In [1]:

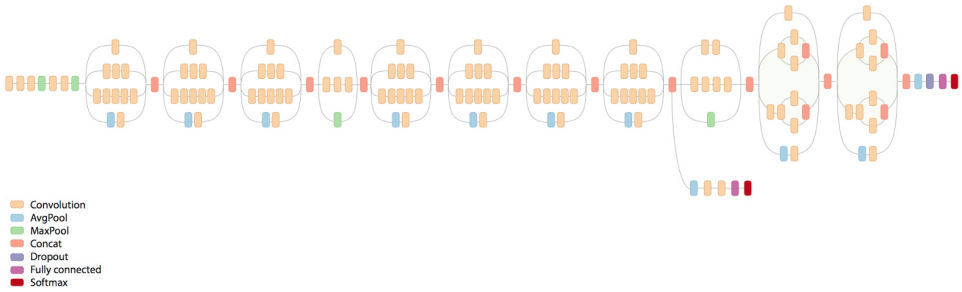
```
! shred -u setup_google_colab.py
! wget https://raw.githubusercontent.com/hse-aml/intro-to-dl/master/setup_google
import setup_google_colab
# please, uncomment the week you're working on
# setup_google_colab.setup_week1()
# setup_google_colab.setup_week2()
# setup_google_colab.setup_week2_honor()
setup_google_colab.setup_week3()
# setup_google_colab.setup_week4()
# setup_google_colab.setup_week5()
# setup_google_colab.setup_week6()
# set tf 1.x for colab

```

```
shred: setup_google_colab.py: failed to open for writing: No such file
or directory
--2022-04-11 20:57:10--  https://raw.githubusercontent.com/hse-aml/int
ro-to-dl/master/setup_google_colab.py (https://raw.githubusercontent.c
om/hse-aml/intro-to-dl/master/setup_google_colab.py)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 18
5.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|18
5.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3636 (3.6K) [text/plain]
Saving to: ‘setup_google_colab.py’

setup_google_colab. 100%[===================>]   3.55K  --.-KB/s    in
0s

2022-04-11 20:57:10 (46.7 MB/s) - ‘setup_google_colab.py’ saved [3636/
3636]

**************************************************
102flowers.tgz
**************************************************
imagelabels.mat
**************************************************
inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
**************************************************
cifar-10-batches-py.tar.gz
**************************************************
mnist.npz
```

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-
```

In [4]:

```
download_utils.link_all_keras_resources()
```

In [5]:

```
!pip uninstall keras-nightly
!pip uninstall -y tensorflow
!pip install keras==2.1.6
!pip install tensorflow==1.15.0
!pip install install h5py==2.10.0

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
```

...

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 = '4KfJM7vv3brSpvrs'### YOUR TOKEN HERE
COURSERA_EMAIL = 'e0321294@u.nus.edu'### YOUR EMAIL HERE
```

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

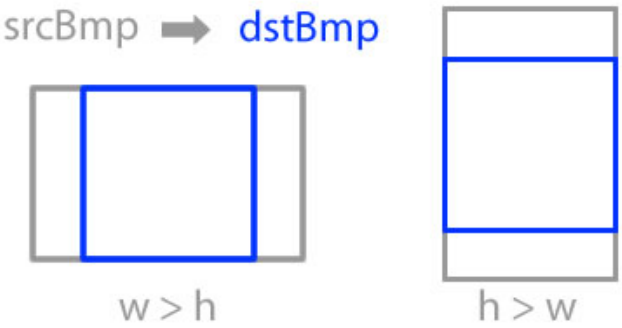
In [9]:

```
# 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 [12]:

```
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.
    """
    h, w = img.shape[0], img.shape[1]
    m = min(h, w)
    cropped_img =  img[(h-m)//2:(h+m)//2, (w-m)//2:(w+m)//2, :]
    ### YOUR CODE HERE


    return cropped_img
```

In [13]:

```
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 ma
    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
    return img
```

In [14]:

```
# reads bytes directly from tar by filename (slow, but ok for testing, takes ~6
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 [15]:

```
# 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()
```

...

In [16]:

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

In [17]:

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

...

Prepare for training

In [18]:

```
# 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
all_labels = scipy.io.loadmat('imagelabels.mat')['labels'][0] - 1  # read class
# all_files and all_labels are aligned now
N_CLASSES = len(np.unique(all_labels))
print(N_CLASSES)
```

...

In [19]:

```
# 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, stra
```

In [20]:

```
# 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 [21]:

```
# 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.
    """
    #
    batch = []
    for i, item in enumerate(items):
        batch.append(item)
        if i % batch_size == batch_size-1:
            yield batch
            batch = []
    if batch[0]:
        yield [item for item in batch if item]
    ### YOUR CODE HERE
```

In [22]:

```
## 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_item
```

In [23]:

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

...

In [24]:

```
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
            yield batch_imgs, batch_targets
```

In [25]:

```
# 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
```

...

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 [26]:

```
# 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/
```

...

In [27]:

```
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_S
                                           weights='imagenet' if use_imagenet els
    # 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 [28]:

```
model = inception()
```

...

In [29]:

```
model.summary()
```

...

In [30]:

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

...

In [31]:

```
# 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
    if not isinstance(layer, keras.layers.BatchNormalization):
        layer.trainable = False
```

In [32]:

```
# 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 becau
    metrics=['accuracy']  # report accuracy during training
)
```

...

In [33]:

```
# 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 [34]:

```
# 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 mo
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
)
```

...

In [35]:

```
## 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)
```

...

In [36]:

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

...

That's it! Congratulations!

What you've done:

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