```
import glob
import os.path
import numpy as np
import tensorflow as tf
from tensorflow.python.platform import gfile

# 原始输入数据的目录，这个目录下有5个子目录，每个子目录底下保存这属于该
# 类别的所有图片。
INPUT_DATA = './flowers/flower_photos'
# 输出文件地址。我们将整理后的图片数据通过numpy的格式保存。
OUTPUT_FILE = './flower_processed_data.npy'
# 测试数据和验证数据比例。
VALIDATION_PERCENTAGE = 10
TEST_PERCENTAGE = 10
```

```
def create_image_lists(sess, testing_percentage, validation_percentage):
    sub_dirs = [x[0] for x in os.walk(INPUT_DATA)]
    is_root_dir = True

    # 初始化各个数据集。
    training_images = []
    training_labels = []
    testing_images = []
    testing_labels = []
    validation_images = []
    validation_labels = []
    current_label = 0

    # 读取所有的子目录。
    for sub_dir in sub_dirs:
        if is_root_dir:
            is_root_dir = False
            continue

        # 获取一个子目录中所有的图片文件。
        extensions = ['jpg', 'jpeg', 'JPG', 'JPEG']
        file_list = []
        dir_name = os.path.basename(sub_dir)
        for extension in extensions:
            file_glob = os.path.join(INPUT_DATA, dir_name, '*.' + extension)
            file_list.extend(glob.glob(file_glob))
        if not file_list: continue
        print("processing:", dir_name)

        i = 0
```

```
        # 处理图片数据。
        for file_name in file_list:
            i += 1
            image_raw_data = gfile.FastGFile(file_name, 'rb').read()
            image = tf.image.decode_jpeg(image_raw_data)
            if image.dtype != tf.float32:
                image = tf.image.convert_image_dtype(image, dtype=tf.float32)
            image = tf.image.resize_images(image, [224, 224])
            image_value = sess.run(image)

            # 随机划分数据聚。
            chance = np.random.randint(100)
            if chance < validation_percentage:
                validation_images.append(image_value)
                validation_labels.append(current_label)
            elif chance < (testing_percentage + validation_percentage):
                testing_images.append(image_value)
                testing_labels.append(current_label)
            else:
                training_images.append(image_value)
                training_labels.append(current_label)
            if i % 200 == 0:
                print (i, "images processed.")
        current_label += 1

    # 将训练数据随机打乱以获得更好的训练效果。
    state = np.random.get_state()
    np.random.shuffle(training_images)
    np.random.set_state(state)
    np.random.shuffle(training_labels)

    return np.asarray([training_images, training_labels,
                       validation_images, validation_labels,
                       testing_images, testing_labels])
```

```
with tf.Session() as sess:
    processed_data = create_image_lists(sess, TEST_PERCENTAGE, VALIDATION_PERCENTA
GE)
    # 通过numpy格式保存处理后的数据。
    np.save(OUTPUT_FILE, processed_data)
```

```
processing: roses
200 images processed.
400 images processed.
600 images processed.
processing: sunflowers
200 images processed.
400 images processed.
600 images processed.
processing: tulips
200 images processed.
400 images processed.
600 images processed.
processing: daisy
200 images processed.
400 images processed.
600 images processed.
processing: dandelion
200 images processed.
400 images processed.
600 images processed.
800 images processed.
```

```
import glob
import os.path
import numpy as np
import tensorflow as tf
from tensorflow.python.platform import gfile
import tensorflow.contrib.slim as slim
import tensorflow.contrib.slim.python.slim.nets.resnet_v1 as resnet_v1
# 处理好之后的数据文件。
INPUT_DATA = './flower_processed_data.npy'
# 保存训练好的模型的路径。
TRAIN_FILE = './model/'
# 谷歌提供的训练好的模型文件地址。因为GitHub无法保存大于100M的文件，所以
CKPT_FILE = './resnet_v1_50.ckpt'

# 定义训练中使用的参数。
LEARNING_RATE = 0.0001
STEPS = 300
BATCH = 32
N_CLASSES = 5

# 不需要从谷歌训练好的模型中加载的参数。
CHECKPOINT_EXCLUDE_SCOPES = 'Logits'
# 需要训练的网络层参数名称，在fine-tuning的过程中就是最后的全联接层。
TRAINABLE_SCOPES='Logits'
```

```
def get_tuned_variables():
    exclusions = CHECKPOINT_EXCLUDE_SCOPES

    variables_to_restore = []
    # 枚举resnet50模型中所有的参数，然后判断是否需要从加载列表中移除。
    for var in slim.get_model_variables():
        excluded = False
        for exclusion in exclusions:
            if var.op.name.startswith(exclusion):
                excluded = True
                break
        if not excluded:
            variables_to_restore.append(var)
    return variables_to_restore
```

```
def get_trainable_variables():
    scopes = TRAINABLE_SCOPES
    variables_to_train = []

    # 枚举所有需要训练的参数前缀，并通过这些前缀找到所有需要训练的参数。
    for scope in scopes:
        variables = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope)
        variables_to_train.extend(variables)
    return variables_to_train
```

```
def main():
    # 加载预处理好的数据。
    processed_data = np.load(INPUT_DATA)
    training_images = processed_data[0]
    n_training_example = len(training_images)
    training_labels = processed_data[1]

    validation_images = processed_data[2]
    validation_labels = processed_data[3]

    testing_images = processed_data[4]
    testing_labels = processed_data[5]
    resnet_model_path = CKPT_FILE
    model_save_path = TRAIN_FILE
    num_classes = 5
    batch_size = 64
    num_steps = 80
    print("%d training examples, %d validation examples and %d testing examples."
% (
        n_training_example, len(validation_labels), len(testing_labels)))

    # 定义resnet50的输入，images为输入图片，labels为每一张图片对应的标签。
```

```
    inputs = tf.placeholder(tf.float32, shape=[None, 224, 224, 3], name='inputs')
    labels = tf.placeholder(tf.int32, shape=[None], name='labels')
    is_training = tf.placeholder(tf.bool, name='is_training')

    with slim.arg_scope(resnet_v1.resnet_arg_scope()):
        net, endpoints = resnet_v1.resnet_v1_50(inputs, num_classes=None,is_traini
ng=is_training)



    with tf.variable_scope('Logits'):
        net = tf.squeeze(net, axis=[1, 2])
        net = slim.dropout(net, keep_prob=0.5, scope='scope')
        logits = slim.fully_connected(net, num_outputs=num_classes,
                                      activation_fn=None, scope='fc')


    checkpoint_exclude_scopes = 'Logits'
    exclusions = None

    if checkpoint_exclude_scopes:
        exclusions = [
            scope.strip() for scope in checkpoint_exclude_scopes.split(',')]
    variables_to_restore = []
    for var in slim.get_model_variables():
        excluded = False
        for exclusion in exclusions:
            if var.op.name.startswith(exclusion):
                excluded = True
        if not excluded:
            variables_to_restore.append(var)

    losses = tf.nn.sparse_softmax_cross_entropy_with_logits(
        labels=labels, logits=logits)
    loss = tf.reduce_mean(losses)

    logits = tf.nn.softmax(logits)
    with tf.name_scope('evaluation'):
        correct_prediction = tf.equal(tf.cast(tf.argmax(logits, 1), dtype=tf.int32
), labels)
        evaluation_step = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    classes = tf.argmax(logits, axis=1, name='classes')
    accuracy = tf.reduce_mean(tf.cast(
        tf.equal(tf.cast(classes, dtype=tf.int32), labels), dtype=tf.float32))

    optimizer = tf.train.AdamOptimizer(learning_rate=0.0001)
    train_step = optimizer.minimize(loss)
```

```
    init = tf.global_variables_initializer()

    saver_restore = tf.train.Saver(var_list=variables_to_restore)
    saver = tf.train.Saver(tf.global_variables())

    # 计算正确率。


    with tf.Session() as sess:
        sess.run(init)

        # Load the pretrained checkpoint file xxx.ckpt
        saver_restore.restore(sess, resnet_model_path)

        start = 0
        end = batch_size
        for i in range(num_steps):
            #images, groundtruth_lists = get_next_batch(batch_size, ...)
            train_dict = {inputs: training_images[start:end],
                          labels: training_labels[start:end],
                          is_training: True}

            sess.run(train_step, feed_dict=train_dict)

            loss_, acc_ = sess.run([loss, accuracy], feed_dict=train_dict)

            start = end
            if start == n_training_example:
                start = 0

            end = start + batch_size
            if end > n_training_example:
                end = n_training_example

            if (i+1) % 5 == 0 or i + 1 == STEPS:
                train_text = 'Step: {}, Loss: {:.4f}, Accuracy: {:.4f}'.format(i+1
, loss_, acc_)
                print(train_text)
                saver.save(sess, model_save_path, global_step=i+1)
                print('save mode to {}'.format(model_save_path))
        test_accuracy = sess.run(evaluation_step, feed_dict={inputs: testing_image
s, labels: testing_labels, is_training: False})
        print('Final test accuracy = %.1f%%' % (test_accuracy * 100))
```

```
if __name__ == '__main__':
    main()
```

```
2941 training examples, 376 validation examples and 353 testing examples.
INFO:tensorflow:Restoring parameters from ./resnet_v1_50.ckpt
Step: 5, Loss: 1.1779, Accuracy: 0.4844
save mode to ./model/
Step: 10, Loss: 0.4245, Accuracy: 0.8750
save mode to ./model/
Step: 15, Loss: 0.3478, Accuracy: 0.8750
save mode to ./model/
Step: 20, Loss: 0.2472, Accuracy: 0.9375
save mode to ./model/
Step: 25, Loss: 0.2444, Accuracy: 0.8906
save mode to ./model/
Step: 30, Loss: 0.3396, Accuracy: 0.8594
save mode to ./model/
Step: 35, Loss: 0.2773, Accuracy: 0.9062
save mode to ./model/
Step: 40, Loss: 0.2443, Accuracy: 0.8906
save mode to ./model/
Step: 45, Loss: 0.1210, Accuracy: 0.9531
save mode to ./model/
Step: 50, Loss: 0.1438, Accuracy: 0.9219
save mode to ./model/
Step: 55, Loss: 0.0591, Accuracy: 0.9844
save mode to ./model/
Step: 60, Loss: 0.0337, Accuracy: 0.9844
save mode to ./model/
Step: 65, Loss: 0.0375, Accuracy: 1.0000
save mode to ./model/
Step: 70, Loss: 0.0242, Accuracy: 1.0000
save mode to ./model/
Step: 75, Loss: 0.0400, Accuracy: 1.0000
save mode to ./model/
Step: 80, Loss: 0.0089, Accuracy: 1.0000
save mode to ./model/
Final test accuracy = 93.8%
```