```
# DongKyu Kim
# ECE 471 CGML Assignment 4
# CIFAR-100
# 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 cifar100
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
batch_size = 128
epochs = 200
numclass = 100
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) = cifar100.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
        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')

    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','top_k_categorical_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_100 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])
print('Test top 5 accuracy:',scores[2])

# I omitted all the comments that I already made in the cifar-10.
# I stopped and checked this code on cifar-100 when cifar-10 reached 0.8042
# validation accuracy. The top5 accuracy for validation is 0.7435 and test
```

```
# accuracy is 0.7465. This is over the minimum requirement. The model was run
# for 20 epochs. It looks like if I did more epochs I will get a better
# accuracy, but instead of running it, I decided to make a better model.
# I indeed constructed a model that works better with cifar-10.
# Using the updated model top-5 validation accuracy was 0.8418 and
# top-5 test accuracy was 0.8374.
```