pdf/1603.05027.pdf
# [2] Deep Residual Learning for Image Recognition
#   https://arxiv.org/pdf/1512.03385.pdf
# [3] Shake-Shake regularization
#   https://arxiv.org/pdf/1705.07485.pdf
```