```
#!/bin/python3.5
# Ostap Voynarovskiy
# CGML HW4
# October 4 2018
# Professo Curro
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Activati
on, BatchNormalization, AveragePooling2D, Input
from keras import regularizers
from keras.models import Model
from keras.callbacks import ReduceLROnPlateau

from keras.optimizers import Adam
from keras.regularizers import l2

#from keras import backend as K

num_classes=10
BATCH_SIZE = 32
epochs = 200
DROP_RATE =.3
weight_decay = 1e-4
def genTrainAndVal(f,l): #split the features and labels of the training data 80:
20 train and validation
        lx=f.shape[0]
        z = f.shape[0]
        s = np.arange(z)
        np.random.shuffle(s)
        fs = f[s]                       # features shuffled
        ls = l[s]                       # labels shuffled
        lx = f.shape[0]         # len of the features
        nv = int( lx *.2)       # num validation samp
        print (fs[nv:].shape, ls[nv:].shape, fs[:nv].shape, ls[:nv].shape)
        return fs[nv:], ls[nv:], fs[:nv], ls[:nv]


datagen = ImageDataGenerator(featurewise_center=False,samplewise_center=False,
        featurewise_std_normalization=False,samplewise_std_normalization=False,
        zca_whitening=False,zca_epsilon=1e-06,rotation_range=0,width_shift_range
=0.1,
        height_shift_range=0.1,shear_range=0.,zoom_range=0.,channel_shift_range=
0.,
        fill_mode='nearest',cval=0.,horizontal_flip=True,vertical_flip=False,resca
le=None,
        preprocessing_function=None,data_format=None,validation_split=0.0)

#modified from https://github.com/keras-team/keras/blob/master/examples/cifar10_
resnet.py
def lr_schedule(epoch):
        lr = 1e-3
        if epoch > 150:
                lr *= 0.5e-3
        elif epoch > 120:
                lr *= 1e-3
        elif epoch > 100:
                lr *= 1e-2
        elif epoch > 60:
                lr *= 1e-1
        print('Learning rate: ', lr)
        return lr
```



```
# load cifar 10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
input_shape = x_train.shape[1:]
print(input_shape)

#convert to float and normalize
x_train,x_test = x_train.astype('float32'),x_test.astype('float32')
x_train,x_test = x_train/255,x_test/255

x_t,y_t,x_v,y_v =genTrainAndVal(x_train,y_train)

#print Shapes
print("Training features shape: ", x_t.shape)
print("Validation features shape: ",x_v.shape)
print("Test features shape: ", x_test.shape)

#one hot encode the labels
y_t = keras.utils.to_categorical(y_t,num_classes)
y_v = keras.utils.to_categorical(y_v,num_classes)
y_test  = keras.utils.to_categorical(y_test, num_classes)
def res_layer(inputs, num_filters=16, kernel_size=3,
        strides=1, activation='elu'):

        conv = Conv2D(num_filters,kernel_size=kernel_size,
                strides=strides,padding='same',kernel_initializer='glorot_uniform',
                kernel_regularizer=l2(weight_decay))

        x = inputs
        x1 = Conv2D(num_filters,kernel_size=kernel_size,
                strides=strides,padding='same',kernel_initializer='glorot_uniform',
                kernel_regularizer=l2(weight_decay))(x)
        x = BatchNormalization()(x1)
        x = Activation(activation)(x)
        x = Conv2D(num_filters,kernel_size=kernel_size,
                strides=strides,padding='same',kernel_initializer='glorot_uniform',
                kernel_regularizer=l2(weight_decay))(x)#32,32,16
        x = BatchNormalization()(x)
        x = Activation(activation)(x)
        x = Conv2D(num_filters,kernel_size=kernel_size,
                strides=strides,padding='same',kernel_initializer='glorot_uniform',
                kernel_regularizer=l2(weight_decay))(x)
        x = BatchNormalization()(x)
        x = keras.layers.add([x1,x])
        return x

def resNet(input_shape, num_classes=10):
        num_filters = 16
        print("hi",input_shape)
        inputs = Input(shape=input_shape)
        print(inputs.shape)
        y = res_layer(inputs=inputs)

        for i in range(2):
                y = Activation("elu")(y)
                y = MaxPooling2D(pool_size=2,strides=2)(y)
                num_filters*=2
                y = res_layer(inputs=y, num_filters = num_filters )


        x = Activation("elu")(y)
        x = AveragePooling2D(pool_size=8)(x)
        y = Flatten()(x)
        outputs = Dense(num_classes,activation='softmax',kernel_initializer='he_nor
mal')(y)

        model = Model(inputs=inputs, outputs=outputs)
        return model
```

```
model = resNet(input_shape, num_classes=10)

model.summary()
model.compile(loss="categorical_crossentropy",
        optimizer=keras.optimizers.Adam(lr=lr_schedule(0)),
        metrics=['accuracy'])

datagen.fit(x_t)


lr_scheduler = LearningRateScheduler(lr_schedule)

lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
        cooldown=0,patience=5,min_lr=0.5e-6)

callbacks = [lr_reducer, lr_scheduler]

model.fit_generator(datagen.flow(x_t, y_t, batch_size=BATCH_SIZE),
        validation_data =(x_v,y_v),
        steps_per_epoch=x_t.shape[0]/BATCH_SIZE,
        epochs = epochs,
        verbose = 1,
        callbacks=callbacks)



score = model.evaluate(x_test,y_test,verbose=1)
print("Test loss:",score[0])
print("Test accuracy:",score[1])
'''
I implemented a variation of the resnet20 from the paper https://arxiv.org/pdf/1512.03385.pdf
and pulled some syntax for the callbacks and live data generation from the
keras github implementation.https://github.com/keras-team/keras/blob/master/examples/cifar10_resnet.py
I went through many different variations and found that some things just dont
really make sense. For instance, when i ran this program with 30% dropout,
i lost 2% accuracy on the test set. I realized however that the average pooling
seemed to help. I also attempted a model that didnt use the skip connections. Its hard
to say whether the skip connections helped or not, but they took a long time to figure
out how to implement. The Cifar 100 model used the non skip connection version of the model and it
seemed to work fine although it did have the benefit of top 5 accuracy.
The learning rate drop however did add a lot of accuracy and was proably
the most useful thing. It dropped multiple times and stabilized my model. While
my final model was overfit, it achieved a better accuracy.


'''
```

```
(venv) ostap@ostap-All-Series:~/Documents/DeepLearning/hw4curro$ python CIFAR10R
es.py
Using TensorFlow backend.
(32, 32, 3)
(40000, 32, 32, 3) (40000, 1) (10000, 32, 32, 3) (10000, 1)
Training features shape:  (40000, 32, 32, 3)
Validation features shape:  (10000, 32, 32, 3)
Test features shape:  (10000, 32, 32, 3)
hi (32, 32, 3)
(?, 32, 32, 3)
2018-10-04 00:18:06.742793: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
2018-10-04 00:18:06.819645: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:964] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero
2018-10-04 00:18:06.820032: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
411] Found device 0 with properties:
name: GeForce GTX 980 major: 5 minor: 2 memoryClockRate(GHz): 1.2785
pciBusID: 0000:01:00.0
totalMemory: 3.94GiB freeMemory: 3.05GiB
2018-10-04 00:18:06.870278: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:964] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero
2018-10-04 00:18:06.870668: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
411] Found device 1 with properties:
name: GeForce GTX 970 major: 5 minor: 2 memoryClockRate(GHz): 1.329
pciBusID: 0000:02:00.0
totalMemory: 3.94GiB freeMemory: 3.87GiB
2018-10-04 00:18:06.870818: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
490] Adding visible gpu devices: 0, 1
2018-10-04 00:18:07.225328: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
71] Device interconnect StreamExecutor with strength 1 edge matrix:
2018-10-04 00:18:07.225360: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
77]      0 1
2018-10-04 00:18:07.225365: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
90] 0:   N Y
2018-10-04 00:18:07.225368: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
90] 1:   Y N
2018-10-04 00:18:07.225597: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
103] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 wit
h 2758 MB memory) -> physical GPU (device: 0, name: GeForce GTX 980, pci bus id:
 0000:01:00.0, compute capability: 5.2)
2018-10-04 00:18:07.247495: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
103] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:1 wit
h 3599 MB memory) -> physical GPU (device: 1, name: GeForce GTX 970, pci bus id:
 0000:02:00.0, compute capability: 5.2)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 16)        448
_________________________________________________________________
batch_normalization_1 (BatchNor (None, 32, 32, 16)        64
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 16)        2320
_________________________________________________________________
batch_normalization_2 (BatchNor (None, 32, 32, 16)        64
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 16)        2320
_________________________________________________________________
batch_normalization_3 (BatchNor (None, 32, 32, 16)        64
_________________________________________________________________
add_1 (Add)                  (None, 32, 32, 16)        0
_________________________________________________________________
activation_3 (Activation)    (None, 32, 32, 16)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 32)        4640
_________________________________________________________________
batch_normalization_4 (BatchNor (None, 16, 16, 32)        128
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
batch_normalization_5 (BatchNor (None, 16, 16, 32)        128
_________________________________________________________________
activation_5 (Activation)    (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
batch_normalization_6 (BatchNor (None, 16, 16, 32)        128
_________________________________________________________________
add_2 (Add)                  (None, 16, 16, 32)        0
```



```
conv2d_3 (Conv2D)            (None, 32, 32, 16)        2320
[0]
_________________________________________________________________
_________________
batch_normalization_2 (BatchNor (None, 32, 32, 16)        64
_________________________________________________________________
_________________
activation_2 (Activation)    (None, 32, 32, 16)        0
tion_2[0][0]
_________________________________________________________________
_________________
conv2d_4 (Conv2D)            (None, 32, 32, 16)        2320
[0]
_________________________________________________________________
_________________
batch_normalization_3 (BatchNor (None, 32, 32, 16)        64
_________________________________________________________________
_________________
add_1 (Add)                  (None, 32, 32, 16)        0
tion_3[0][0]
_________________________________________________________________
_________________
activation_3 (Activation)    (None, 32, 32, 16)        0
_________________________________________________________________
_________________
max_pooling2d_1 (MaxPooling2D)  (None, 16, 16, 16)        0
[0]
_________________________________________________________________
_________________
conv2d_6 (Conv2D)            (None, 16, 16, 32)        4640
[0][0]
_________________________________________________________________
_________________
batch_normalization_4 (BatchNor (None, 16, 16, 32)        128
_________________________________________________________________
_________________
activation_4 (Activation)    (None, 16, 16, 32)        0
tion_4[0][0]
_________________________________________________________________
_________________
conv2d_7 (Conv2D)            (None, 16, 16, 32)        9248
[0]
_________________________________________________________________
_________________
batch_normalization_5 (BatchNor (None, 16, 16, 32)        128
_________________________________________________________________
_________________
activation_5 (Activation)    (None, 16, 16, 32)        0
tion_5[0][0]
_________________________________________________________________
_________________
conv2d_8 (Conv2D)            (None, 16, 16, 32)        9248
[0]
_________________________________________________________________
_________________
batch_normalization_6 (BatchNor (None, 16, 16, 32)        128
_________________________________________________________________
_________________
add_2 (Add)                  (None, 16, 16, 32)        0
```

```
Learning rate:  5e-07
1250/1250 [==============================] - 15s 12ms/step - loss: 0.2496 - acc:
 0.9487 - val_loss: 0.4836 - val_acc: 0.8836
Epoch 191/200
Learning rate:  5e-07
1250/1250 [==============================] - 15s 12ms/step - loss: 0.2496 - acc:
 0.9500 - val_loss: 0.4843 - val_acc: 0.8836
Epoch 192/200
Learning rate:  5e-07
1250/1250 [==============================] - 15s 12ms/step - loss: 0.2521 - acc:
 0.9497 - val_loss: 0.4842 - val_acc: 0.8839
Epoch 193/200
Learning rate:  5e-07
1250/1250 [==============================] - 15s 12ms/step - loss: 0.2526 - acc:
 0.9476 - val_loss: 0.4828 - val_acc: 0.8841
Epoch 194/200
Learning rate:  5e-07
1250/1250 [==============================] - 15s 12ms/step - loss: 0.2504 - acc:
 0.9491 - val_loss: 0.4828 - val_acc: 0.8843
Epoch 195/200
Learning rate:  5e-07
1250/1250 [==============================] - 14s 12ms/step - loss: 0.2496 - acc:
 0.9490 - val_loss: 0.4835 - val_acc: 0.8841
Epoch 196/200
Learning rate:  5e-07
1250/1250 [==============================] - 14s 12ms/step - loss: 0.2510 - acc:
 0.9486 - val_loss: 0.4830 - val_acc: 0.8841
Epoch 197/200
Learning rate:  5e-07
1250/1250 [==============================] - 15s 12ms/step - loss: 0.2524 - acc:
 0.9494 - val_loss: 0.4837 - val_acc: 0.8842
Epoch 198/200
Learning rate:  5e-07
1250/1250 [==============================] - 15s 12ms/step - loss: 0.2489 - acc:
 0.9492 - val_loss: 0.4836 - val_acc: 0.8839
Epoch 199/200
Learning rate:  5e-07
1250/1250 [==============================] - 15s 12ms/step - loss: 0.2506 - acc:
 0.9485 - val_loss: 0.4855 - val_acc: 0.8829
Epoch 200/200
Learning rate:  5e-07
1250/1250 [==============================] - 14s 12ms/step - loss: 0.2501 - acc:
 0.9488 - val_loss: 0.4845 - val_acc: 0.8831
10000/10000 [==============================] - 1s 105us/step
Test loss: 0.5143441814422608
Test accuracy: 0.8784
```

```
#!/bin/python3.5
# Ostap Voynarovskiy
# CGML HW4
# October 4 2018
# Professo Curro

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.datasets import cifar10
from keras.layers import Dense, Dropout, Flatten,Conv2D, MaxPooling2D, Activatio
n, BatchNormalization
from keras import regularizers
from keras.metrics import top_k_categorical_accuracy
#from keras import backend as K

num_classes=100
BATCH_SIZE = 32
epochs = 32 #we achieve overfitting after like 15-20 epochs
DROP_RATE =.5
weight_decay = 1e-4

def genTrainAndVal(f,l): #split the features and labels of the training data 80:
20 train and validation
        lx=f.shape[0]
        z = f.shape[0]
        s = np.arange(z)
        np.random.shuffle(s)
        fs = f[s]                       # features shuffled
        ls = l[s]                       # labels shuffled
        lx = f.shape[0]         # len of the features
        nv = int( lx *.2)       # num validation samp
        print (fs[nv:].shape, ls[nv:].shape, fs[:nv].shape, ls[:nv].shape)
        return fs[nv:], ls[nv:], fs[:nv], ls[:nv]


# load cifar 10
#(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# load cifar 100
from keras.datasets import cifar100
(x_train, y_train), (x_test, y_test) = cifar100.load_data(label_mode='fine')

#fonvert to float and normalize
x_train,x_test = x_train.astype('float32'),x_test.astype('float32')
x_train,x_test = x_train/255,x_test/255

x_t,y_t,x_v,y_v =genTrainAndVal(x_train,y_train)

#print Shapes
print("Training features shape: ", x_t.shape)
print("Validation features shape: ",x_v.shape)
print("Test features shape: ", x_test.shape)

#one hot encode the labels
y_t = keras.utils.to_categorical(y_t,num_classes)
y_v = keras.utils.to_categorical(y_v,num_classes)
y_test  = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()

model.add(Conv2D(32,(4,4),padding='same',kernel_regularizer=regularizers.l2(weig
ht_decay), data_format='channels_last', kernel_initializer='glorot_uniform', input_shap
e=x_t[0].shape))
model.add(Activation("elu"))
model.add(BatchNormalization())
model.add(Conv2D(32,(3,3),strides=1,padding='same', kernel_regularizer=regulariz
```



```
ers.l2(weight_decay), kernel_initializer='glorot_uniform'))
model.add(Activation("elu"))

model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(DROP_RATE))

model.add(Conv2D(64,kernel_size=(3,3),padding='same',kernel_regularizer=regulari
zers.l2(weight_decay),data_format='channels_last',kernel_initializer='glorot_uniform'))
model.add(Activation("elu"))
model.add(BatchNormalization())
model.add(Conv2D(64,kernel_size=(4,4),padding='same',kernel_regularizer=regulari
zers.l2(weight_decay),data_format='channels_last',kernel_initializer='glorot_uniform'))
model.add(Activation("elu"))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2,2), strides=2))
model.add(Dropout(DROP_RATE))

model.add(Conv2D(128,kernel_size=(5,5),padding='same',kernel_regularizer=regular
izers.l2(weight_decay),data_format='channels_last',kernel_initializer='glorot_uniform')
)
model.add(Activation("elu"))
model.add(BatchNormalization())
model.add(Conv2D(128,kernel_size=(2,2),padding='same',kernel_regularizer=regular
izers.l2(weight_decay),data_format='channels_last',kernel_initializer='glorot_uniform')
)
model.add(Activation("elu"))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2,2), strides=2))
model.add(Dropout(.2))

model.add(Flatten())
model.add(Dense(num_classes,activation="softmax"))

model.summary()



model.compile(loss=keras.losses.categorical_crossentropy,
        optimizer=keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsi
lon=None, decay=0.0, amsgrad=False),
        metrics=["top_k_categorical_accuracy"])

model.fit(x_t, y_t,
        batch_size=BATCH_SIZE,
        epochs = epochs,
        verbose = 1,
        validation_data =(x_v,y_v))

score = model.evaluate(x_test,y_test,verbose= 1)
print("Test loss:",score[0])
print("Test accuracy:",score[1])
```

```
(venv) ostap@ostap-All-Series:~/Documents/DeepLearning/hw4curro$ python CIFAR100
.py
Using TensorFlow backend.
(40000, 32, 32, 3) (40000, 1) (10000, 32, 32, 3) (10000, 1)
Training features shape:  (40000, 32, 32, 3)
Validation features shape:  (10000, 32, 32, 3)
Test features shape:  (10000, 32, 32, 3)
2018-10-03 18:52:53.455261: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
2018-10-03 18:52:53.548204: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:964] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero
2018-10-03 18:52:53.548624: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
411] Found device 0 with properties:
name: GeForce GTX 980 major: 5 minor: 2 memoryClockRate(GHz): 1.2785
pciBusID: 0000:01:00.0
totalMemory: 3.94GiB freeMemory: 3.04GiB
2018-10-03 18:52:53.614475: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:964] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero
2018-10-03 18:52:53.614870: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
411] Found device 1 with properties:
name: GeForce GTX 970 major: 5 minor: 2 memoryClockRate(GHz): 1.329
pciBusID: 0000:02:00.0
totalMemory: 3.94GiB freeMemory: 3.87GiB
2018-10-03 18:52:53.615025: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
490] Adding visible gpu devices: 0, 1
2018-10-03 18:52:53.967614: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
71] Device interconnect StreamExecutor with strength 1 edge matrix:
2018-10-03 18:52:53.967645: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
77]      0 1
2018-10-03 18:52:53.967650: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
90] 0:   N Y
2018-10-03 18:52:53.967653: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
90] 1:   Y N
2018-10-03 18:52:53.967872: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
103] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 wit
h 2751 MB memory) -> physical GPU (device: 0, name: GeForce GTX 980, pci bus id:
 0000:01:00.0, compute capability: 5.2)
2018-10-03 18:52:53.989668: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
103] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:1 wit
h 3599 MB memory) -> physical GPU (device: 1, name: GeForce GTX 970, pci bus id:
 0000:02:00.0, compute capability: 5.2)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        1568
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        65600
_________________________________________________________________
```



```
activation_4 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         204928
_________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         65664
_________________________________________________________________
activation_6 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               204900
=================================================================
Total params: 572,068
Trainable params: 571,236
Non-trainable params: 832
_________________________________________________________________
Train on 40000 samples, validate on 10000 samples
Epoch 1/32
40000/40000 [==============================] - 16s 409us/step - loss: 4.1911 - t
op_k_categorical_accuracy: 0.3479 - val_loss: 3.3574 - val_top_k_categorical_acc
uracy: 0.4923
Epoch 2/32
40000/40000 [==============================] - 15s 376us/step - loss: 3.1132 - t
op_k_categorical_accuracy: 0.5538 - val_loss: 2.8131 - val_top_k_categorical_acc
uracy: 0.6225
Epoch 3/32
40000/40000 [==============================] - 15s 375us/step - loss: 2.6354 - t
op_k_categorical_accuracy: 0.6573 - val_loss: 2.5696 - val_top_k_categorical_acc
uracy: 0.6761
Epoch 4/32
40000/40000 [==============================] - 15s 376us/step - loss: 2.3939 - t
op_k_categorical_accuracy: 0.7151 - val_loss: 2.3879 - val_top_k_categorical_acc
uracy: 0.7168
Epoch 5/32
40000/40000 [==============================] - 15s 375us/step - loss: 2.2350 - t
op_k_categorical_accuracy: 0.7537 - val_loss: 2.3999 - val_top_k_categorical_acc
uracy: 0.7222
Epoch 6/32
40000/40000 [==============================] - 15s 376us/step - loss: 2.1149 - t
op_k_categorical_accuracy: 0.7805 - val_loss: 2.2855 - val_top_k_categorical_acc
uracy: 0.7483
Epoch 7/32
40000/40000 [==============================] - 15s 376us/step - loss: 2.0224 - t
op_k_categorical_accuracy: 0.8023 - val_loss: 2.3183 - val_top_k_categorical_acc
uracy: 0.7565
Epoch 8/32
40000/40000 [==============================] - 15s 375us/step - loss: 1.9339 - t
op_k_categorical_accuracy: 0.8208 - val_loss: 2.2308 - val_top_k_categorical_acc
uracy: 0.7750
Epoch 9/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.8632 - t
op_k_categorical_accuracy: 0.8364 - val_loss: 2.3028 - val_top_k_categorical_acc
uracy: 0.7636
```

```
Epoch 10/32
40000/40000 [==============================] - 15s 375us/step - loss: 1.8067 - t
op_k_categorical_accuracy: 0.8467 - val_loss: 2.2427 - val_top_k_categorical_acc
uracy: 0.7786
Epoch 11/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.7510 - t
op_k_categorical_accuracy: 0.8609 - val_loss: 2.1793 - val_top_k_categorical_acc
uracy: 0.7912
Epoch 12/32
40000/40000 [==============================] - 15s 375us/step - loss: 1.7062 - t
op_k_categorical_accuracy: 0.8696 - val_loss: 2.2404 - val_top_k_categorical_acc
uracy: 0.7853
Epoch 13/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.6602 - t
op_k_categorical_accuracy: 0.8784 - val_loss: 2.1732 - val_top_k_categorical_acc
uracy: 0.8046
Epoch 14/32
40000/40000 [==============================] - 15s 375us/step - loss: 1.6301 - t
op_k_categorical_accuracy: 0.8872 - val_loss: 2.2326 - val_top_k_categorical_acc
uracy: 0.7956
Epoch 15/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.5979 - t
op_k_categorical_accuracy: 0.8934 - val_loss: 2.3233 - val_top_k_categorical_acc
uracy: 0.7844
Epoch 16/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.5764 - t
op_k_categorical_accuracy: 0.8984 - val_loss: 2.2007 - val_top_k_categorical_acc
uracy: 0.8040
Epoch 17/32
40000/40000 [==============================] - 15s 375us/step - loss: 1.5455 - t
op_k_categorical_accuracy: 0.9033 - val_loss: 2.3343 - val_top_k_categorical_acc
uracy: 0.7897
Epoch 18/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.5250 - t
op_k_categorical_accuracy: 0.9079 - val_loss: 2.2597 - val_top_k_categorical_acc
uracy: 0.8064
Epoch 19/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.4950 - t
op_k_categorical_accuracy: 0.9141 - val_loss: 2.2843 - val_top_k_categorical_acc
uracy: 0.8049
Epoch 20/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.4831 - t
op_k_categorical_accuracy: 0.9169 - val_loss: 2.3495 - val_top_k_categorical_acc
uracy: 0.7995
Epoch 21/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.4702 - t
op_k_categorical_accuracy: 0.9209 - val_loss: 2.3060 - val_top_k_categorical_acc
uracy: 0.8093
Epoch 22/32
40000/40000 [==============================] - 15s 375us/step - loss: 1.4421 - t
op_k_categorical_accuracy: 0.9256 - val_loss: 2.3839 - val_top_k_categorical_acc
uracy: 0.7979
Epoch 23/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.4391 - t
op_k_categorical_accuracy: 0.9255 - val_loss: 2.3491 - val_top_k_categorical_acc
uracy: 0.8038
Epoch 24/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.4216 - t
op_k_categorical_accuracy: 0.9286 - val_loss: 2.3787 - val_top_k_categorical_acc
uracy: 0.7953
Epoch 25/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.4010 - t
op_k_categorical_accuracy: 0.9313 - val_loss: 2.3989 - val_top_k_categorical_acc
uracy: 0.7959
Epoch 26/32
40000/40000 [==============================] - 15s 375us/step - loss: 1.3877 - t
op_k_categorical_accuracy: 0.9346 - val_loss: 2.3856 - val_top_k_categorical_acc
uracy: 0.8006
Epoch 27/32
```



```
40000/40000 [==============================] - 15s 377us/step - loss: 1.3773 - t
op_k_categorical_accuracy: 0.9373 - val_loss: 2.3777 - val_top_k_categorical_acc
uracy: 0.8066
Epoch 28/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.3772 - t
op_k_categorical_accuracy: 0.9367 - val_loss: 2.4399 - val_top_k_categorical_acc
uracy: 0.8018
Epoch 29/32
40000/40000 [==============================] - 15s 377us/step - loss: 1.3589 - t
op_k_categorical_accuracy: 0.9395 - val_loss: 2.4524 - val_top_k_categorical_acc
uracy: 0.7953
Epoch 30/32
40000/40000 [==============================] - 15s 375us/step - loss: 1.3470 - t
op_k_categorical_accuracy: 0.9427 - val_loss: 2.4399 - val_top_k_categorical_acc
uracy: 0.8001
Epoch 31/32
40000/40000 [==============================] - 15s 377us/step - loss: 1.3399 - t
op_k_categorical_accuracy: 0.9427 - val_loss: 2.4711 - val_top_k_categorical_acc
uracy: 0.7989
Epoch 32/32
40000/40000 [==============================] - 15s 376us/step - loss: 1.3318 - t
op_k_categorical_accuracy: 0.9438 - val_loss: 2.5110 - val_top_k_categorical_acc
uracy: 0.7980
10000/10000 [==============================] - 1s 117us/step
Test loss: 2.4556733959197996
Test accuracy: 0.796
```