```python
#!/usr/bin/env python

# ECE472-Samuel Maltz
# Assignment 4: Classification of CIFAR10 and CIFAR100 datasets using
# convolutional neural networks

# As a first attempt at classifying the CIFAR10 dataset, the model used to
# classify the MNIST data was reused with CNN layers with 32, 64, 128 and 256
# filters followed by dense layers with widths of 1024, 512, 256, 128 and 10.
# The learning rate and L2 kernel regularization coefficient were 0.001 and
# dropout between dense layers was 20%. This initial attempt achieved an
# accuracy of 72%. Next, by running through different values for the learning
# rate, kernel regularizer coefficient and dropout it was determined that the
# best values were 0.001, 0.0005 and 0.3 respectively. This raised the
# accuracy to 76%. After this batch normalization was experimented between
# layers and it was determined it was best for only the CNN layers.
# Additionally, dropout was experimented on the CNN layers and was found to
# improve performance as well. These changes raised the accuracy to 81%.
# Finally the amount of convolutional filters and dense widths were
# experimented on and it was determined that doubling the filters in all
# convolutional layers to 64, 128, 256 and 512 and actually removing all dense
# layers besides for the last layer produced the best results. These results
# can be found in the results10.txt file and it can be seen that the model
# achieves an accuracy of 87.35% on the test dataset.
#
# With regards to the CIFAR100 dataset, the same model used on the CIFAR10
# dataset was attempted first. Afterwards, different parameters were varied as
# in the CIFAR10 dataset; however, it turned out that most of the settings
# were optimal for this model structure except that an additional dense layer
# with width 1024 is added. The results can be found in results100.txt which
# show that this model achieves an accuracy of 86.16% on the test dataset,
# unfortunately lower than the 90% goal.
#
# The dataset and unpickle function comes from:
# Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

import tensorflow as tf
import numpy as np
import pickle

from absl import app
from absl import flags

FLAGS = flags.FLAGS
flags.DEFINE_bool(
    "cifar100",
    False,
    "Whether to use the CIFAR-100 dataset instead of the CIFAR-10 dataset",
)
flags.DEFINE_string(
    "cifar10_dir", "cifar-10-batches-py", "Name of directory with CIFAR10 dataset"
)
flags.DEFINE_string(
    "cifar100_dir", "cifar-100-python", "Name of directory with CIFAR100 dataset"
)
flags.DEFINE_list(
    "conv_filters", [64, 128, 256, 512], "Number of filters of convolutional layers"
)
flags.DEFINE_integer(
    "conv_per_pool", 2, "Number of convolutional layers per pooling layer"
)
flags.DEFINE_integer("pool_size", 2, "Window size of max pool")
```



```python
flags.DEFINE_list("dense_widths", [], "Widths of dense layers")
flags.DEFINE_float("dropout", 0.3, "Dropout rate")
flags.DEFINE_float("learning_rate", 0.0005, "Learning rate for Adam optimizer")
flags.DEFINE_integer("epochs", 50, "Number of training epochs")
flags.DEFINE_float("val_split", 0.1, "Validation fraction")
flags.DEFINE_float("kernel_reg", 0.001, "Regularizer coefficient")
flags.DEFINE_integer("random_seed", 12345, "Random seed")


class Data(object):
    def __init__(self, cifar_dir, cifar100):
        if cifar100:
            data = self.unpickle(cifar_dir + "/train")
            self.train_images = self.preprocess_images(data[b"data"])
            self.train_labels = self.preprocess_labels(data[b"fine_labels"])

            data = self.unpickle(cifar_dir + "/test")
            self.test_images = self.preprocess_images(data[b"data"])
            self.test_labels = self.preprocess_labels(data[b"fine_labels"])
        else:
            self.train_images = np.array([]).reshape(0, 32, 32, 3)
            self.train_labels = np.array([])
            for i in range(1, 6):
                data = self.unpickle(cifar_dir + "/data_batch_" + str(i))
                self.train_images = np.concatenate(
                    (self.train_images, self.preprocess_images(data[b"data"]))
                )
                self.train_labels = np.concatenate(
                    (self.train_labels, self.preprocess_labels(data[b"labels"]))
                )

            data = self.unpickle(cifar_dir + "/test_batch")
            self.test_images = self.preprocess_images(data[b"data"])
            self.test_labels = self.preprocess_labels(data[b"labels"])

    def unpickle(self, file):
        with open(file, "rb") as fo:
            dict = pickle.load(fo, encoding="bytes")
        return dict

    def preprocess_images(self, images):
        return np.transpose(np.reshape(images, (-1, 3, 32, 32)), (0, 2, 3, 1)).a
stype(
            "float32"
        )

    def preprocess_labels(self, labels):
        return np.array(labels).astype("float32")


class Model(tf.keras.Model):
    def __init__(
        self,
        conv_filters,
        conv_per_pool,
        pool_size,
        dense_widths,
        dropout,
        kernel_reg,
        num_categories,
    ):
        super().__init__()
```

```
        self.regularizer = tf.keras.regularizers.L2(kernel_reg)

        # Convolution block
        self.conv = [
            {
                "conv": [
                    {
                        "conv": tf.keras.layers.Conv2D(i, 3, padding="same"),
                        "batchnorm": tf.keras.layers.BatchNormalization(),
                        "relu": tf.keras.layers.ReLU(),
                    }
                    for j in range(conv_per_pool)
                ],
                "maxpool": tf.keras.layers.MaxPool2D(pool_size),
                "dropout": tf.keras.layers.Dropout(dropout),
            }
            for i in conv_filters
        ]
        self.flatten = tf.keras.layers.Flatten()

        # Dense block
        self.dense = [
            {
                "dense": tf.keras.layers.Dense(i, kernel_regularizer=self.regular
izer),
                "relu": tf.keras.layers.ReLU(),
                "dropout": tf.keras.layers.Dropout(dropout),
            }
            for i in dense_widths
        ]
        self.final_dense = tf.keras.layers.Dense(
            num_categories, activation="softmax", kernel_regularizer=self.regular
izer
        )

    def call(self, x, training=False):
        for conv_block in self.conv:
            for conv_layer in conv_block["conv"]:
                x = conv_layer["conv"](x)
                x = conv_layer["batchnorm"](x)
                x = conv_layer["relu"](x)

            x = conv_block["maxpool"](x)
            x = conv_block["dropout"](x)

        x = self.flatten(x)
        for dense_layer in self.dense:
            x = dense_layer["dense"](x)
            x = dense_layer["relu"](x)
            if training:
                x = dense_layer["dropout"](x)

        return self.final_dense(x)


def main(a):
    tf.random.set_seed(FLAGS.random_seed)

    FLAGS.conv_filters = list(map(int, FLAGS.conv_filters))
    FLAGS.dense_widths = list(map(int, FLAGS.dense_widths))

    if FLAGS.cifar100:
```



```
        num_categories = 100
        cifar_dir = FLAGS.cifar100_dir
        k = 5  # for top k accuracy
    else:
        num_categories = 10
        cifar_dir = FLAGS.cifar10_dir
        k = 1

    data = Data(cifar_dir, FLAGS.cifar100)
    model = Model(
        FLAGS.conv_filters,
        FLAGS.conv_per_pool,
        FLAGS.pool_size,
        FLAGS.dense_widths,
        FLAGS.dropout,
        FLAGS.kernel_reg,
        num_categories,
    )

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=FLAGS.learning_rate),
        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
        metrics=tf.keras.metrics.SparseTopKCategoricalAccuracy(k),
    )

    callback = tf.keras.callbacks.EarlyStopping(
        monitor="val_sparse_top_k_categorical_accuracy",
        patience=3,
        restore_best_weights=True,
    )

    model.fit(
        data.train_images,
        data.train_labels,
        epochs=FLAGS.epochs,
        callbacks=[callback],
        verbose=2,
        validation_split=FLAGS.val_split,
    )

    model.summary()

    model.evaluate(data.test_images, data.test_labels, verbose=2)


if __name__ == "__main__":
    app.run(main)
```

```
Epoch 1/50
1407/1407 - 434s - loss: 1.5445 - sparse_top_k_categorical_accuracy: 0.4825 - va
l_loss: 1.8410 - val_sparse_top_k_categorical_accuracy: 0.4762
Epoch 2/50
1407/1407 - 510s - loss: 0.9631 - sparse_top_k_categorical_accuracy: 0.6706 - va
l_loss: 0.9283 - val_sparse_top_k_categorical_accuracy: 0.6896
Epoch 3/50
1407/1407 - 524s - loss: 0.7738 - sparse_top_k_categorical_accuracy: 0.7356 - va
l_loss: 0.7658 - val_sparse_top_k_categorical_accuracy: 0.7384
Epoch 4/50
1407/1407 - 532s - loss: 0.6607 - sparse_top_k_categorical_accuracy: 0.7753 - va
l_loss: 0.6290 - val_sparse_top_k_categorical_accuracy: 0.7896
Epoch 5/50
1407/1407 - 533s - loss: 0.5831 - sparse_top_k_categorical_accuracy: 0.8044 - va
l_loss: 0.6516 - val_sparse_top_k_categorical_accuracy: 0.7900
Epoch 6/50
1407/1407 - 536s - loss: 0.5144 - sparse_top_k_categorical_accuracy: 0.8275 - va
l_loss: 0.5149 - val_sparse_top_k_categorical_accuracy: 0.8294
Epoch 7/50
1407/1407 - 536s - loss: 0.4636 - sparse_top_k_categorical_accuracy: 0.8450 - va
l_loss: 0.6845 - val_sparse_top_k_categorical_accuracy: 0.7904
Epoch 8/50
1407/1407 - 538s - loss: 0.4141 - sparse_top_k_categorical_accuracy: 0.8628 - va
l_loss: 0.5380 - val_sparse_top_k_categorical_accuracy: 0.8248
Epoch 9/50
1407/1407 - 544s - loss: 0.3739 - sparse_top_k_categorical_accuracy: 0.8761 - va
l_loss: 0.4915 - val_sparse_top_k_categorical_accuracy: 0.8450
Epoch 10/50
1407/1407 - 537s - loss: 0.3372 - sparse_top_k_categorical_accuracy: 0.8901 - va
l_loss: 0.4638 - val_sparse_top_k_categorical_accuracy: 0.8504
Epoch 11/50
1407/1407 - 538s - loss: 0.2998 - sparse_top_k_categorical_accuracy: 0.9025 - va
l_loss: 0.5199 - val_sparse_top_k_categorical_accuracy: 0.8432
Epoch 12/50
1407/1407 - 536s - loss: 0.2720 - sparse_top_k_categorical_accuracy: 0.9115 - va
l_loss: 0.5314 - val_sparse_top_k_categorical_accuracy: 0.8406
Epoch 13/50
1407/1407 - 539s - loss: 0.2461 - sparse_top_k_categorical_accuracy: 0.9223 - va
l_loss: 0.4978 - val_sparse_top_k_categorical_accuracy: 0.8514
Epoch 14/50
1407/1407 - 538s - loss: 0.2236 - sparse_top_k_categorical_accuracy: 0.9285 - va
l_loss: 0.4823 - val_sparse_top_k_categorical_accuracy: 0.8640
Epoch 15/50
1407/1407 - 648s - loss: 0.2023 - sparse_top_k_categorical_accuracy: 0.9360 - va
l_loss: 0.4991 - val_sparse_top_k_categorical_accuracy: 0.8602
Epoch 16/50
1407/1407 - 523s - loss: 0.1873 - sparse_top_k_categorical_accuracy: 0.9422 - va
l_loss: 0.5096 - val_sparse_top_k_categorical_accuracy: 0.8630
Epoch 17/50
1407/1407 - 539s - loss: 0.1675 - sparse_top_k_categorical_accuracy: 0.9499 - va
l_loss: 0.5199 - val_sparse_top_k_categorical_accuracy: 0.8652
Epoch 18/50
1407/1407 - 539s - loss: 0.1590 - sparse_top_k_categorical_accuracy: 0.9521 - va
l_loss: 0.5378 - val_sparse_top_k_categorical_accuracy: 0.8684
Epoch 19/50
1407/1407 - 539s - loss: 0.1461 - sparse_top_k_categorical_accuracy: 0.9581 - va
l_loss: 0.5377 - val_sparse_top_k_categorical_accuracy: 0.8638
Epoch 20/50
1407/1407 - 541s - loss: 0.1358 - sparse_top_k_categorical_accuracy: 0.9610 - va
l_loss: 0.4795 - val_sparse_top_k_categorical_accuracy: 0.8742
Epoch 21/50
1407/1407 - 539s - loss: 0.1302 - sparse_top_k_categorical_accuracy: 0.9640 - va
```



```
l_loss: 0.5363 - val_sparse_top_k_categorical_accuracy: 0.8656
Epoch 22/50
1407/1407 - 540s - loss: 0.1169 - sparse_top_k_categorical_accuracy: 0.9681 - va
l_loss: 0.4846 - val_sparse_top_k_categorical_accuracy: 0.8792
Epoch 23/50
1407/1407 - 560s - loss: 0.1167 - sparse_top_k_categorical_accuracy: 0.9679 - va
l_loss: 0.5688 - val_sparse_top_k_categorical_accuracy: 0.8552
Epoch 24/50
1407/1407 - 543s - loss: 0.1134 - sparse_top_k_categorical_accuracy: 0.9706 - va
l_loss: 0.5152 - val_sparse_top_k_categorical_accuracy: 0.8674
Epoch 25/50
1407/1407 - 541s - loss: 0.1088 - sparse_top_k_categorical_accuracy: 0.9709 - va
l_loss: 0.5456 - val_sparse_top_k_categorical_accuracy: 0.8664
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization (BatchNo multiple                  256
_________________________________________________________________
conv2d (Conv2D)              multiple                  1792
_________________________________________________________________
re_lu (ReLU)                 multiple                  0
_________________________________________________________________
batch_normalization_1 (Batch multiple                  256
_________________________________________________________________
conv2d_1 (Conv2D)            multiple                  36928
_________________________________________________________________
re_lu_1 (ReLU)               multiple                  0
_________________________________________________________________
dropout (Dropout)            multiple                  0
_________________________________________________________________
max_pooling2d (MaxPooling2D) multiple                  0
_________________________________________________________________
batch_normalization_2 (Batch multiple                  512
_________________________________________________________________
conv2d_2 (Conv2D)            multiple                  73856
_________________________________________________________________
re_lu_2 (ReLU)               multiple                  0
_________________________________________________________________
batch_normalization_3 (Batch multiple                  512
_________________________________________________________________
conv2d_3 (Conv2D)            multiple                  147584
_________________________________________________________________
re_lu_3 (ReLU)               multiple                  0
_________________________________________________________________
dropout_1 (Dropout)          multiple                  0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 multiple                  0
_________________________________________________________________
batch_normalization_4 (Batch multiple                  1024
_________________________________________________________________
conv2d_4 (Conv2D)            multiple                  295168
_________________________________________________________________
re_lu_4 (ReLU)               multiple                  0
_________________________________________________________________
batch_normalization_5 (Batch multiple                  1024
_________________________________________________________________
conv2d_5 (Conv2D)            multiple                  590080
_________________________________________________________________
re_lu_5 (ReLU)               multiple                  0
_________________________________________________________________
dropout_2 (Dropout)          multiple                  0
```

```
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 multiple                  0
_________________________________________________________________
batch_normalization_6 (Batch multiple                  2048
_________________________________________________________________
conv2d_6 (Conv2D)            multiple                  1180160
_________________________________________________________________
re_lu_6 (ReLU)               multiple                  0
_________________________________________________________________
batch_normalization_7 (Batch multiple                  2048
_________________________________________________________________
conv2d_7 (Conv2D)            multiple                  2359808
_________________________________________________________________
re_lu_7 (ReLU)               multiple                  0
_________________________________________________________________
dropout_3 (Dropout)          multiple                  0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 multiple                  0
_________________________________________________________________
flatten (Flatten)            multiple                  0
_________________________________________________________________
dense (Dense)                multiple                  20490
=================================================================
Total params: 4,713,546
Trainable params: 4,709,706
Non-trainable params: 3,840
_________________________________________________________________
313/313 - 22s - loss: 0.5073 - sparse_top_k_categorical_accuracy: 0.8735
```

```
Epoch 1/50
1407/1407 - 558s - loss: 4.8463 - sparse_top_k_categorical_accuracy: 0.2209 - va
l_loss: 4.1047 - val_sparse_top_k_categorical_accuracy: 0.3274
Epoch 2/50
1407/1407 - 550s - loss: 3.6963 - sparse_top_k_categorical_accuracy: 0.4058 - va
l_loss: 3.3439 - val_sparse_top_k_categorical_accuracy: 0.4820
Epoch 3/50
1407/1407 - 548s - loss: 3.1793 - sparse_top_k_categorical_accuracy: 0.5360 - va
l_loss: 2.8671 - val_sparse_top_k_categorical_accuracy: 0.6138
Epoch 4/50
1407/1407 - 547s - loss: 2.8652 - sparse_top_k_categorical_accuracy: 0.6166 - va
l_loss: 2.7123 - val_sparse_top_k_categorical_accuracy: 0.6562
Epoch 5/50
1407/1407 - 548s - loss: 2.6498 - sparse_top_k_categorical_accuracy: 0.6670 - va
l_loss: 2.6213 - val_sparse_top_k_categorical_accuracy: 0.6736
Epoch 6/50
1407/1407 - 553s - loss: 2.4943 - sparse_top_k_categorical_accuracy: 0.7024 - va
l_loss: 2.3515 - val_sparse_top_k_categorical_accuracy: 0.7336
Epoch 7/50
1407/1407 - 549s - loss: 2.3380 - sparse_top_k_categorical_accuracy: 0.7357 - va
l_loss: 2.2155 - val_sparse_top_k_categorical_accuracy: 0.7600
Epoch 8/50
1407/1407 - 547s - loss: 2.2271 - sparse_top_k_categorical_accuracy: 0.7620 - va
l_loss: 2.0761 - val_sparse_top_k_categorical_accuracy: 0.7920
Epoch 9/50
1407/1407 - 547s - loss: 2.1153 - sparse_top_k_categorical_accuracy: 0.7832 - va
l_loss: 2.0784 - val_sparse_top_k_categorical_accuracy: 0.7886
Epoch 10/50
1407/1407 - 546s - loss: 2.0237 - sparse_top_k_categorical_accuracy: 0.8001 - va
l_loss: 2.0184 - val_sparse_top_k_categorical_accuracy: 0.8006
Epoch 11/50
1407/1407 - 548s - loss: 1.9421 - sparse_top_k_categorical_accuracy: 0.8153 - va
l_loss: 1.9437 - val_sparse_top_k_categorical_accuracy: 0.8230
Epoch 12/50
1407/1407 - 548s - loss: 1.8569 - sparse_top_k_categorical_accuracy: 0.8323 - va
l_loss: 1.8504 - val_sparse_top_k_categorical_accuracy: 0.8284
Epoch 13/50
1407/1407 - 552s - loss: 1.7854 - sparse_top_k_categorical_accuracy: 0.8448 - va
l_loss: 1.9561 - val_sparse_top_k_categorical_accuracy: 0.8136
Epoch 14/50
1407/1407 - 547s - loss: 1.7126 - sparse_top_k_categorical_accuracy: 0.8579 - va
l_loss: 1.8940 - val_sparse_top_k_categorical_accuracy: 0.8240
Epoch 15/50
1407/1407 - 539s - loss: 1.6422 - sparse_top_k_categorical_accuracy: 0.8694 - va
l_loss: 1.8467 - val_sparse_top_k_categorical_accuracy: 0.8358
Epoch 16/50
1407/1407 - 503s - loss: 1.5792 - sparse_top_k_categorical_accuracy: 0.8811 - va
l_loss: 1.8459 - val_sparse_top_k_categorical_accuracy: 0.8374
Epoch 17/50
1407/1407 - 545s - loss: 1.5125 - sparse_top_k_categorical_accuracy: 0.8911 - va
l_loss: 1.8398 - val_sparse_top_k_categorical_accuracy: 0.8432
Epoch 18/50
1407/1407 - 543s - loss: 1.4519 - sparse_top_k_categorical_accuracy: 0.9008 - va
l_loss: 1.7931 - val_sparse_top_k_categorical_accuracy: 0.8530
Epoch 19/50
1407/1407 - 516s - loss: 1.3937 - sparse_top_k_categorical_accuracy: 0.9092 - va
l_loss: 1.8359 - val_sparse_top_k_categorical_accuracy: 0.8434
Epoch 20/50
1407/1407 - 543s - loss: 1.3385 - sparse_top_k_categorical_accuracy: 0.9188 - va
l_loss: 1.8708 - val_sparse_top_k_categorical_accuracy: 0.8448
Epoch 21/50
1407/1407 - 546s - loss: 1.2860 - sparse_top_k_categorical_accuracy: 0.9246 - va
```



```
l_loss: 1.7184 - val_sparse_top_k_categorical_accuracy: 0.8632
Epoch 22/50
1407/1407 - 544s - loss: 1.2433 - sparse_top_k_categorical_accuracy: 0.9302 - va
l_loss: 1.7458 - val_sparse_top_k_categorical_accuracy: 0.8592
Epoch 23/50
1407/1407 - 544s - loss: 1.1948 - sparse_top_k_categorical_accuracy: 0.9374 - va
l_loss: 1.7774 - val_sparse_top_k_categorical_accuracy: 0.8574
Epoch 24/50
1407/1407 - 545s - loss: 1.1464 - sparse_top_k_categorical_accuracy: 0.9439 - va
l_loss: 1.6832 - val_sparse_top_k_categorical_accuracy: 0.8678
Epoch 25/50
1407/1407 - 538s - loss: 1.1138 - sparse_top_k_categorical_accuracy: 0.9482 - va
l_loss: 1.7822 - val_sparse_top_k_categorical_accuracy: 0.8632
Epoch 26/50
1407/1407 - 538s - loss: 1.0718 - sparse_top_k_categorical_accuracy: 0.9513 - va
l_loss: 1.8786 - val_sparse_top_k_categorical_accuracy: 0.8488
Epoch 27/50
1407/1407 - 560s - loss: 1.0290 - sparse_top_k_categorical_accuracy: 0.9572 - va
l_loss: 1.8121 - val_sparse_top_k_categorical_accuracy: 0.8600
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization (BatchNo multiple                  256
_________________________________________________________________
conv2d (Conv2D)              multiple                  1792
_________________________________________________________________
re_lu (ReLU)                 multiple                  0
_________________________________________________________________
batch_normalization_1 (Batch multiple                  256
_________________________________________________________________
conv2d_1 (Conv2D)            multiple                  36928
_________________________________________________________________
re_lu_1 (ReLU)               multiple                  0
_________________________________________________________________
dropout (Dropout)            multiple                  0
_________________________________________________________________
max_pooling2d (MaxPooling2D) multiple                  0
_________________________________________________________________
batch_normalization_2 (Batch multiple                  512
_________________________________________________________________
conv2d_2 (Conv2D)            multiple                  73856
_________________________________________________________________
re_lu_2 (ReLU)               multiple                  0
_________________________________________________________________
batch_normalization_3 (Batch multiple                  512
_________________________________________________________________
conv2d_3 (Conv2D)            multiple                  147584
_________________________________________________________________
re_lu_3 (ReLU)               multiple                  0
_________________________________________________________________
dropout_1 (Dropout)          multiple                  0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 multiple                  0
_________________________________________________________________
batch_normalization_4 (Batch multiple                  1024
_________________________________________________________________
conv2d_4 (Conv2D)            multiple                  295168
_________________________________________________________________
re_lu_4 (ReLU)               multiple                  0
_________________________________________________________________
batch_normalization_5 (Batch multiple                  1024
```

```
_________________________________________________________________
conv2d_5 (Conv2D)            multiple                  590080
_________________________________________________________________
re_lu_5 (ReLU)               multiple                  0
_________________________________________________________________
dropout_2 (Dropout)          multiple                  0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 multiple                  0
_________________________________________________________________
batch_normalization_6 (Batch multiple                  2048
_________________________________________________________________
conv2d_6 (Conv2D)            multiple                  1180160
_________________________________________________________________
re_lu_6 (ReLU)               multiple                  0
_________________________________________________________________
batch_normalization_7 (Batch multiple                  2048
_________________________________________________________________
conv2d_7 (Conv2D)            multiple                  2359808
_________________________________________________________________
re_lu_7 (ReLU)               multiple                  0
_________________________________________________________________
dropout_3 (Dropout)          multiple                  0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 multiple                  0
_________________________________________________________________
flatten (Flatten)            multiple                  0
_________________________________________________________________
dense (Dense)                multiple                  2098176
_________________________________________________________________
dropout_4 (Dropout)          multiple                  0
_________________________________________________________________
re_lu_8 (ReLU)               multiple                  0
_________________________________________________________________
dense_1 (Dense)              multiple                  102500
=================================================================
Total params: 6,893,732
Trainable params: 6,889,892
Non-trainable params: 3,840
_________________________________________________________________
313/313 - 23s - loss: 1.6981 - sparse_top_k_categorical_accuracy: 0.8616
```