Md Marufi Rahman

Thasina Tabassum

Md Farhad Mokhter

Double-click (or enter) to edit

Transfer learning with a pretrained ConvNet

 View on TensorFlow.org  Run in Google Colab  View source on GitHub  Download notebook

In this tutorial, you will learn how to classify images of cats and dogs by using transfer learning from a

A pre-trained model is a saved network that was previously trained on a large dataset, typically on a la either use the pretrained model as is or use transfer learning to customize this model to a given task.

The intuition behind transfer learning for image classification is that if a model is trained on a large ar effectively serve as a generic model of the visual world. You can then take advantage of these learnec from scratch by training a large model on a large dataset.

In this notebook, you will try two ways to customize a pretrained model:

1. Feature Extraction: Use the representations learned by a previous network to extract meaningful add a new classifier, which will be trained from scratch, on top of the pretrained model so that y learned previously for the dataset.

   You do not need to (re)train the entire model. The base convolutional network already contains f classifying pictures. However, the final, classification part of the pretrained model is specific to t subsequently specific to the set of classes on which the model was trained.

2. Fine-Tuning: Unfreeze a few of the top layers of a frozen model base and jointly train both the ne layers of the base model. This allows us to "fine-tune" the higher-order feature representations i more relevant for the specific task.

You will follow the general machine learning workflow.

1. Examine and understand the data
2. Build an input pipeline, in this case using Keras ImageDataGenerator
3. Compose the model
    - Load in the pretrained base model (and pretrained weights)
    - Stack the classification layers on top
4. Train the model
5. Evaluate model

Md Marufi Rahman

Thasina Tabassum

Md Farhad Mokhter

```
import os

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf
```

Data preprocessing

Data download

Use TensorFlow Datasets to load the cats and dogs dataset.

This `tfds` package is the easiest way to load pre-defined data. If you have your own data, and are inte
TensorFlow see loading image data

```
import tensorflow_datasets as tfds
tfds.disable_progress_bar()
```

The `tfds.load` method downloads and caches the data, and returns a `tf.data.Dataset` object. Thes
methods for manipulating data and piping it into your model.

Since `"cats_vs_dogs"` doesn't define standard splits, use the subsplit feature to divide it into (train, va
the data respectively.

```
(raw_train, raw_validation, raw_test), metadata = tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
```

The resulting `tf.data.Dataset` objects contain `(image, label)` pairs where the images have variabl
scalar.

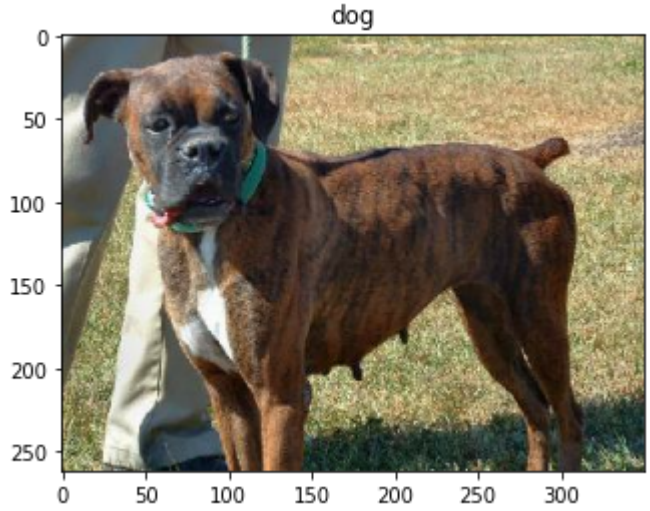
```
print(raw_train)
print(raw_validation)
print(raw_test)
```

```
<DatasetV1Adapter shapes: ((None, None, 3), ()), types: (tf.uint8, tf.int64)>
<DatasetV1Adapter shapes: ((None, None, 3), ()), types: (tf.uint8, tf.int64)>
<DatasetV1Adapter shapes: ((None, None, 3), ()), types: (tf.uint8, tf.int64)>
```

Show the first two images and labels from the training set:

```
get_label_name = metadata.features['label'].int2str

for image, label in raw_train.take(2):
  plt.figure()
  plt.imshow(image)
  plt.title(get_label_name(label))
```



Format the Data

Use the `tf.image` module to format the images for the task.

Resize the images to a fixed input size, and rescale the input channels to a range of `[-1,1]`

```
IMG_SIZE = 160 # All images will be resized to 160x160

def format_example(image, label):
  image = tf.cast(image, tf.float32)
  image = (image/127.5) - 1
  image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
  return image, label
```

Apply this function to each item in the dataset using the map method:

```
train = raw_train.map(format_example)
validation = raw_validation.map(format_example)
test = raw_test.map(format_example)
```

Now shuffle and batch the data.

```
BATCH_SIZE = 32
SHUFFLE_BUFFER_SIZE = 1000

train_batches = train.shuffle(SHUFFLE_BUFFER_SIZE).batch(BATCH_SIZE)
validation_batches = validation.batch(BATCH_SIZE)
test_batches = test.batch(BATCH_SIZE)
```

Inspect a batch of data:

```
for image_batch, label_batch in train_batches.take(1):
   pass

image_batch.shape
```

```
TensorShape([32, 160, 160, 3])
```

Create the base model from the pre-trained convnets

You will create the base model from the **MobileNet V2** model developed at Google. This is pre-trained consisting of 1.4M images and 1000 classes. ImageNet is a research training dataset with a wide var `syringe`. This base of knowledge will help us classify cats and dogs from our specific dataset.

First, you need to pick which layer of MobileNet V2 you will use for feature extraction. The very last cl diagrams of machine learning models go from bottom to top) is not very useful. Instead, you will follo very last layer before the flatten operation. This layer is called the "bottleneck layer". The bottleneck la compared to the final/top layer.

First, instantiate a MobileNet V2 model pre-loaded with weights trained on ImageNet. By specifying th network that doesn't include the classification layers at the top, which is ideal for feature extraction.

```
IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)

# Create the base model from the pre-trained model MobileNet V2
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

This feature extractor converts each `160x160x3` image into a `5x5x1280` block of features. See what it

```
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 5, 5, 1280)
```

Feature extraction

In this step, you will freeze the convolutional base created from the previous step and to use as a feat
classifier on top of it and train the top-level classifier.

Freeze the convolutional base

It is important to freeze the convolutional base before you compile and train the model. Freezing (by s
weights in a given layer from being updated during training. MobileNet V2 has many layers, so setting
will freeze all the layers.

```
base_model.trainable = False

# Let's take a look at the base model architecture
base_model.summary()
```

```
Model: "mobilenetv2_1.00_160"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 160, 160, 3) 0
__________________________________________________________________________________________________
Conv1_pad (ZeroPadding2D)       (None, 161, 161, 3)  0           input_2[0][0]
__________________________________________________________________________________________________
Conv1 (Conv2D)                  (None, 80, 80, 32)   864         Conv1_pad[0][0]
__________________________________________________________________________________________________
bn_Conv1 (BatchNormalization)   (None, 80, 80, 32)   128         Conv1[0][0]
__________________________________________________________________________________________________
Conv1_relu (ReLU)               (None, 80, 80, 32)   0           bn_Conv1[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise (Depthw (None, 80, 80, 32)   288         Conv1_relu[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise_BN (Bat (None, 80, 80, 32)   128         expanded_conv_depthwise
__________________________________________________________________________________________________
expanded_conv_depthwise_relu (R (None, 80, 80, 32)   0           expanded_conv_depthwise
__________________________________________________________________________________________________
expanded_conv_project (Conv2D)  (None, 80, 80, 16)   512         expanded_conv_depthwise
__________________________________________________________________________________________________
expanded_conv_project_BN (Batch (None, 80, 80, 16)   64          expanded_conv_project[0
__________________________________________________________________________________________________
block_1_expand (Conv2D)         (None, 80, 80, 96)   1536        expanded_conv_project_B
__________________________________________________________________________________________________
block_1_expand_BN (BatchNormali (None, 80, 80, 96)   384         block_1_expand[0][0]
__________________________________________________________________________________________________
block_1_expand_relu (ReLU)      (None, 80, 80, 96)   0           block_1_expand_BN[0][0]
__________________________________________________________________________________________________
block_1_pad (ZeroPadding2D)     (None, 81, 81, 96)   0           block_1_expand_relu[0][
__________________________________________________________________________________________________
block_1_depthwise (DepthwiseCon (None, 40, 40, 96)   864         block_1_pad[0][0]
__________________________________________________________________________________________________
block_1_depthwise_BN (BatchNorm (None, 40, 40, 96)   384         block_1_depthwise[0][0]
__________________________________________________________________________________________________
block_1_depthwise_relu (ReLU)   (None, 40, 40, 96)   0           block_1_depthwise_BN[0]
__________________________________________________________________________________________________
block_1_project (Conv2D)        (None, 40, 40, 24)   2304        block_1_depthwise_relu[
__________________________________________________________________________________________________
block_1_project_BN (BatchNormal (None, 40, 40, 24)   96          block_1_project[0][0]
__________________________________________________________________________________________________
block_2_expand (Conv2D)         (None, 40, 40, 144)  3456        block_1_project_BN[0][0
__________________________________________________________________________________________________
block_2_expand_BN (BatchNormali (None, 40, 40, 144)  576         block_2_expand[0][0]
__________________________________________________________________________________________________
block_2_expand_relu (ReLU)      (None, 40, 40, 144)  0           block_2_expand_BN[0][0]
__________________________________________________________________________________________________
block_2_depthwise (DepthwiseCon (None, 40, 40, 144)  1296        block_2_expand_relu[0][
__________________________________________________________________________________________________
block_2_depthwise_BN (BatchNorm (None, 40, 40, 144)  576         block_2_depthwise[0][0]
__________________________________________________________________________________________________
block_2_depthwise_relu (ReLU)   (None, 40, 40, 144)  0           block_2_depthwise_BN[0]
__________________________________________________________________________________________________
block_2_project (Conv2D)        (None, 40, 40, 24)   3456        block_2_depthwise_relu[
__________________________________________________________________________________________________
block_2_project_BN (BatchNormal (None, 40, 40, 24)   96          block_2_project[0][0]
```

block_2_add (Add)	(None, 40, 40, 24)	0	block_1_project_BN[0][0] block_2_project_BN[0][0]
block_3_expand (Conv2D)	(None, 40, 40, 144)	3456	block_2_add[0][0]
block_3_expand_BN (BatchNormali	(None, 40, 40, 144)	576	block_3_expand[0][0]
block_3_expand_relu (ReLU)	(None, 40, 40, 144)	0	block_3_expand_BN[0][0]
block_3_pad (ZeroPadding2D)	(None, 41, 41, 144)	0	block_3_expand_relu[0][
block_3_depthwise (DepthwiseCon	(None, 20, 20, 144)	1296	block_3_pad[0][0]
block_3_depthwise_BN (BatchNorm	(None, 20, 20, 144)	576	block_3_depthwise[0][0]
block_3_depthwise_relu (ReLU)	(None, 20, 20, 144)	0	block_3_depthwise_BN[0]
block_3_project (Conv2D)	(None, 20, 20, 32)	4608	block_3_depthwise_relu[
block_3_project_BN (BatchNormal	(None, 20, 20, 32)	128	block_3_project[0][0]
block_4_expand (Conv2D)	(None, 20, 20, 192)	6144	block_3_project_BN[0][0
block_4_expand_BN (BatchNormali	(None, 20, 20, 192)	768	block_4_expand[0][0]
block_4_expand_relu (ReLU)	(None, 20, 20, 192)	0	block_4_expand_BN[0][0]
block_4_depthwise (DepthwiseCon	(None, 20, 20, 192)	1728	block_4_expand_relu[0][
block_4_depthwise_BN (BatchNorm	(None, 20, 20, 192)	768	block_4_depthwise[0][0]
block_4_depthwise_relu (ReLU)	(None, 20, 20, 192)	0	block_4_depthwise_BN[0]
block_4_project (Conv2D)	(None, 20, 20, 32)	6144	block_4_depthwise_relu[
block_4_project_BN (BatchNormal	(None, 20, 20, 32)	128	block_4_project[0][0]
block_4_add (Add)	(None, 20, 20, 32)	0	block_3_project_BN[0][0] block_4_project_BN[0][0]
block_5_expand (Conv2D)	(None, 20, 20, 192)	6144	block_4_add[0][0]
block_5_expand_BN (BatchNormali	(None, 20, 20, 192)	768	block_5_expand[0][0]
block_5_expand_relu (ReLU)	(None, 20, 20, 192)	0	block_5_expand_BN[0][0]
block_5_depthwise (DepthwiseCon	(None, 20, 20, 192)	1728	block_5_expand_relu[0][
block_5_depthwise_BN (BatchNorm	(None, 20, 20, 192)	768	block_5_depthwise[0][0]
block_5_depthwise_relu (ReLU)	(None, 20, 20, 192)	0	block_5_depthwise_BN[0]
block_5_project (Conv2D)	(None, 20, 20, 32)	6144	block_5_depthwise_relu[
block_5_project_BN (BatchNormal	(None, 20, 20, 32)	128	block_5_project[0][0]
block_5_add (Add)	(None, 20, 20, 32)	0	block_4_add[0][0]

			block_5_project_BN[0][0
block_6_expand (Conv2D)	(None, 20, 20, 192)	6144	block_5_add[0][0]
block_6_expand_BN (BatchNormali	(None, 20, 20, 192)	768	block_6_expand[0][0]
block_6_expand_relu (ReLU)	(None, 20, 20, 192)	0	block_6_expand_BN[0][0]
block_6_pad (ZeroPadding2D)	(None, 21, 21, 192)	0	block_6_expand_relu[0][
block_6_depthwise (DepthwiseCon	(None, 10, 10, 192)	1728	block_6_pad[0][0]
block_6_depthwise_BN (BatchNorm	(None, 10, 10, 192)	768	block_6_depthwise[0][0]
block_6_depthwise_relu (ReLU)	(None, 10, 10, 192)	0	block_6_depthwise_BN[0]
block_6_project (Conv2D)	(None, 10, 10, 64)	12288	block_6_depthwise_relu[
block_6_project_BN (BatchNormal	(None, 10, 10, 64)	256	block_6_project[0][0]
block_7_expand (Conv2D)	(None, 10, 10, 384)	24576	block_6_project_BN[0][0
block_7_expand_BN (BatchNormali	(None, 10, 10, 384)	1536	block_7_expand[0][0]
block_7_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_7_expand_BN[0][0]
block_7_depthwise (DepthwiseCon	(None, 10, 10, 384)	3456	block_7_expand_relu[0][
block_7_depthwise_BN (BatchNorm	(None, 10, 10, 384)	1536	block_7_depthwise[0][0]
block_7_depthwise_relu (ReLU)	(None, 10, 10, 384)	0	block_7_depthwise_BN[0]
block_7_project (Conv2D)	(None, 10, 10, 64)	24576	block_7_depthwise_relu[
block_7_project_BN (BatchNormal	(None, 10, 10, 64)	256	block_7_project[0][0]
block_7_add (Add)	(None, 10, 10, 64)	0	block_6_project_BN[0][0 block_7_project_BN[0][0
block_8_expand (Conv2D)	(None, 10, 10, 384)	24576	block_7_add[0][0]
block_8_expand_BN (BatchNormali	(None, 10, 10, 384)	1536	block_8_expand[0][0]
block_8_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_8_expand_BN[0][0]
block_8_depthwise (DepthwiseCon	(None, 10, 10, 384)	3456	block_8_expand_relu[0][
block_8_depthwise_BN (BatchNorm	(None, 10, 10, 384)	1536	block_8_depthwise[0][0]
block_8_depthwise_relu (ReLU)	(None, 10, 10, 384)	0	block_8_depthwise_BN[0]
block_8_project (Conv2D)	(None, 10, 10, 64)	24576	block_8_depthwise_relu[
block_8_project_BN (BatchNormal	(None, 10, 10, 64)	256	block_8_project[0][0]
block_8_add (Add)	(None, 10, 10, 64)	0	block_7_add[0][0] block_8_project_BN[0][0

block_9_expand (Conv2D)	(None, 10, 10, 384)	24576	block_8_add[0][0]
block_9_expand_BN (BatchNormali	(None, 10, 10, 384)	1536	block_9_expand[0][0]
block_9_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_9_expand_BN[0][0]
block_9_depthwise (DepthwiseCon	(None, 10, 10, 384)	3456	block_9_expand_relu[0][
block_9_depthwise_BN (BatchNorm	(None, 10, 10, 384)	1536	block_9_depthwise[0][0]
block_9_depthwise_relu (ReLU)	(None, 10, 10, 384)	0	block_9_depthwise_BN[0]
block_9_project (Conv2D)	(None, 10, 10, 64)	24576	block_9_depthwise_relu[
block_9_project_BN (BatchNormal	(None, 10, 10, 64)	256	block_9_project[0][0]
block_9_add (Add)	(None, 10, 10, 64)	0	block_8_add[0][0] block_9_project_BN[0][0
block_10_expand (Conv2D)	(None, 10, 10, 384)	24576	block_9_add[0][0]
block_10_expand_BN (BatchNormal	(None, 10, 10, 384)	1536	block_10_expand[0][0]
block_10_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_10_expand_BN[0][0
block_10_depthwise (DepthwiseCo	(None, 10, 10, 384)	3456	block_10_expand_relu[0]
block_10_depthwise_BN (BatchNor	(None, 10, 10, 384)	1536	block_10_depthwise[0][0
block_10_depthwise_relu (ReLU)	(None, 10, 10, 384)	0	block_10_depthwise_BN[0
block_10_project (Conv2D)	(None, 10, 10, 96)	36864	block_10_depthwise_relu
block_10_project_BN (BatchNorma	(None, 10, 10, 96)	384	block_10_project[0][0]
block_11_expand (Conv2D)	(None, 10, 10, 576)	55296	block_10_project_BN[0][
block_11_expand_BN (BatchNormal	(None, 10, 10, 576)	2304	block_11_expand[0][0]
block_11_expand_relu (ReLU)	(None, 10, 10, 576)	0	block_11_expand_BN[0][0
block_11_depthwise (DepthwiseCo	(None, 10, 10, 576)	5184	block_11_expand_relu[0]
block_11_depthwise_BN (BatchNor	(None, 10, 10, 576)	2304	block_11_depthwise[0][0
block_11_depthwise_relu (ReLU)	(None, 10, 10, 576)	0	block_11_depthwise_BN[0
block_11_project (Conv2D)	(None, 10, 10, 96)	55296	block_11_depthwise_relu
block_11_project_BN (BatchNorma	(None, 10, 10, 96)	384	block_11_project[0][0]
block_11_add (Add)	(None, 10, 10, 96)	0	block_10_project_BN[0][block_11_project_BN[0][
block_12_expand (Conv2D)	(None, 10, 10, 576)	55296	block_11_add[0][0]
block_12_expand_BN (BatchNormal	(None, 10, 10, 576)	2304	block_12_expand[0][0]

block_12_expand_relu (ReLU)	(None, 10, 10, 576)	0	block_12_expand_BN[0][0
block_12_depthwise (DepthwiseCo	(None, 10, 10, 576)	5184	block_12_expand_relu[0]
block_12_depthwise_BN (BatchNor	(None, 10, 10, 576)	2304	block_12_depthwise[0][0
block_12_depthwise_relu (ReLU)	(None, 10, 10, 576)	0	block_12_depthwise_BN[0
block_12_project (Conv2D)	(None, 10, 10, 96)	55296	block_12_depthwise_relu
block_12_project_BN (BatchNorma	(None, 10, 10, 96)	384	block_12_project[0][0]
block_12_add (Add)	(None, 10, 10, 96)	0	block_11_add[0][0] block_12_project_BN[0][
block_13_expand (Conv2D)	(None, 10, 10, 576)	55296	block_12_add[0][0]
block_13_expand_BN (BatchNormal	(None, 10, 10, 576)	2304	block_13_expand[0][0]
block_13_expand_relu (ReLU)	(None, 10, 10, 576)	0	block_13_expand_BN[0][0
block_13_pad (ZeroPadding2D)	(None, 11, 11, 576)	0	block_13_expand_relu[0]
block_13_depthwise (DepthwiseCo	(None, 5, 5, 576)	5184	block_13_pad[0][0]
block_13_depthwise_BN (BatchNor	(None, 5, 5, 576)	2304	block_13_depthwise[0][0
block_13_depthwise_relu (ReLU)	(None, 5, 5, 576)	0	block_13_depthwise_BN[0
block_13_project (Conv2D)	(None, 5, 5, 160)	92160	block_13_depthwise_relu
block_13_project_BN (BatchNorma	(None, 5, 5, 160)	640	block_13_project[0][0]
block_14_expand (Conv2D)	(None, 5, 5, 960)	153600	block_13_project_BN[0][
block_14_expand_BN (BatchNormal	(None, 5, 5, 960)	3840	block_14_expand[0][0]
block_14_expand_relu (ReLU)	(None, 5, 5, 960)	0	block_14_expand_BN[0][0
block_14_depthwise (DepthwiseCo	(None, 5, 5, 960)	8640	block_14_expand_relu[0]
block_14_depthwise_BN (BatchNor	(None, 5, 5, 960)	3840	block_14_depthwise[0][0
block_14_depthwise_relu (ReLU)	(None, 5, 5, 960)	0	block_14_depthwise_BN[0
block_14_project (Conv2D)	(None, 5, 5, 160)	153600	block_14_depthwise_relu
block_14_project_BN (BatchNorma	(None, 5, 5, 160)	640	block_14_project[0][0]
block_14_add (Add)	(None, 5, 5, 160)	0	block_13_project_BN[0][block_14_project_BN[0][
block_15_expand (Conv2D)	(None, 5, 5, 960)	153600	block_14_add[0][0]
block_15_expand_BN (BatchNormal	(None, 5, 5, 960)	3840	block_15_expand[0][0]
block_15_expand_relu (ReLU)	(None, 5, 5, 960)	0	block_15_expand_BN[0][0

```
block_15_depthwise (DepthwiseCo (None, 5, 5, 960)    8640        block_15_expand_relu[0]
block_15_depthwise_BN (BatchNor (None, 5, 5, 960)    3840        block_15_depthwise[0][0
block_15_depthwise_relu (ReLU)  (None, 5, 5, 960)    0           block_15_depthwise_BN[0
block_15_project (Conv2D)       (None, 5, 5, 160)    153600      block_15_depthwise_relu
block_15_project_BN (BatchNorma (None, 5, 5, 160)    640         block_15_project[0][0]
block_15_add (Add)              (None, 5, 5, 160)    0           block_14_add[0][0]
                                                                 block_15_project_BN[0][
block_16_expand (Conv2D)        (None, 5, 5, 960)    153600      block_15_add[0][0]
block_16_expand_BN (BatchNormal (None, 5, 5, 960)    3840        block_16_expand[0][0]
block_16_expand_relu (ReLU)     (None, 5, 5, 960)    0           block_16_expand_BN[0][0
block_16_depthwise (DepthwiseCo (None, 5, 5, 960)    8640        block_16_expand_relu[0]
block_16_depthwise_BN (BatchNor (None, 5, 5, 960)    3840        block_16_depthwise[0][0
block_16_depthwise_relu (ReLU)  (None, 5, 5, 960)    0           block_16_depthwise_BN[0
block_16_project (Conv2D)       (None, 5, 5, 320)    307200      block_16_depthwise_relu
block_16_project_BN (BatchNorma (None, 5, 5, 320)    1280        block_16_project[0][0]
Conv_1 (Conv2D)                 (None, 5, 5, 1280)   409600      block_16_project_BN[0][
Conv_1_bn (BatchNormalization)  (None, 5, 5, 1280)   5120        Conv_1[0][0]
out_relu (ReLU)                 (None, 5, 5, 1280)   0           Conv_1_bn[0][0]
==================================================================================================
Total params: 2,257,984
Trainable params: 0
Non-trainable params: 2,257,984
__________________________________________________________________________________________________
```

Add a classification head

To generate predictions from the block of features, average over the spatial `5x5` spatial locations, usi `tf.keras.layers.GlobalAveragePooling2D` layer to convert the features to a single 1280-element vec

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

Apply a `tf.keras.layers.Dense` layer to convert these features into a single prediction per image. Yo
because this prediction will be treated as a `logit`, or a raw prediction value. Positive numbers predict
0.

```
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 1)
```

Now stack the feature extractor, and these two layers using a `tf.keras.Sequential` model:

```
model = tf.keras.Sequential([
  base_model,
  global_average_layer,
  prediction_layer
])
```

Compile the model

You must compile the model before training it. Since there are two classes, use a binary cross-entropy
model provides a linear output.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
mobilenetv2_1.00_160 (Model) (None, 5, 5, 1280)        2257984   
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1280)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 1281      
=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

The 2.5M parameters in MobileNet are frozen, but there are 1.2K *trainable* parameters in the Dense la

```
len(model.trainable_variables)
```

```
2
```

Train the model

After training for 10 epochs, you should see ~96% accuracy.

```
initial_epochs = 10
validation_steps=20

loss0,accuracy0 = model.evaluate(validation_batches, steps = validation_steps)
```

```
20/20 [==============================] - 1s 56ms/step - loss: 0.9008 - accuracy: 0.4781
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 0.90
initial accuracy: 0.48
```

```
history = model.fit(train_batches,
                    epochs=initial_epochs,
                    validation_data=validation_batches)
```

```
Epoch 1/10
582/582 [==============================] - 46s 78ms/step - loss: 0.2287 - accuracy: 0.89
Epoch 2/10
582/582 [==============================] - 44s 76ms/step - loss: 0.0714 - accuracy: 0.97
Epoch 3/10
582/582 [==============================] - 45s 77ms/step - loss: 0.0572 - accuracy: 0.97
Epoch 4/10
582/582 [==============================] - 45s 78ms/step - loss: 0.0513 - accuracy: 0.98
Epoch 5/10
582/582 [==============================] - 45s 77ms/step - loss: 0.0479 - accuracy: 0.98
Epoch 6/10
582/582 [==============================] - 45s 78ms/step - loss: 0.0456 - accuracy: 0.98
Epoch 7/10
582/582 [==============================] - 45s 78ms/step - loss: 0.0440 - accuracy: 0.98
Epoch 8/10
582/582 [==============================] - 45s 78ms/step - loss: 0.0427 - accuracy: 0.98
Epoch 9/10
582/582 [==============================] - 45s 77ms/step - loss: 0.0417 - accuracy: 0.98
Epoch 10/10
582/582 [==============================] - 46s 79ms/step - loss: 0.0410 - accuracy: 0.98
```

Learning curves

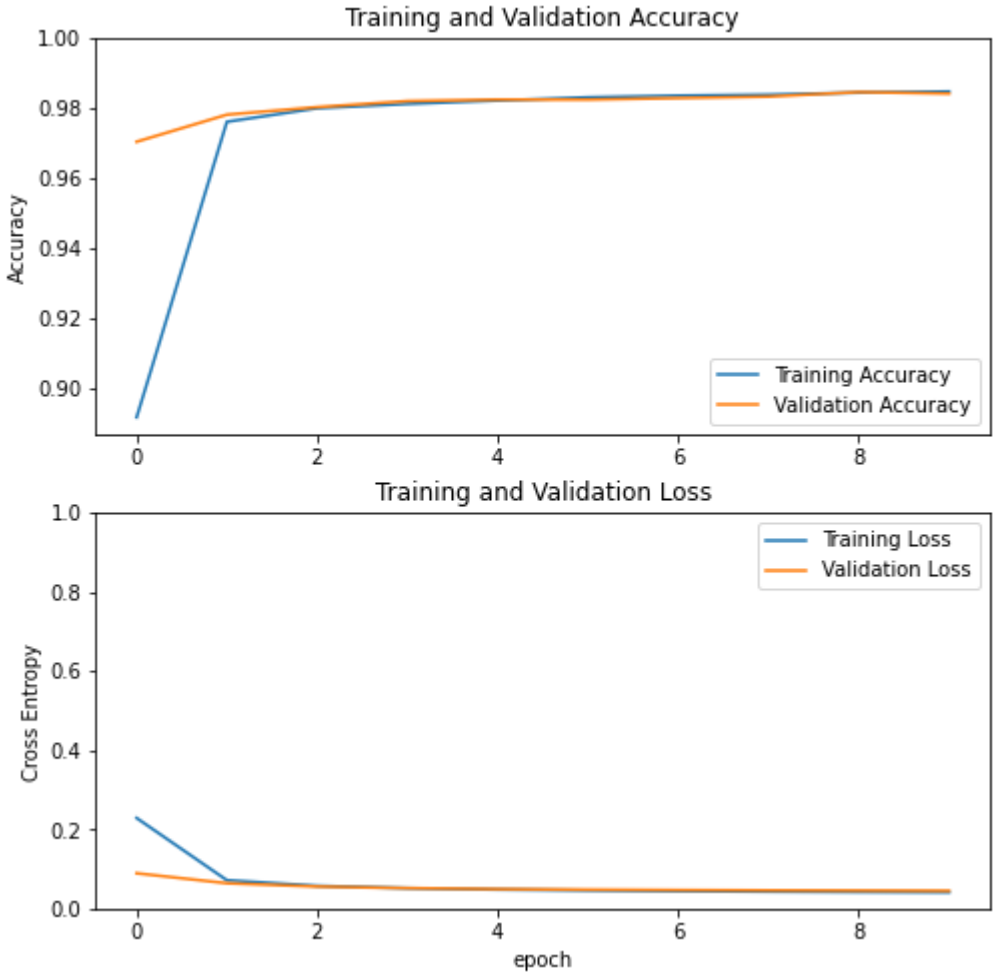
Let's take a look at the learning curves of the training and validation accuracy/loss when using the Mc extractor.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Note: If you are wondering why the validation metrics are clearly better than the training metrics, the r `tf.keras.layers.BatchNormalization` and `tf.keras.layers.Dropout` affect accuracy during training validation loss.

To a lesser extent, it is also because training metrics report the average for an epoch, while validation validation metrics see a model that has trained slightly longer.

Fine tuning

In the feature extraction experiment, you were only training a few layers on top of an MobileNet V2 ba network were **not** updated during training.

One way to increase performance even further is to train (or "fine-tune") the weights of the top layers training of the classifier you added. The training process will force the weights to be tuned from gene specifically with the dataset.

Note: This should only be attempted after you have trained the top-level classifier with the pre-trained randomly initialized classifier on top of a pre-trained model and attempt to train all layers jointly, the m too large (due to the random weights from the classifier) and your pre-trained model will forget what i

Also, you should try to fine-tune a small number of top layers rather than the whole MobileNet model. higher up a layer is, the more specialized it is. The first few layers learn very simple and generic featur images. As you go higher up, the features are increasingly more specific to the dataset on which the r is to adapt these specialized features to work with the new dataset, rather than overwrite the generic

Un-freeze the top layers of the model

All you need to do is unfreeze the `base_model` and set the bottom layers to be un-trainable. Then, you for these changes to take effect), and resume training.

```
base_model.trainable = True


# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  155
```

Compile the model

Compile the model using a much lower learning rate.

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(lr=base_learning_rate/10),
              metrics=['accuracy'])


model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
mobilenetv2_1.00_160 (Model) (None, 5, 5, 1280)        2257984   
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1280)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 1281      
=================================================================
Total params: 2,259,265
Trainable params: 1,863,873
Non-trainable params: 395,392
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
58
```

Continue training the model

If you trained to convergence earlier, this step will improve your accuracy by a few percentage points.

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_batches,
                         epochs=total_epochs,
                         initial_epoch =  history.epoch[-1],
                         validation_data=validation_batches)
```

```
Epoch 10/20
582/582 [==============================] - 50s 87ms/step - loss: 0.1239 - accuracy: 0.95
Epoch 11/20
582/582 [==============================] - 50s 85ms/step - loss: 0.0660 - accuracy: 0.97
Epoch 12/20
582/582 [==============================] - 50s 86ms/step - loss: 0.0504 - accuracy: 0.98
Epoch 13/20
582/582 [==============================] - 50s 86ms/step - loss: 0.0347 - accuracy: 0.98
Epoch 14/20
582/582 [==============================] - 50s 86ms/step - loss: 0.0316 - accuracy: 0.98
Epoch 15/20
582/582 [==============================] - 51s 87ms/step - loss: 0.0240 - accuracy: 0.99
Epoch 16/20
582/582 [==============================] - 50s 86ms/step - loss: 0.0215 - accuracy: 0.99
Epoch 17/20
582/582 [==============================] - 50s 86ms/step - loss: 0.0146 - accuracy: 0.99
Epoch 18/20
582/582 [==============================] - 51s 87ms/step - loss: 0.0104 - accuracy: 0.99
Epoch 19/20
582/582 [==============================] - 50s 86ms/step - loss: 0.0094 - accuracy: 0.99
Epoch 20/20
582/582 [==============================] - 51s 87ms/step - loss: 0.0081 - accuracy: 0.99
```

Let's take a look at the learning curves of the training and validation accuracy/loss when fine-tuning th base model and training the classifier on top of it. The validation loss is much higher than the training

You may also get some overfitting as the new training set is relatively small and similar to the original

After fine tuning the model nearly reaches 98% accuracy.

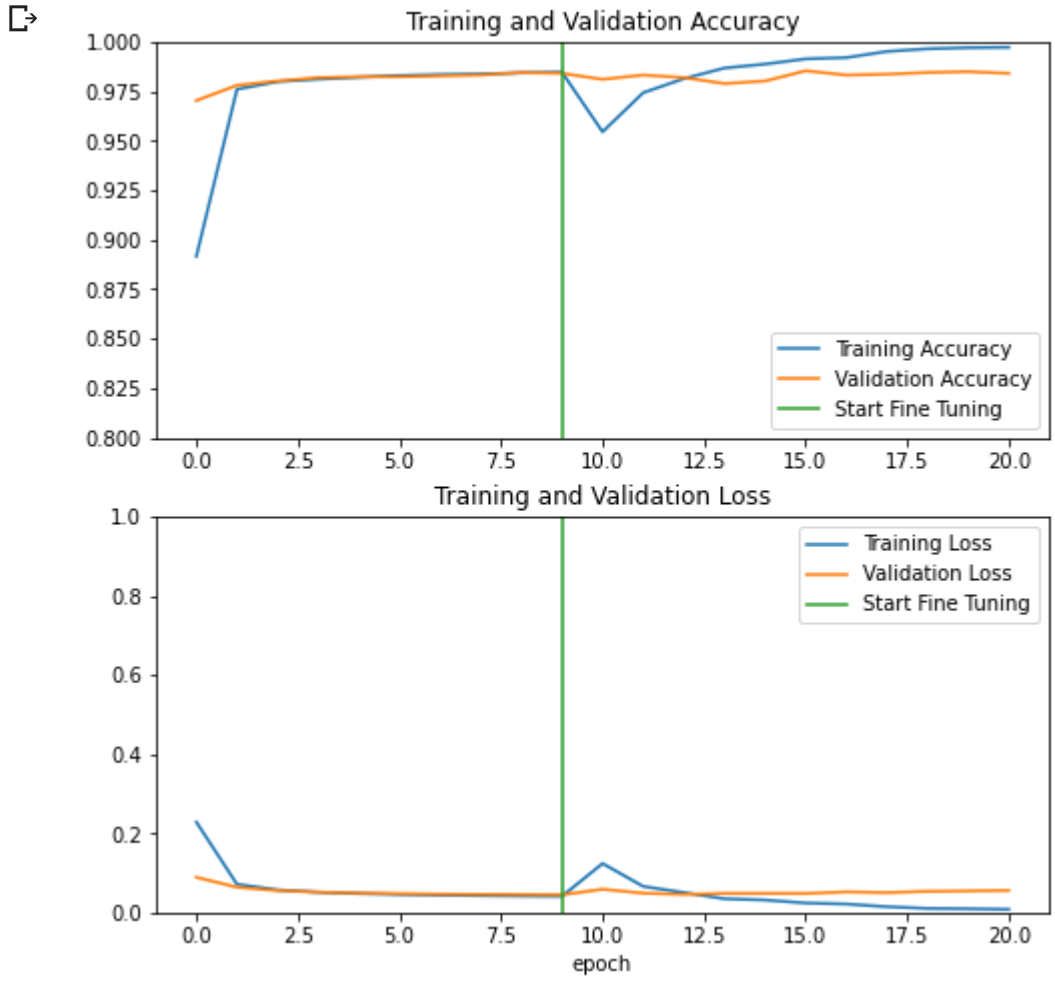
```
acc3 =acc+ history_fine.history['accuracy']
val_acc3 = val_acc+ history_fine.history['val_accuracy']

loss3=loss + history_fine.history['loss']
val_loss3=val_loss + history_fine.history['val_loss']


plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc3, label='Training Accuracy')
plt.plot(val_acc3, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss3, label='Training Loss')
plt.plot(val_loss3, label='Validation Loss')
```

```
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```




```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 120

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  155
```

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(lr=base_learning_rate/10),
```

```
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
mobilenetv2_1.00_160 (Model) (None, 5, 5, 1280)        2257984   
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1280)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 1281      
=================================================================
Total params: 2,259,265
Trainable params: 1,626,177
Non-trainable params: 633,088
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
38
```

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_batches,
                         epochs=total_epochs,
                         initial_epoch =  history.epoch[-1],
                         validation_data=validation_batches)
```

```
Epoch 10/20
```

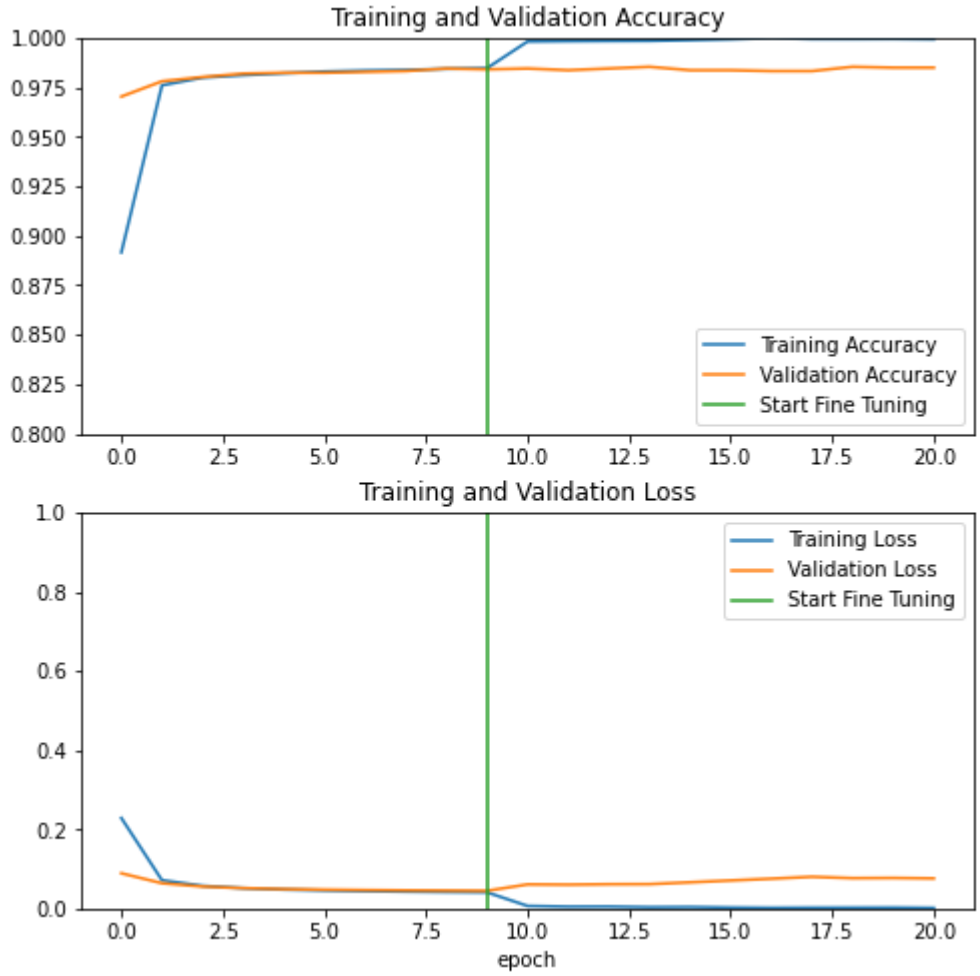
```
acc1 = acc+ history_fine.history['accuracy']
val_acc1 =val_acc+ history_fine.history['val_accuracy']

loss1 =loss + history_fine.history['loss']
val_loss1= val_loss + history_fine.history['val_loss']
```

```
582/582 [==============================] - 49s 83ms/step - loss: 0.0033 - accuracy: 0.99
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc1, label='Training Accuracy')
plt.plot(val_acc1, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss1, label='Training Loss')
plt.plot(val_loss1, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```




```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 140

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  155
```

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(lr=base_learning_rate/10),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_160 (Model) (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1280)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 1281
=================================================================
Total params: 2,259,265
Trainable params: 1,041,281
Non-trainable params: 1,217,984
_________________________________________________________________
```

```
len(model.trainable_variables)
```

```
17
```

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_batches,
                         epochs=total_epochs,
                         initial_epoch =  history.epoch[-1],
                         validation_data=validation_batches)
```

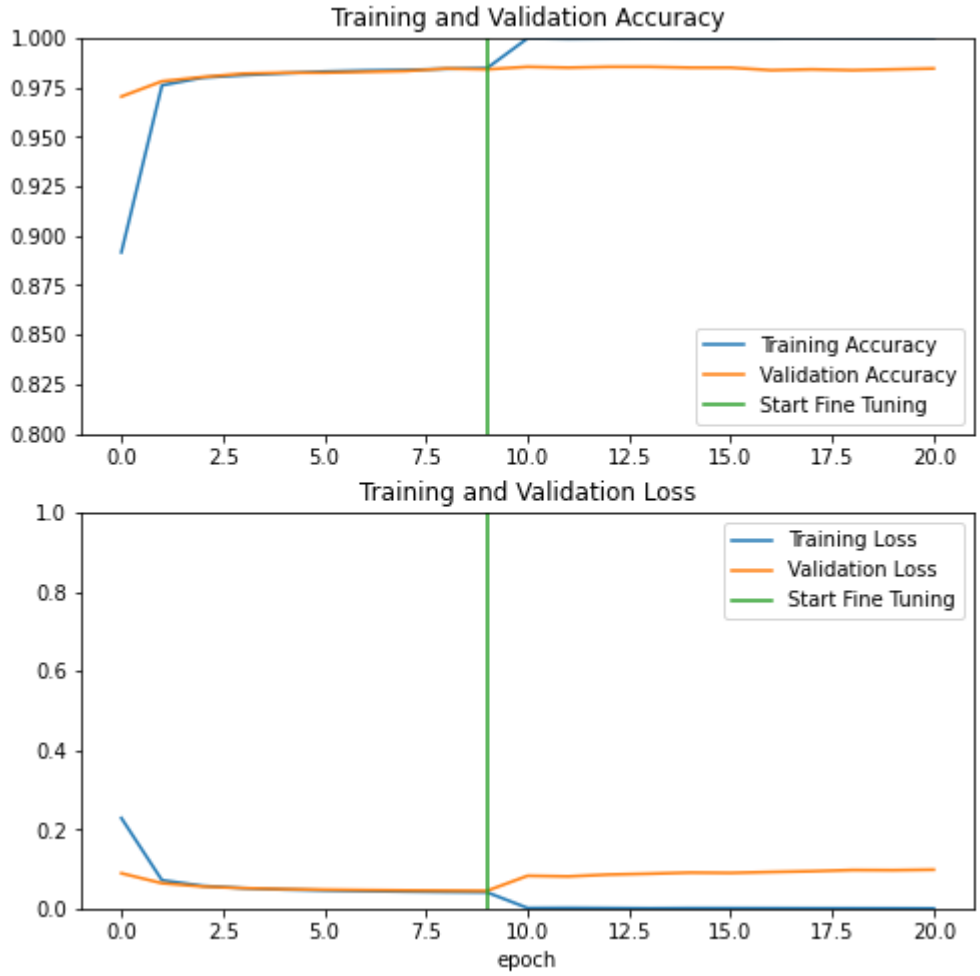
```
Epoch 10/20
582/582 [==============================] - 48s 83ms/step - loss: 0.0013 - accuracy: 0.99
Epoch 11/20
582/582 [==============================] - 47s 81ms/step - loss: 0.0016 - accuracy: 0.99
Epoch 12/20
582/582 [==============================] - 47s 81ms/step - loss: 0.0011 - accuracy: 0.99
Epoch 13/20
582/582 [==============================] - 47s 81ms/step - loss: 6.2954e-04 - accuracy:
Epoch 14/20
582/582 [==============================] - 47s 81ms/step - loss: 0.0010 - accuracy: 0.99
Epoch 15/20
582/582 [==============================] - 47s 81ms/step - loss: 9.9117e-04 - accuracy:
Epoch 16/20
582/582 [==============================] - 47s 81ms/step - loss: 8.9140e-04 - accuracy:
Epoch 17/20
582/582 [==============================] - 47s 82ms/step - loss: 7.3116e-04 - accuracy:
Epoch 18/20
582/582 [==============================] - 47s 81ms/step - loss: 6.8844e-04 - accuracy:
Epoch 19/20
582/582 [==============================] - 46s 80ms/step - loss: 6.0191e-04 - accuracy:
Epoch 20/20
582/582 [==============================] - 47s 81ms/step - loss: 4.0777e-04 - accuracy:
```

```
acc2 = acc+ history_fine.history['accuracy']
val_acc2 =val_acc+ history_fine.history['val_accuracy']
```

```
loss2 =loss + history_fine.history['loss']
val_loss2= val_loss + history_fine.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc2, label='Training Accuracy')
plt.plot(val_acc2, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss2, label='Training Loss')
plt.plot(val_loss2, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



discussion

```
'''
when finetune=100
accuracy: 0.9973 - val_accuracy: 0.9841
when finetune=120
accuracy: 0.9994 - val_accuracy: 0.9850
when finetune=140
accuracy: 0.9998  val_accuracy: 0.9845
while varying the depth by finetuning, it seems training accuracy is going higher compared to
can cause overfitting.
'''
```

Double-click (or enter) to edit

Summary:

- **Using a pre-trained model for feature extraction**: When working with a small dataset, it is a com features learned by a model trained on a larger dataset in the same domain. This is done by inst adding a fully-connected classifier on top. The pre-trained model is "frozen" and only the weights training. In this case, the convolutional base extracted all the features associated with each ima determines the image class given that set of extracted features.
- **Fine-tuning a pre-trained model**: To further improve performance, one might want to repurpose models to the new dataset via fine-tuning. In this case, you tuned your weights such that your m the dataset. This technique is usually recommended when the training dataset is large and very trained model was trained on.