```
# Seyun Kim ECE472 Homework 4
# CIFAR100
```

```
import numpy as np
import tensorflow as tf
import tarfile
from sklearn.model_selection import train_test_split
```

```
# Upload cifar100 dataset
!wget https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz

# Unzip dataset
!tar --gunzip --extract --verbose --file=cifar-100-python.tar.gz

%cd cifar-100-python/
```

```
# Code provided by https://www.cs.toronto.edu/~kriz/
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

```
test = unpickle('test')
train = unpickle('train')
```

```
X_test = test[b'data']
y_test = test[b'fine_labels']
```

```
X_temp = train[b'data']
y_temp = train[b'fine_labels']
```

```
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size = 0.1)
```

```
X_train = np.reshape(X_train, (40000,32,32,3))
X_val = np.reshape(X_val, (10000,32,32,3))
X_test = np.reshape(X_test, (10000,32,32,3))

X_train = X_train.astype('float32')
X_val = X_val.astype('float32')
X_test = X_test.astype('float32')
```

```
# Nomalization vlaues taken from https://gist.github.com/weiaicunzai/e623931921efefd4c331622c
mu = [0.5071, 0.4867, 0.4408]
```

```
sigma = [0.2675, 0.2565, 0.2761]

def normalize(input):
  input = input/255
  for i in range(0,2):
    input[:,:,:,i] = (input[:,:,:,i]-mu[i])/sigma[i]
  return input


X_train = normalize(X_train)
X_val = normalize(X_val)
# X_test = normalize(X_test)


import keras
from keras.datasets import cifar100
from keras.models import Model, Sequential
from keras import optimizers, regularizers
from keras.layers import Dense, Dropout, Activation, Flatten, PReLU
from keras.layers import Conv2D, MaxPooling2D, Input, BatchNormalization
from keras.callbacks import LearningRateScheduler, EarlyStopping
from keras.preprocessing.image import ImageDataGenerator# Make label one hot encoded


# Make label one hot encoded
digits = 100
y_train = keras.utils.to_categorical(y_train, num_classes = digits)
y_val = keras.utils.to_categorical(y_val, num_classes = digits)
y_test = keras.utils.to_categorical(y_test, num_classes= digits)


def conv_layer(input, filter, reg, dropout_size, psize, weight_decay = 1e-4):
  if reg:
    x = Conv2D(filter, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_dec
    x = BatchNormalization()(x)
    x = Conv2D(filter, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_dec
    x = BatchNormalization()(x)
    x = MaxPooling2D(pool_size=(psize, psize))(x)
    x = Dropout(dropout_size)(x)
  else:
    x = Conv2D(filter, [3,3], padding = 'same')(input)
    x = Activation('relu')(x)
    x = BatchNormalization()(x)
    x = Conv2D(filter, [3,3], padding = 'same')(x)
    x = Activation('relu')(x)
    x = BatchNormalization()(x)
  x = MaxPooling2D(pool_size=(psize, psize))(x)
  x = Dropout(dropout_size)(x)
  return x


# Construct Neural Network
#define the convnet
```

```
#from keras.models import Sequential
r = True
weight_decay = 1e-4
psize = 2
input = Input(shape = X_train.shape[1:])
x = Conv2D(32, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), act
x = BatchNormalization()(x)
x = Conv2D(32, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), act
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(psize, psize))(x)
x = Dropout(0.2)(x)

x = Conv2D(64, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), act
x = BatchNormalization()(x)
x = Conv2D(64, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), act
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(psize, psize))(x)
x = Dropout(0.25)(x)

x = Conv2D(128, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), ac
x = BatchNormalization()(x)
x = Conv2D(128, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), ac
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(psize, psize))(x)
x = Dropout(0.35)(x)

x = Conv2D(256, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), ac
x = BatchNormalization()(x)
x = Conv2D(256, [3,3], padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), ac
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(psize, psize))(x)
x = Dropout(0.45)(x)

x = Flatten()(x)
x = Dense(800, activation = 'relu', kernel_regularizer=regularizers.l2(weight_decay))(x)
x = Dropout(0.35)(x)
y = Dense(digits, activation = 'softmax')(x)

model = Model(inputs = input, outputs = y)
model.summary()


batch_size = 128
iterations = 50
lr = 0.003


# # Adaptive learning rate code referenced from:
def lr_scheduler(epoch):
    l = lr
    if epoch > 30:
        l = 0.001
    if epoch > 60
    if epoch > 60:
        l = 0.0005
    if epoch > 100:
        l = 0.0001
    return l



# Data augmentation
datagen = ImageDataGenerator(
    rotation_range = 15,
    width_shift_range=4,
    height_shift_range=4,
    horizontal_flip=True,
    fill_mode = 'constant'
    )
datagen.fit(X_train)


model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adamax(learning_rate = lr),
              metrics=['accuracy', 'top_k_categorical_accuracy'])


history = model.fit(datagen.flow(X_train, y_train, batch_size=batch_size),
          steps_per_epoch=X_train.shape[0] // batch_size,
          epochs=iterations,
          verbose=2,
          validation_data= (X_val, y_val),
          callbacks = [LearningRateScheduler(lr_scheduler), EarlyStopping(monitor = 'val_top_
)




score = model.evaluate(X_val, y_val)
print(score[2])
```

```
313/313 [==============================] - 1s 4ms/step - loss: 2.0531 - accuracy: 0.563
0.8406999707221985
```