lab_fine_tune_partial_colab_flickr_dl4111

November 22, 2021

Lab: Transfer Learning with a Pre-Trained Deep Neural Network

As we discussed earlier, state-of-the-art neural networks involve millions of parameters that are prohibitively difficult to train from scratch. In this lab, we will illustrate a powerful technique called *fine-tuning* where we start with a large pre-trained network and then re-train only the final layers to adapt to a new task. The method is also called *transfer learning* and can produce excellent results on very small datasets with very little computational time.

This lab is based partially on this excellent blog. In performing the lab, you will learn to: * Build a custom image dataset * Fine tune the final layers of an existing deep neural network for a new classification task. * Load images with a `DataGenerator`.

The lab has two versions: * *CPU version*: In this version, you use lower resolution images so that the lab can be performed on your laptop. The resulting accuracy is lower. The code will also take considerable time to execute. * *GPU version*: This version uses higher resolution images but requires a GPU instance. See the notes on setting up a GPU instance on Google Cloud Platform. The GPU training is much faster (< 1 minute).

MS students must complete the GPU version of this lab.

```
[70]: from google.colab import drive
      drive.mount('/content/drive')
```

Mounted at /content/drive

0.1 Create a Dataset

In this example, we will try to develop a classifier that can discriminate between two classes: `cars` and `bicycles`. One could imagine this type of classifier would be useful in vehicle vision systems. The first task is to build a dataset.

TODO: Create training and test datasets with: * 1000 training images of cars * 1000 training images of bicylces * 300 test images of cars * 300 test images of bicylces * The images don't need to be the same size. But, you can reduce the resolution if you need to save disk space.

The images should be organized in the following directory structure:

```
./train
    /car
        car_0000.jpg
        car_0001.jpg
        ...
```

```
        car_0999.jpg
    /bicycle
        bicycle_0000.jpg
        bicycle_0001.jpg
        ...
        bicycle_0999.jpg
./test
    /car
        car_1001.jpg
        car_1001.jpg
        ...
        car_1299.jpg
    /bicycle
        bicycle_1000.jpg
        bicycle_1001.jpg
        ...
        bicycle_1299.jpg
```

The naming of the files within the directories does not matter. The `ImageDataGenerator` class below will find the filenames. Just make sure there are the correct number of files in each directory.

A nice automated way of building such a dataset if through the FlickrAPI. Remember that if you run the FlickrAPI twice, it may collect the same images. So, you need to run it once and split the images into training and test directories.

Installing the flickrapi in colab:

[72]:
```
!pip install flickrapi
```

```
Collecting flickrapi
  Downloading flickrapi-2.4.0-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: requests>=2.2.1 in /usr/local/lib/python3.7/dist-
packages (from flickrapi) (2.23.0)
Requirement already satisfied: six>=1.5.2 in /usr/local/lib/python3.7/dist-
packages (from flickrapi) (1.15.0)
Requirement already satisfied: requests-oauthlib>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from flickrapi) (1.3.0)
Collecting requests-toolbelt>=0.3.1
  Downloading requests_toolbelt-0.9.1-py2.py3-none-any.whl (54 kB)
       |                        | 54 kB 2.2 MB/s
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.2.1->flickrapi) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.2.1->flickrapi)
(2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.2.1->flickrapi) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.2.1->flickrapi)
```

```
(1.24.3)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-
packages (from requests-oauthlib>=0.4.0->flickrapi) (3.1.1)
Installing collected packages: requests-toolbelt, flickrapi
Successfully installed flickrapi-2.4.0 requests-toolbelt-0.9.1
```

```python
[73]: import flickrapi
      import urllib.request
      import matplotlib.pyplot as plt
      import numpy as np
      import skimage.io
      import skimage.transform
      import requests
      from io import BytesIO
      %matplotlib inline
```

Make sure to get a api key and secret code and change below:

```python
[74]: # see the flickr demo on where to get keys.
      api_key = u'b899eb9af7eb27a4edab5a5edd3f739d'  ␣
       ↪#u'0bcc03asdfasge12412341324041a6d457cb18321'
      api_secret = u'3a64fb4bb3a4b393'  #u'bbeasdfasdf3327a0'
      flickr = flickrapi.FlickrAPI(api_key, api_secret)
```

Download bicycles

```python
[75]: import warnings
      import os
      dir_name = 'train'
      dir_exists = os.path.isdir(dir_name)
      if not dir_exists:
          os.mkdir(dir_name)
          print("Making directory %s" % dir_name)
      else:
          print("Will store images in directory %s" % dir_name)

      dir_name = 'test'
      dir_exists = os.path.isdir(dir_name)
      if not dir_exists:
          os.mkdir(dir_name)
          print("Making directory %s" % dir_name)
      else:
          print("Will store images in directory %s" % dir_name)

      dir_folder = os.path.join(os.getcwd(),'train','bicycle')
      dir_exists = os.path.isdir(dir_folder)
      if not dir_exists:
          os.mkdir(dir_folder)
```

```python
        print("Making directory %s" % dir_folder)
else:
        print("Will store images in directory %s" % dir_folder)

dir_folder = os.path.join(os.getcwd(),'train','car')
dir_exists = os.path.isdir(dir_folder)
if not dir_exists:
        os.mkdir(dir_folder)
        print("Making directory %s" % dir_folder)
else:
        print("Will store images in directory %s" % dir_folder)

dir_folder = os.path.join(os.getcwd(),'test','bicycle')
dir_exists = os.path.isdir(dir_folder)
if not dir_exists:
        os.mkdir(dir_folder)
        print("Making directory %s" % dir_folder)
else:
        print("Will store images in directory %s" % dir_folder)

dir_folder = os.path.join(os.getcwd(),'test','car')
dir_exists = os.path.isdir(dir_folder)
if not dir_exists:
        os.mkdir(dir_folder)
        print("Making directory %s" % dir_folder)
else:
        print("Will store images in directory %s" % dir_folder)
```

```
Will store images in directory train
Will store images in directory test
Will store images in directory /content/train/bicycle
Will store images in directory /content/train/car
Will store images in directory /content/test/bicycle
Will store images in directory /content/test/car
```

[76]:
```python
keyword = 'bicycle'
dir_name = 'bicycle'
photos = flickr.walk(text=keyword, tag_mode='all', tags=keyword,extras='url_c',\
                     sort='relevance',per_page=100)

nimage = 1300
i = 0
nrow = 224
ncol = 224
for photo in photos:
    url=photo.get('url_c')
    if not (url is None):
```

```python
        # Create a file from the URL
        # This may only work in Python3
        response = requests.get(url)
        file = BytesIO(response.content)

        # Read image from file
        im = skimage.io.imread(file)

        # Resize images
        im1 = skimage.transform.resize(im,(nrow,ncol),mode='constant')

        # Convert to uint8, suppress the warning about the precision loss
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
            im2 = skimage.img_as_ubyte(im1)

        # Save the image
        if i< 1000:
          local_name = '{0:s}/{1:s}/{2:s}_{3:04d}.jpg'.
→format('train',dir_name,keyword, i)
        else:
          local_name = '{0:s}/{1:s}/{2:s}_{3:04d}.jpg'.
→format('test',dir_name,keyword, i)
        skimage.io.imsave(local_name, im2)
        if (i%100) == 1:
          print(local_name)
        i = i + 1
    if (i >= nimage):
        break
```

```
train/bicycle/bicycle_0001.jpg
train/bicycle/bicycle_0101.jpg
train/bicycle/bicycle_0201.jpg
train/bicycle/bicycle_0301.jpg
train/bicycle/bicycle_0401.jpg
train/bicycle/bicycle_0501.jpg
train/bicycle/bicycle_0601.jpg
train/bicycle/bicycle_0701.jpg
train/bicycle/bicycle_0801.jpg
train/bicycle/bicycle_0901.jpg
test/bicycle/bicycle_1001.jpg
test/bicycle/bicycle_1101.jpg
test/bicycle/bicycle_1201.jpg
```

Download cars

```
[77]: keyword = 'car'
      dir_name = 'car'
      photos = flickr.walk(text=keyword, tag_mode='all', tags=keyword,extras='url_c',\
                           sort='relevance',per_page=100)


      nimage = 1300
      i = 0
      nrow = 224
      ncol = 224
      for photo in photos:
          url=photo.get('url_c')
          if not (url is None):

              # Create a file from the URL
              # This may only work in Python3
              response = requests.get(url)
              file = BytesIO(response.content)

              # Read image from file
              im = skimage.io.imread(file)

              # Resize images
              im1 = skimage.transform.resize(im,(nrow,ncol),mode='constant')

              # Convert to uint8, suppress the warning about the precision loss
              with warnings.catch_warnings():
                  warnings.simplefilter("ignore")
                  im2 = skimage.img_as_ubyte(im1)

              # Save the image
              if i< 1000:
                local_name = '{0:s}/{1:s}/{2:s}_{3:04d}.jpg'.
      →format('train',dir_name,keyword, i)
              else:
                local_name = '{0:s}/{1:s}/{2:s}_{3:04d}.jpg'.
      →format('test',dir_name,keyword, i)
              skimage.io.imsave(local_name, im2)
              if (i%100) == 1:
                print(local_name)
              i = i + 1
          if (i >= nimage):
              break

      train/car/car_0001.jpg
      train/car/car_0101.jpg
      train/car/car_0201.jpg
      train/car/car_0301.jpg
      train/car/car_0401.jpg
```

```
train/car/car_0501.jpg
train/car/car_0601.jpg
train/car/car_0701.jpg
train/car/car_0801.jpg
train/car/car_0901.jpg
test/car/car_1001.jpg
test/car/car_1101.jpg
test/car/car_1201.jpg
```

0.2   Loading a Pre-Trained Deep Network

We follow the VGG16 demo to load a pre-trained deep VGG16 network. First, run a command to verify your instance is connected to a GPU.

[324]:
```python
# TODO 1:
import tensorflow as tf
tf.test.is_gpu_available()
```

[324]: True

Now load the appropriate tensorflow packages.

[325]:
```python
from tensorflow.keras import applications
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense
```

We also load some standard packages.

[326]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

Clear the Keras session.

[327]:
```python
# TODO 2:
import tensorflow.keras.backend as K
K.clear_session()
```

Set the dimensions of the input image. The sizes below would work on a GPU machine. But, if you have a CPU image, you can use a smaller image size, like 64 x 64.

[328]:
```python
# TODO 3:  Set to smaller values if you are using a CPU.
# Otherwise, do not change this code.
nrow = 150
ncol = 150
```

Now we follow the VGG16 demo and load the deep VGG16 network. Alternatively, you can use any other pre-trained model in keras. When using the applications.VGG16 method you will need to: * Set include_top=False to not include the top layer * Set the image_shape based on the

7

above dimensions. Remember, `image_shape` should be `height x width x 3` since the images are color.

```
[329]: # TODO 4:  Load the VGG16 network
       # input_shape = ...
       # base_model = applications.VGG16(weights='imagenet', ...)
```

```
[330]: # Select 'xception' or 'vgg16'
       pre_trained = 'vgg16'

       # Load appropriate packages
       from tensorflow.keras.applications.xception import Xception
       from tensorflow.keras.applications.vgg16 import VGG16
       from tensorflow.keras.preprocessing import image
       if pre_trained == 'xception':
           from tensorflow.keras.applications.xception import decode_predictions,
        ↪preprocess_input
       elif pre_trained == 'vgg16':
           from tensorflow.keras.applications.vgg16 import decode_predictions,
        ↪preprocess_input
       else:
           raise Exception("Unknown model")
```

```
[331]: input_shape = (nrow,ncol,3)

       if pre_trained == 'xception':
           base_model = Xception(weights='imagenet',input_shape=input_shape,
        ↪include_top=False)
       else:
           base_model = VGG16(weights='imagenet', input_shape = input_shape,
        ↪include_top=False)
```

```
[332]: base_model.summary()
```

```
Model: "vgg16"

_____
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856
```

```
block2_conv2 (Conv2D)        (None, 75, 75, 128)        147584

block2_pool (MaxPooling2D)  (None, 37, 37, 128)  0

block3_conv1 (Conv2D)        (None, 37, 37, 256)        295168

block3_conv2 (Conv2D)        (None, 37, 37, 256)        590080

block3_conv3 (Conv2D)        (None, 37, 37, 256)        590080

block3_pool (MaxPooling2D)  (None, 18, 18, 256)  0

block4_conv1 (Conv2D)        (None, 18, 18, 512)        1180160

block4_conv2 (Conv2D)        (None, 18, 18, 512)        2359808

block4_conv3 (Conv2D)        (None, 18, 18, 512)        2359808

block4_pool (MaxPooling2D)  (None, 9, 9, 512)        0

block5_conv1 (Conv2D)        (None, 9, 9, 512)        2359808

block5_conv2 (Conv2D)        (None, 9, 9, 512)        2359808

block5_conv3 (Conv2D)        (None, 9, 9, 512)        2359808

block5_pool (MaxPooling2D)  (None, 4, 4, 512)        0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

_____
```

To create now new model, we create a Sequential model. Then, loop over the layers in `base_model.layers` and add each layer to the new model.

```
[333]: # Create a new model
       model = Sequential()

       # TODO 5:  Loop over base_model.layers and add each layer to model
       bn = []
       for layer in base_model.layers:
           bn.append(layer)
       model = Sequential(bn)
```

```
[334]: model.summary()
```

Model: "sequential_1"

```
----------------------------------------------------------------
Layer (type)                 Output Shape             Param #
================================================================
block1_conv1 (Conv2D)        (None, 150, 150, 64)     1792

block1_conv2 (Conv2D)        (None, 150, 150, 64)     36928

block1_pool (MaxPooling2D)   (None, 75, 75, 64)       0

block2_conv1 (Conv2D)        (None, 75, 75, 128)      73856

block2_conv2 (Conv2D)        (None, 75, 75, 128)      147584

block2_pool (MaxPooling2D)   (None, 37, 37, 128)      0

block3_conv1 (Conv2D)        (None, 37, 37, 256)      295168

block3_conv2 (Conv2D)        (None, 37, 37, 256)      590080

block3_conv3 (Conv2D)        (None, 37, 37, 256)      590080

block3_pool (MaxPooling2D)   (None, 18, 18, 256)      0

block4_conv1 (Conv2D)        (None, 18, 18, 512)      1180160

block4_conv2 (Conv2D)        (None, 18, 18, 512)      2359808

block4_conv3 (Conv2D)        (None, 18, 18, 512)      2359808

block4_pool (MaxPooling2D)   (None, 9, 9, 512)        0

block5_conv1 (Conv2D)        (None, 9, 9, 512)        2359808

block5_conv2 (Conv2D)        (None, 9, 9, 512)        2359808

block5_conv3 (Conv2D)        (None, 9, 9, 512)        2359808

block5_pool (MaxPooling2D)   (None, 4, 4, 512)        0

================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

----------------------------------------------------------------
```

Next, loop through the layers in `model`, and freeze each layer by setting `layer.trainable = False`. This way, you will not have to *re-train* any of the existing layers.

```
[335]: # TODO 6
       for layer in model.layers:
         layer.trainable = False
```

```
[336]: model.summary()
```

Model: "sequential_1"

```
-------------------------------------------------------------------
 Layer (type)              Output Shape              Param #
===================================================================
 block1_conv1 (Conv2D)     (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)     (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D) (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)     (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)     (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D) (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)     (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)     (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)     (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D) (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)     (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)     (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)     (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D) (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)     (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)     (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)     (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D) (None, 4, 4, 512)         0

===================================================================
Total params: 14,714,688
```

```
Trainable params: 0
Non-trainable params: 14,714,688

-------------------------------------------------------------------
```

Now, add the following layers to `model`: * A `Flatten()` layer which reshapes the outputs to a single channel. * A fully-connected layer with 256 output units and `relu` activation * A `Dropout(0.5)` layer. * A final fully-connected layer. Since this is a binary classification, there should be one output and `sigmoid` activation.

[337]: `model.output_shape`

[337]: `(None, 4, 4, 512)`

[338]:
```python
# TODO 7
# model.add(...)
# model.add(...)
model.add(Flatten())
model.add(Dense(units=256, activation='relu', name='denseRelu'))
model.add(Dropout(0.5))
model.add(Dense(units=1, activation='sigmoid', name='denseOutput'))
```

[339]: `model.output_shape`

[339]: `(None, 1)`

Print the model summary. This will display the number of trainable parameters vs. the non-trainable parameters.

[340]:
```python
# TODO 8
model.summary()
```

```
Model: "sequential_1"

-------------------------------------------------------------------
 Layer (type)                Output Shape              Param #
===================================================================
 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168
```

```
block3_conv2 (Conv2D)       (None, 37, 37, 256)      590080

block3_conv3 (Conv2D)       (None, 37, 37, 256)      590080

block3_pool (MaxPooling2D)  (None, 18, 18, 256)      0

block4_conv1 (Conv2D)       (None, 18, 18, 512)      1180160

block4_conv2 (Conv2D)       (None, 18, 18, 512)      2359808

block4_conv3 (Conv2D)       (None, 18, 18, 512)      2359808

block4_pool (MaxPooling2D)  (None, 9, 9, 512)        0

block5_conv1 (Conv2D)       (None, 9, 9, 512)        2359808

block5_conv2 (Conv2D)       (None, 9, 9, 512)        2359808

block5_conv3 (Conv2D)       (None, 9, 9, 512)        2359808

block5_pool (MaxPooling2D)  (None, 4, 4, 512)        0

flatten (Flatten)           (None, 8192)             0

denseRelu (Dense)           (None, 256)              2097408

dropout (Dropout)           (None, 256)              0

denseOutput (Dense)         (None, 1)                257

=================================================================
Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688

-----------------------------------------------------------------
```

0.3   Using Generators to Load Data

Up to now, the training data has been represented in a large matrix. This is not possible for image data when the datasets are very large. For these applications, the `keras` package provides a `ImageDataGenerator` class that can fetch images on the fly from a directory of images. Using multi-threading, training can be performed on one mini-batch while the image reader can read files for the next mini-batch. The code below creates an `ImageDataGenerator` for the training data. In addition to the reading the files, the `ImageDataGenerator` creates random deformations of the image to expand the total dataset size. When the training data is limited, using data augmentation is very important.

```
[341]: train_data_dir = './train'
       batch_size = 32
       train_datagen = ImageDataGenerator(rescale=1./255,
                                          shear_range=0.2,
                                          zoom_range=0.2,
                                          horizontal_flip=True)
       train_generator = train_datagen.flow_from_directory(
                           train_data_dir,
                           target_size=(nrow,ncol),
                           batch_size=batch_size,
                           class_mode='binary')
```

Found 2000 images belonging to 2 classes.

Now, create a similar `test_generator` for the test data.

```
[342]: # TODO 9
       # test_generator = ...
       test_data_dir = './test'
       batch_size = 32
       test_datagen = ImageDataGenerator(rescale=1./255,
                                         shear_range=0.2,
                                         zoom_range=0.2,
                                         horizontal_flip=True)
       test_generator = test_datagen.flow_from_directory(
                           test_data_dir,
                           target_size=(nrow,ncol),
                           batch_size=batch_size,
                           class_mode='binary')
```

Found 600 images belonging to 2 classes.

The following function displays images that will be useful below.

```
[343]: # Display the image
       def disp_image(im):
           if (len(im.shape) == 2):
               # Gray scale image
               plt.imshow(im, cmap='gray')
           else:
               # Color image.
               im1 = (im-np.min(im))/(np.max(im)-np.min(im))*255
               im1 = im1.astype(np.uint8)
               plt.imshow(im1)

           # Remove axis ticks
           plt.xticks([])
           plt.yticks([])
```

To see how the `train_generator` works, use the `train_generator.next()` method to get a mini-

batch of data `X,y`. Display the first 8 images in this mini-batch and label the image with the class label. You should see that bicycles have `y=0` and cars have `y=1`.

```python
[344]:  # TODO 10
        plt.figure(figsize=(20,20))
        nplot = 8
        for i in range(nplot):
            plt.subplot(1,nplot,i+1)
            X,y = train_generator.next()
            disp_image(X[i])
            plt.title('y= '+str(int(y[i])))
```
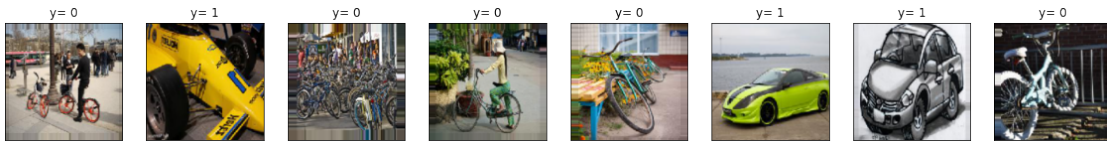

0.4 Train the Model

Compile the model. Select the correct `loss` function, `optimizer` and `metrics`. Remember that we are performing binary classification.

```python
[345]:  # TODO 11
        # model.compile(...)

        opt = optimizers.Adam(lr=0.001)
        model.compile(optimizer=opt,
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

When using an `ImageDataGenerator`, we have to set two parameters manually: * `steps_per_epoch = training data size // batch_size` * `validation_steps = test data size // batch_size`

We can obtain the training and test data size from `train_generator.n` and `test_generator.n`, respectively.

```python
[346]:  # TODO 12
        steps_per_epoch = train_generator.n // batch_size
        validation_steps = test_generator.n // batch_size
```

```python
[347]:  print(steps_per_epoch, validation_steps)
```

```
62 18
```

Now, we run the fit. If you are using a CPU on a regular laptop, each epoch will take about 3-4 minutes, so you should be able to finish 5 epochs or so within 20 minutes. On a reasonable GPU, even with the larger images, it will take about 10 seconds per epoch. * If you use (nrow,ncol) = (64,64) images, you should get around 90% accuracy after 5 epochs. * If you use (nrow,ncol) = (150,150) images, you should get around 96% accuracy after 5 epochs. But, this will need a GPU.

You will get full credit for either version. With more epochs, you may get slightly higher, but you will have to play with the damping.

Remember to record the history of the fit, so that you can plot the training and validation accuracy curve.

```
[348]: nepochs = 5   # Number of epochs

       # Call the fit_generator function
       hist = model.fit_generator(
           train_generator,
           steps_per_epoch=steps_per_epoch,
           epochs=nepochs,
           validation_data=test_generator,
           validation_steps=validation_steps)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  if __name__ == '__main__':

Epoch 1/5
62/62 [==============================] - 22s 336ms/step - loss: 0.2970 -
accuracy: 0.8913 - val_loss: 0.0865 - val_accuracy: 0.9740
Epoch 2/5
62/62 [==============================] - 20s 326ms/step - loss: 0.0770 -
accuracy: 0.9736 - val_loss: 0.0853 - val_accuracy: 0.9722
Epoch 3/5
62/62 [==============================] - 20s 326ms/step - loss: 0.0817 -
accuracy: 0.9700 - val_loss: 0.0806 - val_accuracy: 0.9844
Epoch 4/5
62/62 [==============================] - 20s 326ms/step - loss: 0.0646 -
accuracy: 0.9817 - val_loss: 0.0723 - val_accuracy: 0.9809
Epoch 5/5
62/62 [==============================] - 20s 324ms/step - loss: 0.0433 -
accuracy: 0.9853 - val_loss: 0.0908 - val_accuracy: 0.9740
```
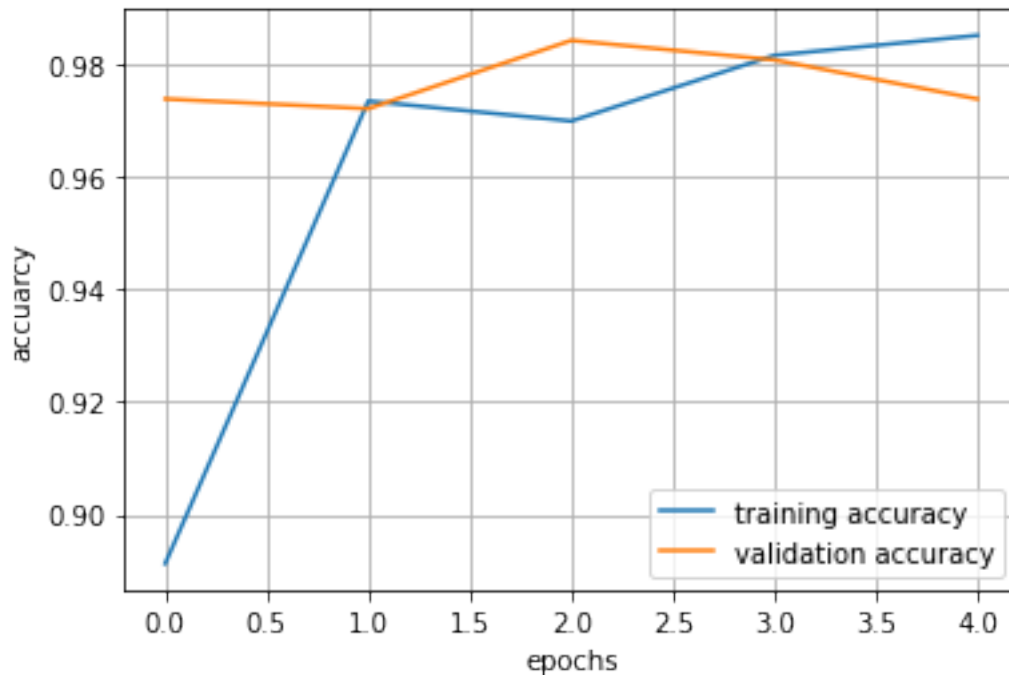
```
[357]: # Plot the training accuracy and validation accuracy curves on the same figure.

       # TODO 13
       tr_accuracy = hist.history['accuracy']
       val_accuracy = hist.history['val_accuracy']
```

```
plt.plot(tr_accuracy)
plt.plot(val_accuracy)
plt.grid()
plt.xlabel('epochs')
plt.ylabel('accuarcy')
plt.legend(['training accuracy', 'validation accuracy'])
```

[357]: <matplotlib.legend.Legend at 0x7f971024e850>

0.5   Plotting the Error Images

Now try to plot some images that were in error:

- Generate a mini-batch `Xts,yts` from the `test_generator.next()` method
- Get the class probabilities using the `model.predict( )` method and compute predicted labels `yhat`.
- Get the images where `yts[i] ~= yhat[i]`.
- If you did not get any prediction error in one minibatch, run it multiple times.
- After you a get a few error images (say 4-8), plot the error images with the true labels and class probabilities predicted by the classifie

[350]: ```
TODO 14
Xts, yts = test_generator.next()
yhat = model.predict(Xts)
```

```
WARNING:tensorflow:5 out of the last 22 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f9714a03dd0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

[351]:
```python
almost_same = []
for i in range(len(yhat)):
  if abs(yhat[i]-yts[i])<=0.1:
    almost_same.append(Xts[i])
```

[352]:
```python
len(almost_same)
```

[352]: 28

[353]:
```python
len(yhat)
```

[353]: 32

[354]:
```python
not_same = []
yts_not_same = []
yhat_not_same = []
for i in range(len(yhat)):
  if abs(yhat[i]-yts[i])>0.1:
    not_same.append(Xts[i])
    yts_not_same.append(yts[i])
    yhat_not_same.append(yhat[i])
```

[355]:
```python
len(not_same)
```

[355]: 4

[356]:
```python
plt.figure(figsize=(20,20))
for i in range(len(not_same)):
  plt.subplot(1,nplot,i+1)
  disp_image(not_same[i])
  plt.title('true: '+str(int(yts_not_same[i]))+' predict:␣
  ↪'+str(round(yhat_not_same[i][0],3)))
```

true: 0 predict: 0.119   true: 1 predict: 0.204   true: 1 predict: 0.63   true: 1 predict: 0.617

[356]: