Lab: Solution - Featur...

Here's the solution I came up with

Lab: Solution - Featur...

```
import pickle
import tensorflow as tf
import numpy as np
from keras.layers import Input, Flatten, Dense
from keras.models import Model

flags = tf.app.flags
FLAGS = flags.FLAGS

# command line flags
flags.DEFINE_string('training_file', '', "Bottleneck features trainin
g file (.p)")
flags.DEFINE_string('validation_file', '', "Bottleneck features valid
ation file (.p)")
flags.DEFINE_integer('epochs', 50, "The number of epochs.")
flags.DEFINE_integer('batch_size', 256, "The batch size.")


def load_bottleneck_data(training_file, validation_file):
    """
    Utility function to load bottleneck features.

    Arguments:
        training_file - String
        validation_file - String
    """
    print("Training file", training_file)
    print("Validation file", validation_file)

    with open(training_file, 'rb') as f:
        train_data = pickle.load(f)
    with open(validation_file, 'rb') as f:
        validation_data = pickle.load(f)

    X_train = train_data['features']
    y_train = train_data['labels']
    X_val = validation_data['features']
    y_val = validation_data['labels']
```

Lab: Solution - Featur...

```
    return X_train, y_train, X_val, y_val
```

```
def main(_):
    # load bottleneck data
    X_train, y_train, X_val, y_val = load_bottleneck_data(FLAGS.train
ing_file, FLAGS.validation_file)

    print(X_train.shape, y_train.shape)
    print(X_val.shape, y_val.shape)

    nb_classes = len(np.unique(y_train))

    # define model
    input_shape = X_train.shape[1:]
    inp = Input(shape=input_shape)
    x = Flatten()(inp)
    x = Dense(nb_classes, activation='softmax')(x)
    model = Model(inp, x)
    model.compile(optimizer='adam', loss='sparse_categorical_crossent
ropy', metrics=['accuracy'])

    # train model
    model.fit(X_train, y_train, nb_epoch=FLAGS.epochs, batch_size=FLA
GS.batch_size, validation_data=(X_val, y_val), shuffle=True)


# parses flags and calls the `main` function above
if __name__ == '__main__':
    tf.app.run()
```

Let's go over the changes.

```
import numpy as np
from keras.layers import Input, Flatten, Dense
from keras.models import Model
```

Import the additional libraries required.

```
flags.DEFINE_integer('epochs', 50, "The number of epochs.")
flags.DEFINE_integer('batch_size', 256, "The batch size.")
```

I add a couple of command-line flags to set the number of epochs and batch size. This is more for convenience than anything else.

```
nb_classes = len(np.unique(y_train))
```

Here I find the number of classes for the dataset. **`np.unique` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.unique.html)** returns all the unique elements of a numpy array. The elements of `y_train` are integers, `0-9` for Cifar10 and `0-42` for Traffic Signs. So, when combined with `len` we get back the number of classes.

```
# define model
input_shape = X_train.shape[1:]
inp = Input(shape=input_shape)
x = Flatten()(inp)
x = Dense(nb_classes, activation='softmax')(x)
model = Model(inp, x)
model.compile(optimizer='adam', loss='sparse_categorical_crossentrop
y', metrics=['accuracy'])
```

Here I define a very simple model, a linear layer (`Dense` in Keras terms) followed by a softmax activation. The Adam optimizer is used.

```
# train model
model.fit(X_train, y_train, nb_epoch=FLAGS.epochs, batch_size=FLAGS.b
atch_size, validation_data=(X_val, y_val), shuffle=True)
```

Finally, the model is trained. Notice here `FLAGS.epochs` and `FLAGS.batch_size` are used.

After 50 epochs these are the results for each model:

VGG

```
Epoch 50/50
1000/1000 [==============================] - 0s - loss: 0.2418 - acc:
 0.9540 - val_loss: 0.8759 - val_acc: 0.7235
```

Inception

```
Epoch 50/50
1000/1000 [==============================] - 0s - loss: 0.0887 - acc:
 1.0000 - val_loss: 1.0428 - val_acc: 0.6556
```

ResNet

```
Epoch 50/50
1000/1000 [==============================] - 0s - loss: 0.0790 - acc:
 1.0000 - val_loss: 0.8005 - val_acc: 0.7347
```

NEXT