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

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):

Flowers classification dataset (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:

Running on Google Colab

Copy all files of intro-to-dl-master in root_path

```
! shred -u setup_google_colab.py
! wget https://raw.githubusercontent.com/hse-aml/intro-to-dl/master/setup_google_colab.
```

```
shred: setup_google_colab.py: failed to open for writing: No such file or directory
--2021-01-17 22:56:44--  https://raw.githubusercontent.com/hse-aml/intro-to-dl/master
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:44
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

2021-01-17 22:56:45 (36.9 MB/s) - 'setup_google_colab.py' saved [3636/3636]
```

```
import setup_google_colab
setup_google_colab.setup_week3()
```

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

Import stuff

```
root_path = '/content/gdrive/MyDrive/HSE_Introduction to Deep Learning'
```

```
import sys
sys.path.append(root_path)
import grading_utils
import download_utils
```

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

```
download_utils.link_all_keras_resources()
```

```
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
import grading
```

```
1.15.2
2.3.1
```

Fill in your Coursera token and email

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

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

```
# token expires every 30 min
COURSERA_TOKEN = 'Rd1AjChnFdHTNvS8'       ### YOUR TOKEN HERE
COURSERA_EMAIL = 'knowtech94@gmail.com'   ### 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/102/102flowers.tgz
- http://www.robots.ox.ac.uk/~vgg/data/flowers/102/imagelabels.mat

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

Prepare images for model

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

```
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:

```
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.
    """

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

    if w > h:
      cropped_img = img[int((w-h)/2):int((w+h)/2), 0:h, 0:c]
    elif h > w:
      cropped_img = img[0:w, int((h-w)/2):int((h+w)/2), 0:c]
    else:
      cropped_img = img[0:w, 0:h, 0:c]

    return cropped_img

```

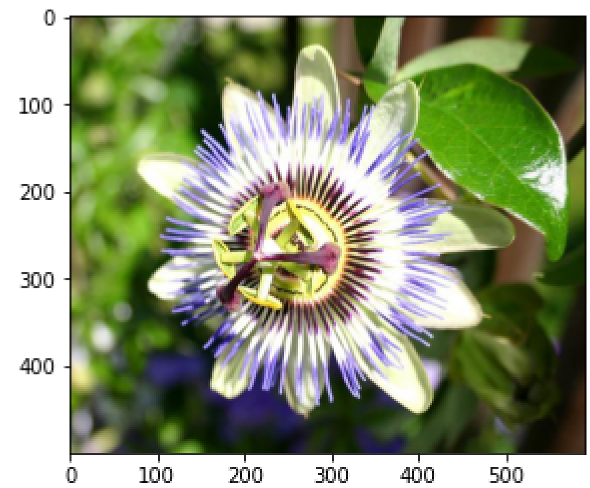
```
def prepare_raw_bytes_for_model(raw_bytes, normalize_for_model=True):
    img = decode_image_from_raw_bytes(raw_bytes)  # decode image raw bytes to matrix
```

```
    img = 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 mo
    return img
```

```
# 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()
```

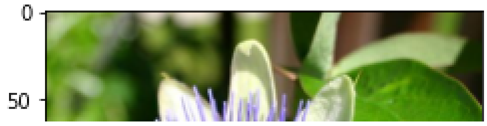
```
# test cropping
raw_bytes = read_raw_from_tar("102flowers.tgz", "jpg/image_00001.jpg")

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

```

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



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

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



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

```
# 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

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

```
102
```

```
# 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=al
```

```
# 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]
```

```
# 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
    batch_generator = []

    for item in items:
      batch_generator.append(item)
      if len(batch_generator) == batch_size:
        yield batch_generator
        batch_generator = []
    if batch_generator:
      yield batch_generator

```

```
## 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_gener
```

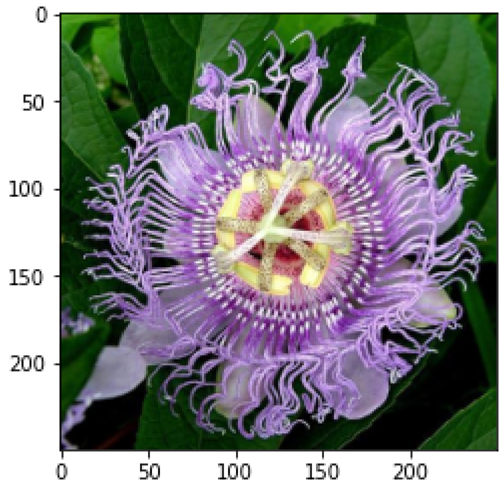
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

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

```
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_CLASSE
            yield batch_imgs, batch_targets
```

```
# 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.

```
# 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 si
```

```
WARNING:tensorflow:From /content/gdrive/MyDrive/HSE_Introduction to Deep Learning/ker

WARNING:tensorflow:From /content/gdrive/MyDrive/HSE_Introduction to Deep Learning/ker

WARNING:tensorflow:From /content/gdrive/MyDrive/HSE_Introduction to Deep Learning/ker
```

```
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, IM
                                          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
```

```
1 model = inception()
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/ops/resou
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/keras/backend/tensorflow_backen

WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/keras/backend/tensorflow_backen
```

```
1 model.summary()
```

```
batch_normalization_87 (BatchNo (None, 6, 6, 384)    1152        conv2d_87[0][0]
________________________________________________________________________________
batch_normalization_91 (BatchNo (None, 6, 6, 384)    1152        conv2d_91[0][0]
________________________________________________________________________________
activation_87 (Activation)      (None, 6, 6, 384)    0           batch_normalizati
________________________________________________________________________________
activation_91 (Activation)      (None, 6, 6, 384)    0           batch_normalizati
________________________________________________________________________________
conv2d_88 (Conv2D)              (None, 6, 6, 384)    442368      activation_87[0][
________________________________________________________________________________
conv2d_89 (Conv2D)              (None, 6, 6, 384)    442368      activation_87[0][
________________________________________________________________________________
conv2d_92 (Conv2D)              (None, 6, 6, 384)    442368      activation_91[0][
________________________________________________________________________________
conv2d_93 (Conv2D)              (None, 6, 6, 384)    442368      activation_91[0][
________________________________________________________________________________
average_pooling2d_9 (AveragePoo (None, 6, 6, 2048)   0           mixed9[0][0]
________________________________________________________________________________
conv2d_86 (Conv2D)              (None, 6, 6, 320)    655360      mixed9[0][0]
________________________________________________________________________________
batch_normalization_88 (BatchNo (None, 6, 6, 384)    1152        conv2d_88[0][0]
________________________________________________________________________________
batch_normalization_89 (BatchNo (None, 6, 6, 384)    1152        conv2d_89[0][0]
________________________________________________________________________________
batch_normalization_92 (BatchNo (None, 6, 6, 384)    1152        conv2d_92[0][0]
________________________________________________________________________________
batch_normalization_93 (BatchNo (None, 6, 6, 384)    1152        conv2d_93[0][0]
________________________________________________________________________________
conv2d_94 (Conv2D)              (None, 6, 6, 192)    393216      average_pooling2d_
________________________________________________________________________________
batch_normalization_86 (BatchNo (None, 6, 6, 320)    960         conv2d_86[0][0]
________________________________________________________________________________
activation_88 (Activation)      (None, 6, 6, 384)    0           batch_normalizati
________________________________________________________________________________
activation_89 (Activation)      (None, 6, 6, 384)    0           batch_normalizati
________________________________________________________________________________
activation_92 (Activation)      (None, 6, 6, 384)    0           batch_normalizati
________________________________________________________________________________
activation_93 (Activation)      (None, 6, 6, 384)    0           batch_normalizati
________________________________________________________________________________
batch_normalization_94 (BatchNo (None, 6, 6, 192)    576         conv2d_94[0][0]
________________________________________________________________________________
activation_86 (Activation)      (None, 6, 6, 320)    0           batch_normalizati
________________________________________________________________________________
mixed9_1 (Concatenate)          (None, 6, 6, 768)    0           activation_88[0][
                                                                 activation_89[0][
________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 6, 6, 768)    0           activation_92[0][
```

```
concatenate_2 (Concatenate)     (None, 6, 6, 768)    0           activation_92[0][(
                                                                 activation_93[0][(
__________________________________________________________________________________
activation_94 (Activation)      (None, 6, 6, 192)    0           batch_normalizati(
__________________________________________________________________________________
mixed10 (Concatenate)           (None, 6, 6, 2048)   0           activation_86[0][(
                                                                 mixed9_1[0][0]
                                                                 concatenate_2[0][(
                                                                 activation_94[0][(
__________________________________________________________________________________
global_average_pooling2d_1 (Glo (None, 2048)         0           mixed10[0][0]
__________________________________________________________________________________
```

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

```
313
```

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

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

```
# 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.

```
# 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 ofte
model.fit_generator(
    train_generator(tr_files, tr_labels),
    steps_per_epoch=len(tr_files) // BATCH_SIZE // 8,
    epochs= 10,
    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 /tensorflow-1.15.2/python3.6/keras/backend/tensorflow_backenc


Epoch 1/10
**************************
loss: 4.4671; accuracy: 0.1904; val_loss: 21.5944; val_accuracy: 0.1432
Model saved in flowers.000.hdf5

Epoch 2/10
**************************
loss: 2.5581; accuracy: 0.4232; val_loss: 6.8255; val_accuracy: 0.3698
Model saved in flowers.001.hdf5

Epoch 3/10
**************************
loss: 1.5520; accuracy: 0.6190; val_loss: 3.0175; val_accuracy: 0.5339
Model saved in flowers.002.hdf5

Epoch 4/10
**************************
loss: 1.0951; accuracy: 0.7233; val_loss: 0.5756; val_accuracy: 0.7292
Model saved in flowers.003.hdf5

Epoch 5/10
**************************
loss: 0.7477; accuracy: 0.8038; val_loss: 1.2126; val_accuracy: 0.8128
Model saved in flowers.004.hdf5

Epoch 6/10
**************************
loss: 0.6746; accuracy: 0.8394; val_loss: 1.0167; val_accuracy: 0.8802
Model saved in flowers.005.hdf5

Epoch 7/10
**************************
loss: 0.5160; accuracy: 0.8741; val_loss: 0.1965; val_accuracy: 0.9167
Model saved in flowers.006.hdf5

Epoch 8/10
**************************
loss: 0.4151; accuracy: 0.8800; val_loss: 0.2613; val_accuracy: 0.8828
Model saved in flowers.007.hdf5

Epoch 9/10
**************************
```

```
loss: 0.2868; accuracy: 0.9304; val_loss: 0.2643; val_accuracy: 0.9022
Model saved in flowers.008.hdf5

Epoch 10/10
**************************
loss: 0.2178; accuracy: 0.9549; val_loss: 0.1252; val_accuracy: 0.9271
Model saved in flowers.009.hdf5
<keras.callbacks.callbacks.History at 0x7fbea1169eb8>
```

```
## 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)
```

```
0.9137499928474426
```

```
# 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