In [1]:

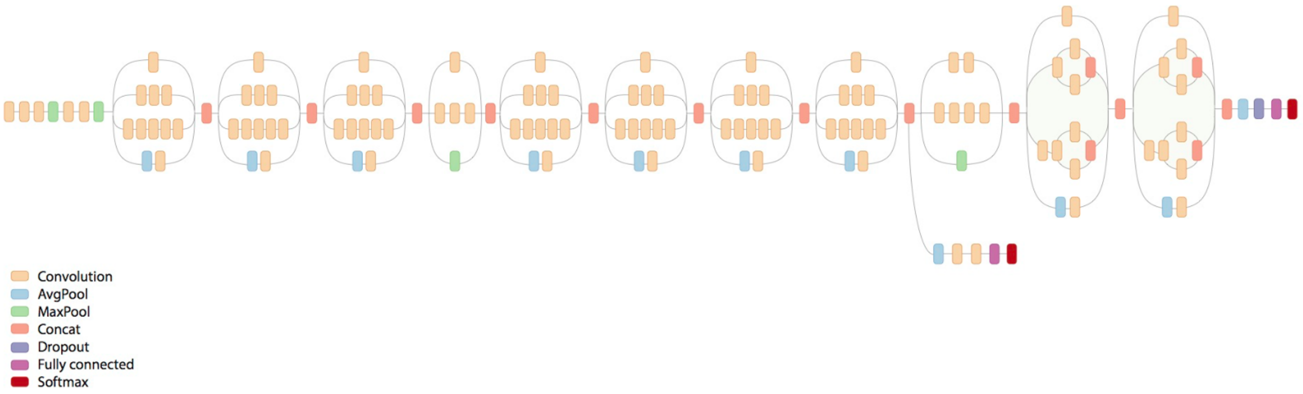
```
# set tf 1.x for colab
%tensorflow_version 1.x
```

```
UsageError: Line magic function `%tensorflow_version` not found.
```

Fine-tuning InceptionV3 for flowers classification

In this task you will fine-tune InceptionV3 architecture for flowers classification task.

InceptionV3 architecture (https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html (https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html)):



Flowers classification dataset (http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html (http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html)) consists of 102 flower categories commonly occurring in the United Kingdom. Each class contains between 40 and 258 images:

Import stuff

In [2]:

```
import sys
sys.path.append("..")
import grading
import download_utils
```

In [3]:

```
# !!! remember to clear session/graph if you rebuild your graph to avoid out-of-memory errors !!!
```

In [4]:

```
download_utils.link_all_keras_resources()
```

In [5]:

```
import tensorflow as tf
import keras
from keras import backend as K
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
print(tf.__version__)
print(keras.__version__)
import cv2  # for image processing
from sklearn.model_selection import train_test_split
import scipy.io
import os
import tarfile
import keras_utils
from keras_utils import reset_tf_session
```

```
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing
```

Fill in your Coursera token and email

To successfully submit your answers to our grader, please fill in your Coursera submission token and email

In [6]:

```
grader = grading.Grader(assignment_key="2v-uxpD7EeeMxQ6FWsz5LA",
                        all_parts=["wuwwC", "a4FK1", "qRsZ1"])
```

In [7]:

```
# token expires every 30 min
COURSERA_TOKEN = "IcfYrqfw78BHHOHV"
COURSERA_EMAIL = "lxwvictor@gmail.com"
```

Load dataset

Dataset was downloaded for you, it takes 12 min and 400mb. Relevant links (just in case):

- http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html (http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html)
- http://www.robots.ox.ac.uk/~vgg/data/flowers/102/102flowers.tgz (http://www.robots.ox.ac.uk/~vgg/data/flowers/102/102flowers.tgz)
- http://www.robots.ox.ac.uk/~vgg/data/flowers/102/imagelabels.mat (http://www.robots.ox.ac.uk/~vgg/data/flowers/102/imagelabels.mat)

In [8]:

```
# we downloaded them for you, just link them here
download_utils.link_week_3_resources()
```

Prepare images for model

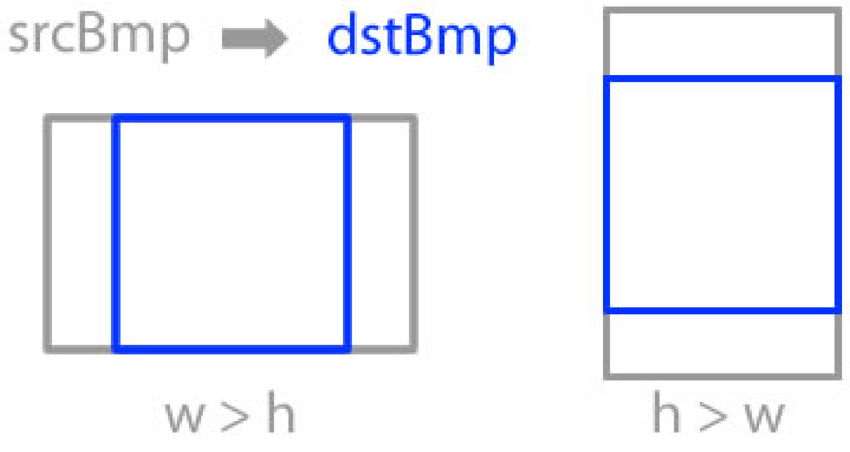
In [9]:

```
# we will crop and resize input images to IMG_SIZE x IMG_SIZE
IMG_SIZE = 250
```

In [10]:

```
def decode_image_from_raw_bytes(raw_bytes):
    img = cv2.imdecode(np.asarray(bytearray(raw_bytes), dtype=np.uint8), 1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img
```

We will take a center crop from each image like this:



In [11]:

```python
def image_center_crop(img):
    """
    Makes a square center crop of an img, which is a [h, w, 3] numpy array.
    Returns [min(h, w), min(h, w), 3] output with same width and height.
    For cropping use numpy slicing.
    """

    ### YOUR CODE HERE
    h, w, c = img.shape

    if w > h:
        w_start = (w-h)//2
        w_end = w_start + h
        cropped_img = img[:, w_start:w_end, :]
    else:
        h_start = (h-w)//2
        h_end = h_start + w
        cropped_img = img[h_start:h_end, :, :]

    # checks for errors
#     h, w, c = img.shape
    assert cropped_img.shape == (min(h, w), min(h, w), c), "error in image_center_crop!"

    return cropped_img
```

In [12]:

```python
def prepare_raw_bytes_for_model(raw_bytes, normalize_for_model=True):
    img = decode_image_from_raw_bytes(raw_bytes)  # decode image raw bytes to matrix
    img = image_center_crop(img)  # take squared center crop
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))  # resize for our model
    if normalize_for_model:
        img = img.astype("float32")  # prepare for normalization
        img = keras.applications.inception_v3.preprocess_input(img)  # normalize for model
    return img
```

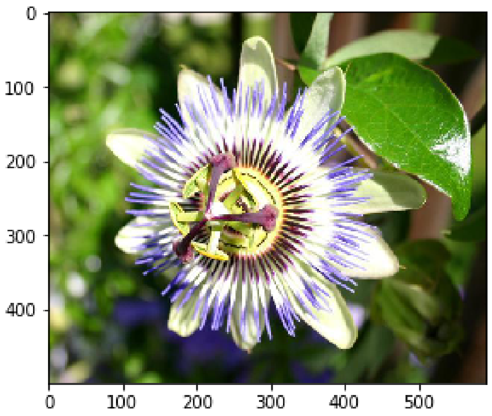
In [13]:

```python
# reads bytes directly from tar by filename (slow, but ok for testing, takes ~6 sec)
def read_raw_from_tar(tar_fn, fn):
    with tarfile.open(tar_fn) as f:
        m = f.getmember(fn)
        return f.extractfile(m).read()
```

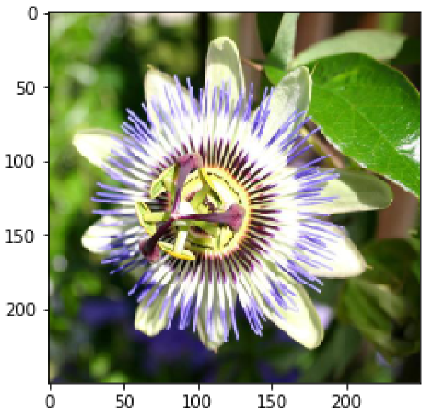
In [14]:

```python
# test cropping
raw_bytes = read_raw_from_tar("102flowers.tgz", "jpg/image_00001.jpg")

img = decode_image_from_raw_bytes(raw_bytes)
print(img.shape)
plt.imshow(img)
plt.show()

img = prepare_raw_bytes_for_model(raw_bytes, normalize_for_model=False)
print(img.shape)
plt.imshow(img)
plt.show()
```

```
(500, 591, 3)
```



```
(250, 250, 3)
```



In [15]:

```python
## GRADED PART, DO NOT CHANGE!
# Test image preparation for model
prepared_img = prepare_raw_bytes_for_model(read_raw_from_tar("102flowers.tgz", "jpg/image_00001.jpg"))
grader.set_answer("qRsZ1", list(prepared_img.shape) + [np.mean(prepared_img), np.std(prepared_img)])
```

In [16]:

```python
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

Prepare for training

In [17]:

```python
# read all filenames and labels for them

# read filenames firectly from tar
def get_all_filenames(tar_fn):
    with tarfile.open(tar_fn) as f:
        return [m.name for m in f.getmembers() if m.isfile()]

all_files = sorted(get_all_filenames("102flowers.tgz"))  # list all files in tar sorted by name
all_labels = scipy.io.loadmat('imagelabels.mat')['labels'][0] - 1  # read class labels (0, 1, 2, ...)
# all_files and all_labels are aligned now
N_CLASSES = len(np.unique(all_labels))
print(N_CLASSES)
```

```
102
```

In [18]:

```python
# split into train/test
tr_files, te_files, tr_labels, te_labels = \
    train_test_split(all_files, all_labels, test_size=0.2, random_state=42, stratify=all_labels)
```

In [19]:

```python
# will yield raw image bytes from tar with corresponding label
def raw_generator_with_label_from_tar(tar_fn, files, labels):
    label_by_fn = dict(zip(files, labels))
    with tarfile.open(tar_fn) as f:
        while True:
            m = f.next()
            if m is None:
                break
            if m.name in label_by_fn:
                yield f.extractfile(m).read(), label_by_fn[m.name]
```

In [20]:

```python
# batch generator
BATCH_SIZE = 32

def batch_generator(items, batch_size):
    """
    Implement batch generator that yields items in batches of size batch_size.
    There's no need to shuffle input items, just chop them into batches.
    Remember about the last batch that can be smaller than batch_size!
    Input: any iterable (list, generator, ...). You should do `for item in items: ...`
        In case of generator you can pass through your items only once!
    Output: In output yield each batch as a list of items.
    """

    ### YOUR CODE HERE
    count = 0
    batch = []
    for item in items:
        batch.append(item)
        count += 1
        if count == batch_size:
            yield batch
            count = 0
            batch = []

    yield batch
```

In [21]:

```python
## GRADED PART, DO NOT CHANGE!
# Test batch generator
def _test_items_generator():
    for i in range(10):
        yield i

grader.set_answer("a4FK1", list(map(lambda x: len(x), batch_generator(_test_items_generator(), 3))))
```

In [22]:

```python
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

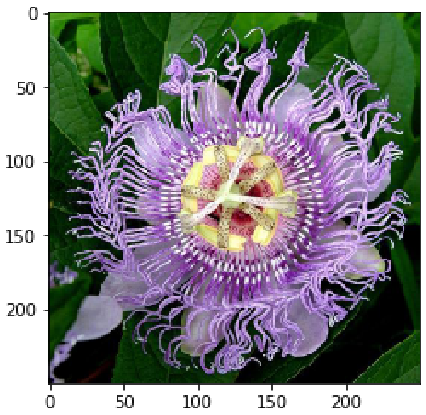
In [23]:

```python
def train_generator(files, labels):
    while True:  # so that Keras can loop through this as long as it wants
        for batch in batch_generator(raw_generator_with_label_from_tar(
                "102flowers.tgz", files, labels), BATCH_SIZE):
            # prepare batch images
            batch_imgs = []
            batch_targets = []
            for raw, label in batch:
                img = prepare_raw_bytes_for_model(raw)
                batch_imgs.append(img)
                batch_targets.append(label)
            # stack images into 4D tensor [batch_size, img_size, img_size, 3]
            batch_imgs = np.stack(batch_imgs, axis=0)
            # convert targets into 2D tensor [batch_size, num_classes]
            batch_targets = keras.utils.np_utils.to_categorical(batch_targets, N_CLASSES)
            yield batch_imgs, batch_targets
```

In [24]:

```python
# test training generator
for _ in train_generator(tr_files, tr_labels):
    print(_[0].shape, _[1].shape)
    plt.imshow(np.clip(_[0][0] / 2. + 0.5, 0, 1))
    break
```

```
(32, 250, 250, 3) (32, 102)
```



Training

You cannot train such a huge architecture from scratch with such a small dataset.

But using fine-tuning of last layers of pre-trained network you can get a pretty good classifier very quickly.

In [25]:

```python
# remember to clear session if you start building graph from scratch!
s = reset_tf_session()
# don't call K.set_learning_phase() !!! (otherwise will enable dropout in train/test simultaneously)
```

```
WARNING:tensorflow:From ..\keras_utils.py:68: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_se
ssion instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:95: The name
tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:98: The name
tf.placeholder_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:102: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From ..\keras_utils.py:75: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.
```

In [26]:

```python
def inception(use_imagenet=True):
    # load pre-trained model graph, don't add final layer
    model = keras.applications.InceptionV3(include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3),
                                          weights='imagenet' if use_imagenet else None)
    # add global pooling just like in InceptionV3
    new_output = keras.layers.GlobalAveragePooling2D()(model.output)
    # add new dense layer for our labels
    new_output = keras.layers.Dense(N_CLASSES, activation='softmax')(new_output)
    model = keras.engine.training.Model(model.inputs, new_output)
    return model
```

In [27]:

```python
model = inception()
```

```
WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:1834: The nam
e tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:3976: The nam
e tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:3980: The nam
e tf.nn.avg_pool is deprecated. Please use tf.nn.avg_pool2d instead.
```

In [28]:

```python
model.summary()
```

```
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
input_1 (InputLayer)            (None, 250, 250, 3)  0
____________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 124, 124, 32) 864         input_1[0][0]
____________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 124, 124, 32) 96          conv2d_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)       (None, 124, 124, 32) 0           batch_normalization_1[0][0]
____________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 122, 122, 32) 9216        activation_1[0][0]
____________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 122, 122, 32) 96          conv2d_2[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)       (None, 122, 122, 32) 0           batch_normalization_2[0][0]
____________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 122, 122, 64) 18432       activation_2[0][0]
____________________________________________________________________________________________________
```

In [29]:

```python
# how many layers our model has
print(len(model.layers))
```

```
313
```

In [30]:

```python
# set all layers trainable by default
for layer in model.layers:
    layer.trainable = True
    if isinstance(layer, keras.layers.BatchNormalization):
        # we do aggressive exponential smoothing of batch norm
        # parameters to faster adjust to our new dataset
        layer.momentum = 0.9

# fix deep layers (fine-tuning only last 50)
for layer in model.layers[:-50]:
    # fix all but batch norm layers, because we neeed to update moving averages for a new dataset!
    if not isinstance(layer, keras.layers.BatchNormalization):
        layer.trainable = False
```

In [31]:

```python
# compile new model
model.compile(
    loss='categorical_crossentropy',  # we train 102-way classification
    optimizer=keras.optimizers.adamax(lr=1e-2),  # we can take big lr here because we fixed first layers
    metrics=['accuracy']  # report accuracy during training
)
```

```
WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\optimizers.py:790: The name tf.train.Optimi
zer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

In [32]:

```python
# we will save model checkpoints to continue training in case of kernel death
model_filename = 'flowers.{0:03d}.hdf5'
last_finished_epoch = None

#### uncomment below to continue training from model checkpoint
#### fill `last_finished_epoch` with your latest finished epoch
# from keras.models import load_model
# s = reset_tf_session()
# last_finished_epoch = 10
# model = load_model(model_filename.format(last_finished_epoch))
```

Training takes **2 hours**. You're aiming for ~0.93 validation accuracy.

In [33]:

```python
# fine tune for 2 epochs (full passes through all training data)
# we make 2*8 epochs, where epoch is 1/8 of our training data to see progress more often
model.fit_generator(
    train_generator(tr_files, tr_labels),
    steps_per_epoch=len(tr_files) // BATCH_SIZE // 8,
    epochs=2 * 8,
    validation_data=train_generator(te_files, te_labels),
    validation_steps=len(te_files) // BATCH_SIZE // 4,
    callbacks=[keras_utils.TqdmProgressCallback(),
               keras_utils.ModelSaveCallback(model_filename)],
    verbose=0,
    initial_epoch=last_finished_epoch or 0
)
```

```
WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\ops\math_grad.py:1250: add_disp
atch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

Epoch 1/16
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Model saved in flowers.000.hdf5

Epoch 2/16
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Model saved in flowers.001.hdf5
```

In [34]:

```
## GRADED PART, DO NOT CHANGE!
# Accuracy on validation set
test_accuracy = model.evaluate_generator(
    train_generator(te_files, te_labels),
    len(te_files) // BATCH_SIZE // 2
)[1]
grader.set_answer("wuwwC", test_accuracy)
print(test_accuracy)
```

```
0.94625
```

In [35]:

```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

That's it! Congratulations!

What you've done:

- prepared images for the model
- implemented your own batch generator
- fine-tuned the pre-trained model