**Transfer learning and fine-tuning**

|  |  |  |  |
| --- | --- | --- | --- |
| [[Image](https://www.tensorflow.org/tutorials/images/transfer_learning)View on TensorFlow.org](https://www.tensorflow.org/tutorials/images/transfer_learning) | [[Image](https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/transfer_learning.ipynb?force_kitty_mode=1&force_corgi_mode=1)Run in Google Colab](https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/transfer_learning.ipynb?force_kitty_mode=1&force_corgi_mode=1) | [[Image](https://github.com/tensorflow/docs/blob/master/site/en/tutorials/images/transfer_learning.ipynb)View source on GitHub](https://github.com/tensorflow/docs/blob/master/site/en/tutorials/images/transfer_learning.ipynb) | [[Image](https://storage.googleapis.com/tensorflow_docs/docs/site/en/tutorials/images/transfer_learning.ipynb)Download notebook](https://storage.googleapis.com/tensorflow_docs/docs/site/en/tutorials/images/transfer_learning.ipynb) |

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

A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. You 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 and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start 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 features from new samples. You simply add a new classifier, which will be trained from scratch, on top of the pretrained model so that you can repurpose the feature maps learned previously for the dataset.

You do not need to (re)train the entire model. The base convolutional network already contains features that are generically useful for classifying pictures. However, the final, classification part of the pretrained model is specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.

1. Fine-Tuning: Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them 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

[1]:



**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**import** os

**import** tensorflow **as** tf

**Data preprocessing**

**Data download**

In this tutorial, you will use a dataset containing several thousand images of cats and dogs. Download and extract a zip file containing the images, then create a tf.data.Dataset for training and validation using the tf.keras.utils.image\_dataset\_from\_directory utility. You can learn more about loading images in this [tutorial](https://www.tensorflow.org/tutorials/load_data/images).

[2]:



\_URL **=** 'https://storage.googleapis.com/mledu-datasets/cats\_and\_dogs\_filtered.zip'

path\_to\_zip **=** tf.keras.utils.get\_file('cats\_and\_dogs.zip', origin**=**\_URL, extract**=True**)

PATH **=** os.path.join(os.path.dirname(path\_to\_zip), 'cats\_and\_dogs\_filtered')

​

train\_dir **=** os.path.join(PATH, 'train')

validation\_dir **=** os.path.join(PATH, 'validation')

​

BATCH\_SIZE **=** 32

IMG\_SIZE **=** (160, 160)

​

train\_dataset **=** tf.keras.utils.image\_dataset\_from\_directory(train\_dir,

shuffle**=True**,

batch\_size**=**BATCH\_SIZE,

image\_size**=**IMG\_SIZE)

Found 2000 files belonging to 2 classes.

2023-03-14 13:16:46.759561: I tensorflow/core/common\_runtime/process\_util.cc:146] Creating new thread pool with default inter op setting: 2. Tune using inter\_op\_parallelism\_threads for best performance.

[3]:



validation\_dataset **=** tf.keras.utils.image\_dataset\_from\_directory(validation\_dir,

shuffle**=True**,

batch\_size**=**BATCH\_SIZE,

image\_size**=**IMG\_SIZE)

Found 1000 files belonging to 2 classes.

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

[4]:



class\_names **=** train\_dataset.class\_names

​

plt.figure(figsize**=**(10, 10))

**for** images, labels **in** train\_dataset.take(1):

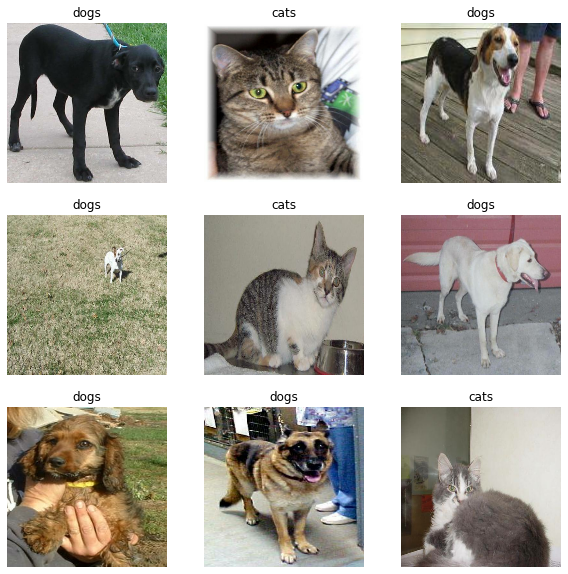
**for** i **in** range(9):

ax **=** plt.subplot(3, 3, i **+** 1)

plt.imshow(images[i].numpy().astype("uint8"))

plt.title(class\_names[labels[i]])

plt.axis("off")



As the original dataset doesn't contain a test set, you will create one. To do so, determine how many batches of data are available in the validation set using tf.data.experimental.cardinality, then move 20% of them to a test set.

[7]:



val\_batches **=** tf.data.experimental.cardinality(validation\_dataset)

test\_dataset **=** validation\_dataset.take(val\_batches **//** 5)

validation\_dataset **=** validation\_dataset.skip(val\_batches **//** 5)

[9]:



print('Number of validation batches: %d' **%** tf.data.experimental.cardinality(validation\_dataset))

print('Number of test batches: %d' **%** tf.data.experimental.cardinality(test\_dataset))

Number of validation batches: 21

Number of test batches: 5

**Configure the dataset for performance**

Use buffered prefetching to load images from disk without having I/O become blocking. To learn more about this method see the [data performance](https://www.tensorflow.org/guide/data_performance) guide.

[11]:



AUTOTUNE **=** tf.data.AUTOTUNE

​

train\_dataset **=** train\_dataset.prefetch(buffer\_size**=**AUTOTUNE)

validation\_dataset **=** validation\_dataset.prefetch(buffer\_size**=**AUTOTUNE)

test\_dataset **=** test\_dataset.prefetch(buffer\_size**=**AUTOTUNE)

**Use data augmentation**

When you don't have a large image dataset, it's a good practice to artificially introduce sample diversity by applying random, yet realistic, transformations to the training images, such as rotation and horizontal flipping. This helps expose the model to different aspects of the training data and reduce [overfitting](https://www.tensorflow.org/tutorials/keras/overfit_and_underfit). You can learn more about data augmentation in this [tutorial](https://www.tensorflow.org/tutorials/images/data_augmentation).

[13]:



data\_augmentation **=** tf.keras.Sequential([

tf.keras.layers.RandomFlip('horizontal'),

tf.keras.layers.RandomRotation(0.2),

])

Note: These layers are active only during training, when you call Model.fit. They are inactive when the model is used in inference mode in Model.evaluate or Model.fit.

Let's repeatedly apply these layers to the same image and see the result.

[14]:



**for** image, \_ **in** train\_dataset.take(1):

plt.figure(figsize**=**(10, 10))

first\_image **=** image[0]

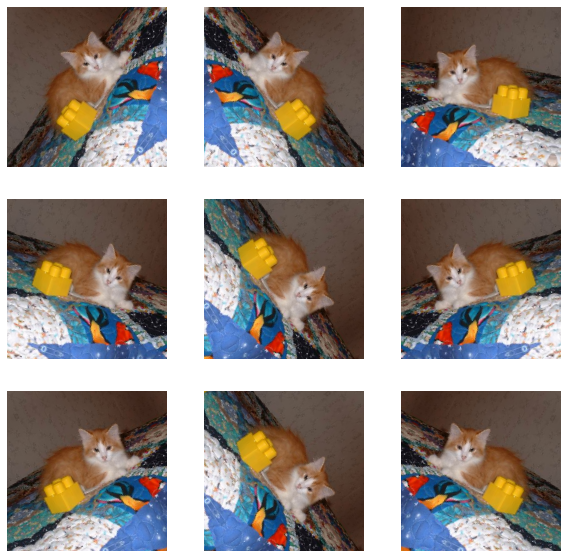
**for** i **in** range(9):

ax **=** plt.subplot(3, 3, i **+** 1)

augmented\_image **=** data\_augmentation(tf.expand\_dims(first\_image, 0))

plt.imshow(augmented\_image[0] **/** 255)

plt.axis('off')



**Rescale pixel values**

In a moment, you will download tf.keras.applications.MobileNetV2 for use as your base model. This model expects pixel values in [-1, 1], but at this point, the pixel values in your images are in [0, 255]. To rescale them, use the preprocessing method included with the model.

[15]:



preprocess\_input **=** tf.keras.applications.mobilenet\_v2.preprocess\_input

Note: Alternatively, you could rescale pixel values from [0, 255] to [-1, 1] using tf.keras.layers.Rescaling.

[16]:



rescale **=** tf.keras.layers.Rescaling(1.**/**127.5, offset**=-**1)

Note: If using other tf.keras.applications, be sure to check the API doc to determine if they expect pixels in [-1, 1] or [0, 1], or use the included preprocess\_input function.

**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 on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes. ImageNet is a research training dataset with a wide variety of categories like jackfruit and 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 classification layer (on "top", as most diagrams of machine learning models go from bottom to top) is not very useful. Instead, you will follow the common practice to depend on the very last layer before the flatten operation. This layer is called the "bottleneck layer". The bottleneck layer features retain more generality as compared to the final/top layer.

First, instantiate a MobileNet V2 model pre-loaded with weights trained on ImageNet. By specifying the **include\_top=False** argument, you load a network that doesn't include the classification layers at the top, which is ideal for feature extraction.

[17]:



*# Create the base model from the pre-trained model MobileNet V2*

IMG\_SHAPE **=** IMG\_SIZE **+** (3,)

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. Let's see what it does to an example batch of images:

[18]:



image\_batch, label\_batch **=** next(iter(train\_dataset))

feature\_batch **=** base\_model(image\_batch)

print(feature\_batch.shape)

User settings:

KMP\_AFFINITY=granularity=fine,verbose,compact,1,0

KMP\_BLOCKTIME=0

KMP\_SETTINGS=1

OMP\_NUM\_THREADS=4

Effective settings:

KMP\_ABORT\_DELAY=0

KMP\_ADAPTIVE\_LOCK\_PROPS='1,1024'

KMP\_ALIGN\_ALLOC=64

KMP\_ALL\_THREADPRIVATE=128

KMP\_ATOMIC\_MODE=2

KMP\_BLOCKTIME=0

KMP\_CPUINFO\_FILE: value is not defined

KMP\_DETERMINISTIC\_REDUCTION=false

KMP\_DEVICE\_THREAD\_LIMIT=2147483647

KMP\_DISP\_NUM\_BUFFERS=7

KMP\_DUPLICATE\_LIB\_OK=false

KMP\_ENABLE\_TASK\_THROTTLING=true

KMP\_FORCE\_REDUCTION: value is not defined

KMP\_FOREIGN\_THREADS\_THREADPRIVATE=true

KMP\_FORKJOIN\_BARRIER='2,2'

KMP\_FORKJOIN\_BARRIER\_PATTERN='hyper,hyper'

KMP\_GTID\_MODE=3

KMP\_HANDLE\_SIGNALS=false

KMP\_HOT\_TEAMS\_MAX\_LEVEL=1

KMP\_HOT\_TEAMS\_MODE=0

KMP\_INIT\_AT\_FORK=true

KMP\_LIBRARY=throughput

KMP\_LOCK\_KIND=queuing

KMP\_MALLOC\_POOL\_INCR=1M

KMP\_NUM\_LOCKS\_IN\_BLOCK=1

KMP\_PLAIN\_BARRIER='2,2'

KMP\_PLAIN\_BARRIER\_PATTERN='hyper,hyper'

KMP\_REDUCTION\_BARRIER='1,1'

KMP\_REDUCTION\_BARRIER\_PATTERN='hyper,hyper'

KMP\_SCHEDULE='static,balanced;guided,iterative'

KMP\_SETTINGS=true

KMP\_SPIN\_BACKOFF\_PARAMS='4096,100'

KMP\_STACKOFFSET=64

KMP\_STACKPAD=0

KMP\_STACKSIZE=8M

KMP\_STORAGE\_MAP=false

KMP\_TASKING=2

KMP\_TASKLOOP\_MIN\_TASKS=0

KMP\_TASK\_STEALING\_CONSTRAINT=1

KMP\_TEAMS\_THREAD\_LIMIT=4

KMP\_TOPOLOGY\_METHOD=all

KMP\_USE\_YIELD=1

KMP\_VERSION=false

KMP\_WARNINGS=true

OMP\_AFFINITY\_FORMAT='OMP: pid %P tid %i thread %n bound to OS proc set {%A}'

OMP\_ALLOCATOR=omp\_default\_mem\_alloc

OMP\_CANCELLATION=false

OMP\_DEFAULT\_DEVICE=0

OMP\_DISPLAY\_AFFINITY=false

OMP\_DISPLAY\_ENV=false

OMP\_DYNAMIC=false

OMP\_MAX\_ACTIVE\_LEVELS=1

OMP\_MAX\_TASK\_PRIORITY=0

OMP\_NESTED: deprecated; max-active-levels-var=1

OMP\_NUM\_THREADS='4'

OMP\_PLACES: value is not defined

OMP\_PROC\_BIND='intel'

OMP\_SCHEDULE='static'

OMP\_STACKSIZE=8M

OMP\_TARGET\_OFFLOAD=DEFAULT

OMP\_THREAD\_LIMIT=2147483647

OMP\_WAIT\_POLICY=PASSIVE

KMP\_AFFINITY='verbose,warnings,respect,granularity=fine,compact,1,0'

OMP: Info #211: KMP\_AFFINITY: decoding x2APIC ids.

OMP: Info #209: KMP\_AFFINITY: Affinity capable, using global cpuid leaf 11 info

OMP: Info #154: KMP\_AFFINITY: Initial OS proc set respected: 0-3

OMP: Info #156: KMP\_AFFINITY: 4 available OS procs

OMP: Info #157: KMP\_AFFINITY: Uniform topology

OMP: Info #179: KMP\_AFFINITY: 1 packages x 2 cores/pkg x 2 threads/core (2 total cores)

OMP: Info #213: KMP\_AFFINITY: OS proc to physical thread map:

OMP: Info #171: KMP\_AFFINITY: OS proc 0 maps to package 0 core 0 thread 0

OMP: Info #171: KMP\_AFFINITY: OS proc 2 maps to package 0 core 0 thread 1

OMP: Info #171: KMP\_AFFINITY: OS proc 1 maps to package 0 core 1 thread 0

OMP: Info #171: KMP\_AFFINITY: OS proc 3 maps to package 0 core 1 thread 1

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6048 thread 0 bound to OS proc set 0

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6316 thread 2 bound to OS proc set 2

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6315 thread 1 bound to OS proc set 1

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6317 thread 3 bound to OS proc set 3

(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 feature extractor. Additionally, you add a 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 setting layer.trainable = False) prevents the weights in a given layer from being updated during training. MobileNet V2 has many layers, so setting the entire model's trainable flag to False will freeze all of them.

[19]:



base\_model.trainable **=** **False**

**Important note about BatchNormalization layers**

Many models contain tf.keras.layers.BatchNormalization layers. This layer is a special case and precautions should be taken in the context of fine-tuning, as shown later in this tutorial.

When you set layer.trainable = False, the BatchNormalization layer will run in inference mode, and will not update its mean and variance statistics.

When you unfreeze a model that contains BatchNormalization layers in order to do fine-tuning, you should keep the BatchNormalization layers in inference mode by passing training = False when calling the base model. Otherwise, the updates applied to the non-trainable weights will destroy what the model has learned.

For more details, see the [Transfer learning guide](https://www.tensorflow.org/guide/keras/transfer_learning).

[20]:



*# 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\_1 (InputLayer) [(None, 160, 160, 3 0 []

)]

Conv1 (Conv2D) (None, 80, 80, 32) 864 ['input\_1[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 (Depth (None, 80, 80, 32) 288 ['Conv1\_relu[0][0]']

wiseConv2D)

expanded\_conv\_depthwise\_BN (Ba (None, 80, 80, 32) 128 ['expanded\_conv\_depthwise[0][0]']

tchNormalization)

expanded\_conv\_depthwise\_relu ( (None, 80, 80, 32) 0 ['expanded\_conv\_depthwise\_BN[0][0

ReLU) ]']

expanded\_conv\_project (Conv2D) (None, 80, 80, 16) 512 ['expanded\_conv\_depthwise\_relu[0]

[0]']

expanded\_conv\_project\_BN (Batc (None, 80, 80, 16) 64 ['expanded\_conv\_project[0][0]']

hNormalization)

block\_1\_expand (Conv2D) (None, 80, 80, 96) 1536 ['expanded\_conv\_project\_BN[0][0]'

]

block\_1\_expand\_BN (BatchNormal (None, 80, 80, 96) 384 ['block\_1\_expand[0][0]']

ization)

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][0]']

block\_1\_depthwise (DepthwiseCo (None, 40, 40, 96) 864 ['block\_1\_pad[0][0]']

nv2D)

block\_1\_depthwise\_BN (BatchNor (None, 40, 40, 96) 384 ['block\_1\_depthwise[0][0]']

malization)

block\_1\_depthwise\_relu (ReLU) (None, 40, 40, 96) 0 ['block\_1\_depthwise\_BN[0][0]']

block\_1\_project (Conv2D) (None, 40, 40, 24) 2304 ['block\_1\_depthwise\_relu[0][0]']

block\_1\_project\_BN (BatchNorma (None, 40, 40, 24) 96 ['block\_1\_project[0][0]']

lization)

block\_2\_expand (Conv2D) (None, 40, 40, 144) 3456 ['block\_1\_project\_BN[0][0]']

block\_2\_expand\_BN (BatchNormal (None, 40, 40, 144) 576 ['block\_2\_expand[0][0]']

ization)

block\_2\_expand\_relu (ReLU) (None, 40, 40, 144) 0 ['block\_2\_expand\_BN[0][0]']

block\_2\_depthwise (DepthwiseCo (None, 40, 40, 144) 1296 ['block\_2\_expand\_relu[0][0]']

nv2D)

block\_2\_depthwise\_BN (BatchNor (None, 40, 40, 144) 576 ['block\_2\_depthwise[0][0]']

malization)

block\_2\_depthwise\_relu (ReLU) (None, 40, 40, 144) 0 ['block\_2\_depthwise\_BN[0][0]']

block\_2\_project (Conv2D) (None, 40, 40, 24) 3456 ['block\_2\_depthwise\_relu[0][0]']

block\_2\_project\_BN (BatchNorma (None, 40, 40, 24) 96 ['block\_2\_project[0][0]']

lization)

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 (BatchNormal (None, 40, 40, 144) 576 ['block\_3\_expand[0][0]']

ization)

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][0]']

block\_3\_depthwise (DepthwiseCo (None, 20, 20, 144) 1296 ['block\_3\_pad[0][0]']

nv2D)

block\_3\_depthwise\_BN (BatchNor (None, 20, 20, 144) 576 ['block\_3\_depthwise[0][0]']

malization)

block\_3\_depthwise\_relu (ReLU) (None, 20, 20, 144) 0 ['block\_3\_depthwise\_BN[0][0]']

block\_3\_project (Conv2D) (None, 20, 20, 32) 4608 ['block\_3\_depthwise\_relu[0][0]']

block\_3\_project\_BN (BatchNorma (None, 20, 20, 32) 128 ['block\_3\_project[0][0]']

lization)

block\_4\_expand (Conv2D) (None, 20, 20, 192) 6144 ['block\_3\_project\_BN[0][0]']

block\_4\_expand\_BN (BatchNormal (None, 20, 20, 192) 768 ['block\_4\_expand[0][0]']

ization)

block\_4\_expand\_relu (ReLU) (None, 20, 20, 192) 0 ['block\_4\_expand\_BN[0][0]']

block\_4\_depthwise (DepthwiseCo (None, 20, 20, 192) 1728 ['block\_4\_expand\_relu[0][0]']

nv2D)

block\_4\_depthwise\_BN (BatchNor (None, 20, 20, 192) 768 ['block\_4\_depthwise[0][0]']

malization)

block\_4\_depthwise\_relu (ReLU) (None, 20, 20, 192) 0 ['block\_4\_depthwise\_BN[0][0]']

block\_4\_project (Conv2D) (None, 20, 20, 32) 6144 ['block\_4\_depthwise\_relu[0][0]']

block\_4\_project\_BN (BatchNorma (None, 20, 20, 32) 128 ['block\_4\_project[0][0]']

lization)

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 (BatchNormal (None, 20, 20, 192) 768 ['block\_5\_expand[0][0]']

ization)

block\_5\_expand\_relu (ReLU) (None, 20, 20, 192) 0 ['block\_5\_expand\_BN[0][0]']

block\_5\_depthwise (DepthwiseCo (None, 20, 20, 192) 1728 ['block\_5\_expand\_relu[0][0]']

nv2D)

block\_5\_depthwise\_BN (BatchNor (None, 20, 20, 192) 768 ['block\_5\_depthwise[0][0]']

malization)

block\_5\_depthwise\_relu (ReLU) (None, 20, 20, 192) 0 ['block\_5\_depthwise\_BN[0][0]']

block\_5\_project (Conv2D) (None, 20, 20, 32) 6144 ['block\_5\_depthwise\_relu[0][0]']

block\_5\_project\_BN (BatchNorma (None, 20, 20, 32) 128 ['block\_5\_project[0][0]']

lization)

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 (BatchNormal (None, 20, 20, 192) 768 ['block\_6\_expand[0][0]']

ization)

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][0]']

block\_6\_depthwise (DepthwiseCo (None, 10, 10, 192) 1728 ['block\_6\_pad[0][0]']

nv2D)

block\_6\_depthwise\_BN (BatchNor (None, 10, 10, 192) 768 ['block\_6\_depthwise[0][0]']

malization)

block\_6\_depthwise\_relu (ReLU) (None, 10, 10, 192) 0 ['block\_6\_depthwise\_BN[0][0]']

block\_6\_project (Conv2D) (None, 10, 10, 64) 12288 ['block\_6\_depthwise\_relu[0][0]']

block\_6\_project\_BN (BatchNorma (None, 10, 10, 64) 256 ['block\_6\_project[0][0]']

lization)

block\_7\_expand (Conv2D) (None, 10, 10, 384) 24576 ['block\_6\_project\_BN[0][0]']

block\_7\_expand\_BN (BatchNormal (None, 10, 10, 384) 1536 ['block\_7\_expand[0][0]']

ization)

block\_7\_expand\_relu (ReLU) (None, 10, 10, 384) 0 ['block\_7\_expand\_BN[0][0]']

block\_7\_depthwise (DepthwiseCo (None, 10, 10, 384) 3456 ['block\_7\_expand\_relu[0][0]']

nv2D)

block\_7\_depthwise\_BN (BatchNor (None, 10, 10, 384) 1536 ['block\_7\_depthwise[0][0]']

malization)

block\_7\_depthwise\_relu (ReLU) (None, 10, 10, 384) 0 ['block\_7\_depthwise\_BN[0][0]']

block\_7\_project (Conv2D) (None, 10, 10, 64) 24576 ['block\_7\_depthwise\_relu[0][0]']

block\_7\_project\_BN (BatchNorma (None, 10, 10, 64) 256 ['block\_7\_project[0][0]']

lization)

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 (BatchNormal (None, 10, 10, 384) 1536 ['block\_8\_expand[0][0]']

ization)

block\_8\_expand\_relu (ReLU) (None, 10, 10, 384) 0 ['block\_8\_expand\_BN[0][0]']

block\_8\_depthwise (DepthwiseCo (None, 10, 10, 384) 3456 ['block\_8\_expand\_relu[0][0]']

nv2D)

block\_8\_depthwise\_BN (BatchNor (None, 10, 10, 384) 1536 ['block\_8\_depthwise[0][0]']

malization)

block\_8\_depthwise\_relu (ReLU) (None, 10, 10, 384) 0 ['block\_8\_depthwise\_BN[0][0]']

block\_8\_project (Conv2D) (None, 10, 10, 64) 24576 ['block\_8\_depthwise\_relu[0][0]']

block\_8\_project\_BN (BatchNorma (None, 10, 10, 64) 256 ['block\_8\_project[0][0]']

lization)

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 (BatchNormal (None, 10, 10, 384) 1536 ['block\_9\_expand[0][0]']

ization)

block\_9\_expand\_relu (ReLU) (None, 10, 10, 384) 0 ['block\_9\_expand\_BN[0][0]']

block\_9\_depthwise (DepthwiseCo (None, 10, 10, 384) 3456 ['block\_9\_expand\_relu[0][0]']

nv2D)

block\_9\_depthwise\_BN (BatchNor (None, 10, 10, 384) 1536 ['block\_9\_depthwise[0][0]']

malization)

block\_9\_depthwise\_relu (ReLU) (None, 10, 10, 384) 0 ['block\_9\_depthwise\_BN[0][0]']

block\_9\_project (Conv2D) (None, 10, 10, 64) 24576 ['block\_9\_depthwise\_relu[0][0]']

block\_9\_project\_BN (BatchNorma (None, 10, 10, 64) 256 ['block\_9\_project[0][0]']

lization)

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 (BatchNorma (None, 10, 10, 384) 1536 ['block\_10\_expand[0][0]']

lization)

block\_10\_expand\_relu (ReLU) (None, 10, 10, 384) 0 ['block\_10\_expand\_BN[0][0]']

block\_10\_depthwise (DepthwiseC (None, 10, 10, 384) 3456 ['block\_10\_expand\_relu[0][0]']

onv2D)

block\_10\_depthwise\_BN (BatchNo (None, 10, 10, 384) 1536 ['block\_10\_depthwise[0][0]']

rmalization)

block\_10\_depthwise\_relu (ReLU) (None, 10, 10, 384) 0 ['block\_10\_depthwise\_BN[0][0]']

block\_10\_project (Conv2D) (None, 10, 10, 96) 36864 ['block\_10\_depthwise\_relu[0][0]']

block\_10\_project\_BN (BatchNorm (None, 10, 10, 96) 384 ['block\_10\_project[0][0]']

alization)

block\_11\_expand (Conv2D) (None, 10, 10, 576) 55296 ['block\_10\_project\_BN[0][0]']

block\_11\_expand\_BN (BatchNorma (None, 10, 10, 576) 2304 ['block\_11\_expand[0][0]']

lization)

block\_11\_expand\_relu (ReLU) (None, 10, 10, 576) 0 ['block\_11\_expand\_BN[0][0]']

block\_11\_depthwise (DepthwiseC (None, 10, 10, 576) 5184 ['block\_11\_expand\_relu[0][0]']

onv2D)

block\_11\_depthwise\_BN (BatchNo (None, 10, 10, 576) 2304 ['block\_11\_depthwise[0][0]']

rmalization)

block\_11\_depthwise\_relu (ReLU) (None, 10, 10, 576) 0 ['block\_11\_depthwise\_BN[0][0]']

block\_11\_project (Conv2D) (None, 10, 10, 96) 55296 ['block\_11\_depthwise\_relu[0][0]']

block\_11\_project\_BN (BatchNorm (None, 10, 10, 96) 384 ['block\_11\_project[0][0]']

alization)

block\_11\_add (Add) (None, 10, 10, 96) 0 ['block\_10\_project\_BN[0][0]',

'block\_11\_project\_BN[0][0]']

block\_12\_expand (Conv2D) (None, 10, 10, 576) 55296 ['block\_11\_add[0][0]']

block\_12\_expand\_BN (BatchNorma (None, 10, 10, 576) 2304 ['block\_12\_expand[0][0]']

lization)

block\_12\_expand\_relu (ReLU) (None, 10, 10, 576) 0 ['block\_12\_expand\_BN[0][0]']

block\_12\_depthwise (DepthwiseC (None, 10, 10, 576) 5184 ['block\_12\_expand\_relu[0][0]']

onv2D)

block\_12\_depthwise\_BN (BatchNo (None, 10, 10, 576) 2304 ['block\_12\_depthwise[0][0]']

rmalization)

block\_12\_depthwise\_relu (ReLU) (None, 10, 10, 576) 0 ['block\_12\_depthwise\_BN[0][0]']

block\_12\_project (Conv2D) (None, 10, 10, 96) 55296 ['block\_12\_depthwise\_relu[0][0]']

block\_12\_project\_BN (BatchNorm (None, 10, 10, 96) 384 ['block\_12\_project[0][0]']

alization)

block\_12\_add (Add) (None, 10, 10, 96) 0 ['block\_11\_add[0][0]',

'block\_12\_project\_BN[0][0]']

block\_13\_expand (Conv2D) (None, 10, 10, 576) 55296 ['block\_12\_add[0][0]']

block\_13\_expand\_BN (BatchNorma (None, 10, 10, 576) 2304 ['block\_13\_expand[0][0]']

lization)

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][0]']

block\_13\_depthwise (DepthwiseC (None, 5, 5, 576) 5184 ['block\_13\_pad[0][0]']

onv2D)

block\_13\_depthwise\_BN (BatchNo (None, 5, 5, 576) 2304 ['block\_13\_depthwise[0][0]']

rmalization)

block\_13\_depthwise\_relu (ReLU) (None, 5, 5, 576) 0 ['block\_13\_depthwise\_BN[0][0]']

block\_13\_project (Conv2D) (None, 5, 5, 160) 92160 ['block\_13\_depthwise\_relu[0][0]']

block\_13\_project\_BN (BatchNorm (None, 5, 5, 160) 640 ['block\_13\_project[0][0]']

alization)

block\_14\_expand (Conv2D) (None, 5, 5, 960) 153600 ['block\_13\_project\_BN[0][0]']

block\_14\_expand\_BN (BatchNorma (None, 5, 5, 960) 3840 ['block\_14\_expand[0][0]']

lization)

block\_14\_expand\_relu (ReLU) (None, 5, 5, 960) 0 ['block\_14\_expand\_BN[0][0]']

block\_14\_depthwise (DepthwiseC (None, 5, 5, 960) 8640 ['block\_14\_expand\_relu[0][0]']

onv2D)

block\_14\_depthwise\_BN (BatchNo (None, 5, 5, 960) 3840 ['block\_14\_depthwise[0][0]']

rmalization)

block\_14\_depthwise\_relu (ReLU) (None, 5, 5, 960) 0 ['block\_14\_depthwise\_BN[0][0]']

block\_14\_project (Conv2D) (None, 5, 5, 160) 153600 ['block\_14\_depthwise\_relu[0][0]']

block\_14\_project\_BN (BatchNorm (None, 5, 5, 160) 640 ['block\_14\_project[0][0]']

alization)

block\_14\_add (Add) (None, 5, 5, 160) 0 ['block\_13\_project\_BN[0][0]',

'block\_14\_project\_BN[0][0]']

block\_15\_expand (Conv2D) (None, 5, 5, 960) 153600 ['block\_14\_add[0][0]']

block\_15\_expand\_BN (BatchNorma (None, 5, 5, 960) 3840 ['block\_15\_expand[0][0]']

lization)

block\_15\_expand\_relu (ReLU) (None, 5, 5, 960) 0 ['block\_15\_expand\_BN[0][0]']

block\_15\_depthwise (DepthwiseC (None, 5, 5, 960) 8640 ['block\_15\_expand\_relu[0][0]']

onv2D)

block\_15\_depthwise\_BN (BatchNo (None, 5, 5, 960) 3840 ['block\_15\_depthwise[0][0]']

rmalization)

block\_15\_depthwise\_relu (ReLU) (None, 5, 5, 960) 0 ['block\_15\_depthwise\_BN[0][0]']

block\_15\_project (Conv2D) (None, 5, 5, 160) 153600 ['block\_15\_depthwise\_relu[0][0]']

block\_15\_project\_BN (BatchNorm (None, 5, 5, 160) 640 ['block\_15\_project[0][0]']

alization)

block\_15\_add (Add) (None, 5, 5, 160) 0 ['block\_14\_add[0][0]',

'block\_15\_project\_BN[0][0]']

block\_16\_expand (Conv2D) (None, 5, 5, 960) 153600 ['block\_15\_add[0][0]']

block\_16\_expand\_BN (BatchNorma (None, 5, 5, 960) 3840 ['block\_16\_expand[0][0]']

lization)

block\_16\_expand\_relu (ReLU) (None, 5, 5, 960) 0 ['block\_16\_expand\_BN[0][0]']

block\_16\_depthwise (DepthwiseC (None, 5, 5, 960) 8640 ['block\_16\_expand\_relu[0][0]']

onv2D)

block\_16\_depthwise\_BN (BatchNo (None, 5, 5, 960) 3840 ['block\_16\_depthwise[0][0]']

rmalization)

block\_16\_depthwise\_relu (ReLU) (None, 5, 5, 960) 0 ['block\_16\_depthwise\_BN[0][0]']

block\_16\_project (Conv2D) (None, 5, 5, 320) 307200 ['block\_16\_depthwise\_relu[0][0]']

block\_16\_project\_BN (BatchNorm (None, 5, 5, 320) 1280 ['block\_16\_project[0][0]']

alization)

Conv\_1 (Conv2D) (None, 5, 5, 1280) 409600 ['block\_16\_project\_BN[0][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, using a tf.keras.layers.GlobalAveragePooling2D layer to convert the features to a single 1280-element vector per image.

[22]:



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. You don't need an activation function here because this prediction will be treated as a logit, or a raw prediction value. Positive numbers predict class 1, negative numbers predict class 0.

[23]:



prediction\_layer **=** tf.keras.layers.Dense(1)

prediction\_batch **=** prediction\_layer(feature\_batch\_average)

print(prediction\_batch.shape)

(32, 1)

Build a model by chaining together the data augmentation, rescaling, base\_model and feature extractor layers using the [Keras Functional API](https://www.tensorflow.org/guide/keras/functional" \t "_blank). As previously mentioned, use training=False as our model contains a BatchNormalization layer.

[25]:



inputs **=** tf.keras.Input(shape**=**(160, 160, 3))

x **=** data\_augmentation(inputs)

x **=** preprocess\_input(x)

x **=** base\_model(x, training**=False**)

x **=** global\_average\_layer(x)

x **=** tf.keras.layers.Dropout(0.2)(x)

outputs **=** prediction\_layer(x)

model **=** tf.keras.Model(inputs, outputs)

**Compile the model**

Compile the model before training it. Since there are two classes, use the tf.keras.losses.BinaryCrossentropy loss with from\_logits=True since the model provides a linear output.

[27]:



base\_learning\_rate **=** 0.0001

model.compile(optimizer**=**tf.keras.optimizers.Adam(learning\_rate**=**base\_learning\_rate),

loss**=**tf.keras.losses.BinaryCrossentropy(from\_logits**=True**),

metrics**=**['accuracy'])

[28]:



model.summary()

Model: "model\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_3 (InputLayer) [(None, 160, 160, 3)] 0

sequential\_1 (Sequential) (None, 160, 160, 3) 0

tf.math.truediv\_1 (TFOpLamb (None, 160, 160, 3) 0

da)

tf.math.subtract\_1 (TFOpLam (None, 160, 160, 3) 0

bda)

mobilenetv2\_1.00\_160 (Funct (None, 5, 5, 1280) 2257984

ional)

global\_average\_pooling2d (G (None, 1280) 0

lobalAveragePooling2D)

dropout\_1 (Dropout) (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 2.5 million parameters in MobileNet are frozen, but there are 1.2 thousand *trainable* parameters in the Dense layer. These are divided between two tf.Variable objects, the weights and biases.

[29]:



len(model.trainable\_variables)

[29]:

2

**Train the model**

After training for 10 epochs, you should see ~94% accuracy on the validation set.

[30]:



initial\_epochs **=** 10

​

loss0, accuracy0 **=** model.evaluate(validation\_dataset)

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6083 thread 4 bound to OS proc set 0

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6426 thread 5 bound to OS proc set 1

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6428 thread 7 bound to OS proc set 3

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6427 thread 6 bound to OS proc set 2

2/21 [=>............................] - ETA: 3s - loss: 0.7727 - accuracy: 0.4688

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6084 thread 8 bound to OS proc set 0

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6429 thread 9 bound to OS proc set 1

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6430 thread 10 bound to OS proc set 2

OMP: Info #249: KMP\_AFFINITY: pid 6048 tid 6431 thread 11 bound to OS proc set 3

21/21 [==============================] - 6s 182ms/step - loss: 0.8192 - accuracy: 0.4753

[31]:



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

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

initial loss: 0.82

initial accuracy: 0.48

[32]:



history **=** model.fit(train\_dataset,

epochs**=**initial\_epochs,

validation\_data**=**validation\_dataset)

Epoch 1/10

63/63 [==============================] - 21s 288ms/step - loss: 0.6953 - accuracy: 0.6060 - val\_loss: 0.5522 - val\_accuracy: 0.6790

Epoch 2/10

63/63 [==============================] - 19s 292ms/step - loss: 0.5273 - accuracy: 0.7120 - val\_loss: 0.4073 - val\_accuracy: 0.7778

Epoch 3/10

63/63 [==============================] - 19s 292ms/step - loss: 0.4176 - accuracy: 0.7835 - val\_loss: 0.3278 - val\_accuracy: 0.8364

Epoch 4/10

63/63 [==============================] - 19s 291ms/step - loss: 0.3630 - accuracy: 0.8250 - val\_loss: 0.2671 - val\_accuracy: 0.8812

Epoch 5/10

63/63 [==============================] - 20s 307ms/step - loss: 0.3133 - accuracy: 0.8570 - val\_loss: 0.2253 - val\_accuracy: 0.9059

Epoch 6/10

63/63 [==============================] - 18s 288ms/step - loss: 0.2826 - accuracy: 0.8680 - val\_loss: 0.2001 - val\_accuracy: 0.9167

Epoch 7/10

63/63 [==============================] - 19s 292ms/step - loss: 0.2582 - accuracy: 0.8885 - val\_loss: 0.1759 - val\_accuracy: 0.9336

Epoch 8/10

63/63 [==============================] - 19s 296ms/step - loss: 0.2322 - accuracy: 0.8960 - val\_loss: 0.1631 - val\_accuracy: 0.9352

Epoch 9/10

63/63 [==============================] - 19s 299ms/step - loss: 0.2219 - accuracy: 0.9050 - val\_loss: 0.1497 - val\_accuracy: 0.9429

Epoch 10/10

63/63 [==============================] - 19s 289ms/step - loss: 0.2115 - accuracy: 0.9060 - val\_loss: 0.1310 - val\_accuracy: 0.9537

**Learning curves**

Let's take a look at the learning curves of the training and validation accuracy/loss when using the MobileNetV2 base model as a fixed feature extractor.

[33]:



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()

Graphical user interface

Description automatically generated

Note: If you are wondering why the validation metrics are clearly better than the training metrics, the main factor is because layers like tf.keras.layers.BatchNormalization and tf.keras.layers.Dropout affect accuracy during training. They are turned off when calculating validation loss.

To a lesser extent, it is also because training metrics report the average for an epoch, while validation metrics are evaluated after the epoch, so 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 MobileNetV2 base model. The weights of the pre-trained 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 of the pre-trained model alongside the training of the classifier you added. The training process will force the weights to be tuned from generic feature maps to features associated specifically with the dataset.

Note: This should only be attempted after you have trained the top-level classifier with the pre-trained model set to non-trainable. If you add a randomly initialized classifier on top of a pre-trained model and attempt to train all layers jointly, the magnitude of the gradient updates will be too large (due to the random weights from the classifier) and your pre-trained model will forget what it has learned.

Also, you should try to fine-tune a small number of top layers rather than the whole MobileNet model. In most convolutional networks, the higher up a layer is, the more specialized it is. The first few layers learn very simple and generic features that generalize to almost all types of images. As you go higher up, the features are increasingly more specific to the dataset on which the model was trained. The goal of fine-tuning is to adapt these specialized features to work with the new dataset, rather than overwrite the generic learning.

**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 should recompile the model (necessary for these changes to take effect), and resume training.

[34]:



base\_model.trainable **=** **True**

[35]:



*# 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: 154

**Compile the model**

As you are training a much larger model and want to readapt the pretrained weights, it is important to use a lower learning rate at this stage. Otherwise, your model could overfit very quickly.

[36]:



model.compile(loss**=**tf.keras.losses.BinaryCrossentropy(from\_logits**=True**),

optimizer **=** tf.keras.optimizers.RMSprop(learning\_rate**=**base\_learning\_rate**/**10),

metrics**=**['accuracy'])

[37]:



model.summary()

Model: "model\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_3 (InputLayer) [(None, 160, 160, 3)] 0

sequential\_1 (Sequential) (None, 160, 160, 3) 0

tf.math.truediv\_1 (TFOpLamb (None, 160, 160, 3) 0

da)

tf.math.subtract\_1 (TFOpLam (None, 160, 160, 3) 0

bda)

mobilenetv2\_1.00\_160 (Funct (None, 5, 5, 1280) 2257984

ional)

global\_average\_pooling2d (G (None, 1280) 0

lobalAveragePooling2D)

dropout\_1 (Dropout) (None, 1280) 0

dense\_1 (Dense) (None, 1) 1281

=================================================================

Total params: 2,259,265

Trainable params: 1,862,721

Non-trainable params: 396,544

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[38]:



len(model.trainable\_variables)

[38]:

56

**Continue training the model**

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

[39]:



fine\_tune\_epochs **=** 10

total\_epochs **=** initial\_epochs **+** fine\_tune\_epochs

​

history\_fine **=** model.fit(train\_dataset,

epochs**=**total\_epochs,

initial\_epoch**=**history.epoch[**-**1],

validation\_data**=**validation\_dataset)

Epoch 10/20

63/63 [==============================] - 33s 440ms/step - loss: 0.1537 - accuracy: 0.9335 - val\_loss: 0.0669 - val\_accuracy: 0.9707

Epoch 11/20

63/63 [==============================] - 28s 434ms/step - loss: 0.1297 - accuracy: 0.9450 - val\_loss: 0.0624 - val\_accuracy: 0.9738

Epoch 12/20

63/63 [==============================] - 28s 438ms/step - loss: 0.0916 - accuracy: 0.9665 - val\_loss: 0.0528 - val\_accuracy: 0.9799

Epoch 13/20

63/63 [==============================] - 27s 431ms/step - loss: 0.0927 - accuracy: 0.9595 - val\_loss: 0.0667 - val\_accuracy: 0.9722

Epoch 14/20

63/63 [==============================] - 27s 432ms/step - loss: 0.0778 - accuracy: 0.9670 - val\_loss: 0.0463 - val\_accuracy: 0.9815

Epoch 15/20

63/63 [==============================] - 28s 440ms/step - loss: 0.0995 - accuracy: 0.9580 - val\_loss: 0.0511 - val\_accuracy: 0.9815

Epoch 16/20

63/63 [==============================] - 28s 435ms/step - loss: 0.0736 - accuracy: 0.9720 - val\_loss: 0.0424 - val\_accuracy: 0.9769

Epoch 17/20

63/63 [==============================] - 28s 434ms/step - loss: 0.0792 - accuracy: 0.9690 - val\_loss: 0.0411 - val\_accuracy: 0.9799

Epoch 18/20

63/63 [==============================] - 28s 446ms/step - loss: 0.0693 - accuracy: 0.9735 - val\_loss: 0.0387 - val\_accuracy: 0.9799

Epoch 19/20

63/63 [==============================] - 28s 438ms/step - loss: 0.0687 - accuracy: 0.9765 - val\_loss: 0.0399 - val\_accuracy: 0.9861

Epoch 20/20

63/63 [==============================] - 28s 433ms/step - loss: 0.0541 - accuracy: 0.9810 - val\_loss: 0.0398 - val\_accuracy: 0.9861

Let's take a look at the learning curves of the training and validation accuracy/loss when fine-tuning the last few layers of the MobileNetV2 base model and training the classifier on top of it. The validation loss is much higher than the training loss, so you may get some overfitting.

You may also get some overfitting as the new training set is relatively small and similar to the original MobileNetV2 datasets.

After fine tuning the model nearly reaches 98% accuracy on the validation set.

[40]:



acc **+=** history\_fine.history['accuracy']

val\_acc **+=** history\_fine.history['val\_accuracy']

​

loss **+=** history\_fine.history['loss']

val\_loss **+=** history\_fine.history['val\_loss']

[41]:



plt.figure(figsize**=**(8, 8))

plt.subplot(2, 1, 1)

plt.plot(acc, label**=**'Training Accuracy')

plt.plot(val\_acc, 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(loss, label**=**'Training Loss')

plt.plot(val\_loss, 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()

A picture containing chart

Description automatically generated

**Evaluation and prediction**

Finally you can verify the performance of the model on new data using test set.

[42]:



loss, accuracy **=** model.evaluate(test\_dataset)

print('Test accuracy :', accuracy)

5/5 [==============================] - 2s 216ms/step - loss: 0.0253 - accuracy: 0.9937

Test accuracy : 0.9937499761581421

And now you are all set to use this model to predict if your pet is a cat or dog.

[43]:



*# Retrieve a batch of images from the test set*

image\_batch, label\_batch **=** test\_dataset.as\_numpy\_iterator().next()

predictions **=** model.predict\_on\_batch(image\_batch).flatten()

​

*# Apply a sigmoid since our model returns logits*

predictions **=** tf.nn.sigmoid(predictions)

predictions **=** tf.where(predictions **<** 0.5, 0, 1)

​

print('Predictions:\n', predictions.numpy())

print('Labels:\n', label\_batch)

​

plt.figure(figsize**=**(10, 10))

**for** i **in** range(9):

ax **=** plt.subplot(3, 3, i **+** 1)

plt.imshow(image\_batch[i].astype("uint8"))

plt.title(class\_names[predictions[i]])

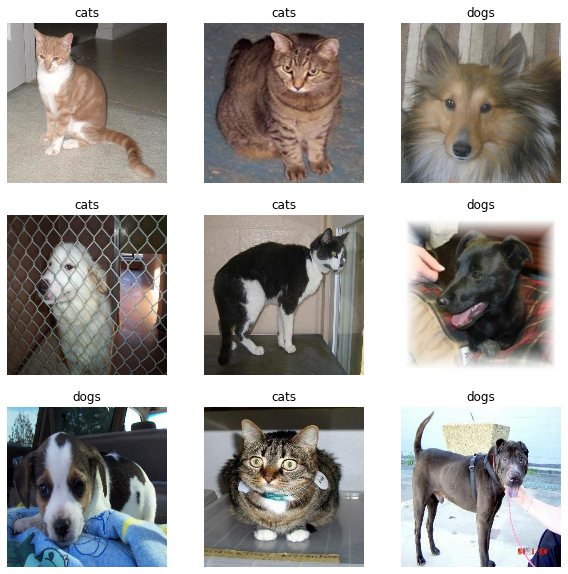
plt.axis("off")

Predictions:

[0 0 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0]

Labels:

[0 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0]



**Summary**

* **Using a pre-trained model for feature extraction**: When working with a small dataset, it is a common practice to take advantage of features learned by a model trained on a larger dataset in the same domain. This is done by instantiating the pre-trained model and adding a fully-connected classifier on top. The pre-trained model is "frozen" and only the weights of the classifier get updated during training. In this case, the convolutional base extracted all the features associated with each image and you just trained a classifier that 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 the top-level layers of the pre-trained models to the new dataset via fine-tuning. In this case, you tuned your weights such that your model learned high-level features specific to the dataset. This technique is usually recommended when the training dataset is large and very similar to the original dataset that the pre-trained model was trained on.

To learn more, visit the [Transfer learning guide](https://www.tensorflow.org/guide/keras/transfer_learning).