```
'''
ECE471, Selected Topics in Machine Learning – Assignment 4
Submit by Oct. 4, 10pm
tldr: Classify cifar10. Acheive performance similar to the state of the art.
Classify cifar100. Achieve a top-5 accuracy of 70%

The architecture is inspired by the MaxOut paper:
   http://proceedings.mlr.press/v28/goodfellow13.pdf
Implementation of MaxOut is adapted from:
   https://github.com/philipperemy/tensorflow-maxout/blob/master/maxout.py
My contributions:
   – Added maxpooling, dropout and batchnorm, contrary to original paper
   – Added a dense layer, contrary to original paper
   – Used Adam instead of SGD with annealed learning rate, contrary to
     original paper
'''

import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets.cifar100 import load_data

# Data parameters
NUM_CLASSES = 100
HEIGHT = 32
WIDTH = 32
NUM_CHANNELS = 3

# Load data and split into train, val, test sets
(x_train, y_train), (x_test, y_test) = load_data()
(x_train, y_train), (x_val, y_val) = \
    (x_train[:40000], y_train[:40000]), (x_train[40000:], y_train[40000:])

# Normalize and reshape data and labels
x_train, x_val, x_test = \
    map(lambda x: (x / 255.0).reshape([-1, HEIGHT, WIDTH, NUM_CHANNELS]),
        [x_train, x_val, x_test])
y_train, y_val, y_test = \
    map(lambda y: keras.utils.to_categorical(y, NUM_CLASSES),
        [y_train, y_val, y_test])


# Adapted from
# https://github.com/philipperemy/tensorflow-maxout/blob/master/maxout.py
def maxout(inputs):
    shape = inputs.get_shape().as_list()
    shape[0] = -1
    shape[-1] = shape[-1] // 2
    shape += [2]
    outputs = tf.reduce_max(tf.reshape(inputs, shape), -1, keepdims=False)
    return outputs


def conv_layer(filters, kernel_size, maxpool=False, dropout=False, model=None):
    model.add(keras.layers.Conv2D(
        filters,
        kernel_size,
        padding='same',
        activation=maxout,
        kernel_regularizer=tf.keras.regularizers.l2(5e-7)
    ))
    if maxpool:
        model.add(keras.layers.MaxPool2D(maxpool))
    if dropout:
        model.add(keras.layers.Dropout(0.3))
    model.add(keras.layers.BatchNormalization())


# Hyperparameters
```



```
BATCH_SIZE = 128
NUM_EPOCHS = 15

model = keras.Sequential()

conv_layer(64, 3, model=model)
conv_layer(64, 3, maxpool=2, dropout=True, model=model)
conv_layer(128, 3, model=model)
conv_layer(128, 3, maxpool=2, dropout=True, model=model)
conv_layer(256, 1, model=model)

model.add(keras.layers.GlobalAveragePooling2D())

model.add(keras.layers.Dense(256, activation=keras.activations.relu))
model.add(keras.layers.Dense(NUM_CLASSES,
                             activation=keras.activations.softmax))

# It looks like a high learning rate is key
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(lr=0.01),
              metrics=['accuracy', 'top_k_categorical_accuracy'])

model.fit(x_train, y_train,
          batch_size=BATCH_SIZE,
          epochs=NUM_EPOCHS,
          verbose=1,
          validation_data=[x_val, y_val])

loss, acc, top5_acc = model.evaluate(x_test, y_test, verbose=1)

print('Test loss:', loss)
print('Test accuracy:', acc)
print('Test top5 accuracy:', top5_acc)

''' Output, omitting Keras logs.
Test loss: 2.1856466199874878
Test accuracy: 0.4477
Test top5 accuracy: 0.7614
'''
```