Transfer learning and fine-tuning

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.

2. 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

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
```

```
2024-08-16 03:08:02.565490: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register c
2024-08-16 03:08:02.586942: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register
2024-08-16 03:08:02.593445: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to registe
```

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`** (https://www.tensorflow.org/api_docs/python/tf/data/Dataset) for training and

validation using the **`tf.keras.utils.image_dataset_from_directory`** (https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image_dataset_from_directory) utility. You can learn more about loading images in this tutorial (https://www.tensorflow.org/tutorials/load_data/images).

```
_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)
```

```
Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
68606236/68606236 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Found 2000 files belonging to 2 classes.
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1723777686.391165  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
I0000 00:00:1723777686.394572  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
I0000 00:00:1723777686.398160  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
I0000 00:00:1723777686.401884  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
I0000 00:00:1723777686.413007  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
I0000 00:00:1723777686.416056  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
I0000 00:00:1723777686.419433  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
I0000 00:00:1723777686.422845  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
I0000 00:00:1723777686.426285  124422 cuda_executor.cc:1015] successful NUMA node read from SysFS had negative
```

```
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:

```
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` (https://www.tensorflow.org/api_docs/python/tf/data/experimental/cardinality), then move 20% of them to a test set.

```
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)
```

```
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: 26
Number of test batches: 6
```

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.

```
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).

```
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`** (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit). They are inactive when the model is used in inference mode in **`Model.evaluate`** (https://www.tensorflow.org/api_docs/python/tf/keras/Model#evaluate), **`Model.predict`** (https://www.tensorflow.org/api_docs/python/tf/keras/Model#predict), or **`Model.call`** (https://www.tensorflow.org/recommenders/api_docs/python/tfrs/models/Model#call).

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

```
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` (https://www.tensorflow.org/api_docs/python/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.

```
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`.

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

Note: If using other `tf.keras.applications` (https://www.tensorflow.org/api_docs/python/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.

```
# 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')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_we
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

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:

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
W0000 00:00:1723777693.629145  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.685023  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.686429  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.687892  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.689288  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.697765  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.699393  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.701006  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.702656  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.706264  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.709984  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.711869  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777693.713916  124422 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
```

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.

```
base_model.trainable = False
```

Important note about BatchNormalization layers

Many models contain `tf.keras.layers.BatchNormalization` (https://www.tensorflow.org/api_docs/python/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).

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

del: "mobilenetv2_1.00_160"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 160, 160, 3)	0	-
Conv1 (Conv2D)	(None, 80, 80, 32)	864	input_layer_1[0]…
bn_Conv1 (BatchNormalizatio…	(None, 80, 80, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 80, 80, 32)	0	bn_Conv1[0][0]
expanded_conv_dept… (DepthwiseConv2D)	(None, 80, 80, 32)	288	Conv1_relu[0][0]
expanded_conv_dept… (BatchNormalizatio…	(None, 80, 80, 32)	128	expanded_conv_de…
expanded_conv_dept… (ReLU)	(None, 80, 80, 32)	0	expanded_conv_de…
expanded_conv_proj… (Conv2D)	(None, 80, 80, 16)	512	expanded_conv_de…
expanded_conv_proj… (BatchNormalizatio…	(None, 80, 80, 16)	64	expanded_conv_pr…
block_1_expand (Conv2D)	(None, 80, 80, 96)	1,536	expanded_conv_pr…
block_1_expand_BN (BatchNormalizatio…	(None, 80, 80, 96)	384	block_1_expand[0…
block_1_expand_relu (ReLU)	(None, 80, 80, 96)	0	block_1_expand_B…
block_1_pad (ZeroPadding2D)	(None, 81, 81, 96)	0	block_1_expand_r…
block_1_depthwise (DepthwiseConv2D)	(None, 40, 40, 96)	864	block_1_pad[0][0]
block_1_depthwise_… (BatchNormalizatio…	(None, 40, 40, 96)	384	block_1_depthwis…
block_1_depthwise_… (ReLU)	(None, 40, 40, 96)	0	block_1_depthwis…
block_1_project (Conv2D)	(None, 40, 40, 24)	2,304	block_1_depthwis…
block_1_project_BN (BatchNormalizatio…	(None, 40, 40, 24)	96	block_1_project[…
block_2_expand (Conv2D)	(None, 40, 40, 144)	3,456	block_1_project_…
block_2_expand_BN (BatchNormalizatio…	(None, 40, 40, 144)	576	block_2_expand[0…
block_2_expand_relu (ReLU)	(None, 40, 40, 144)	0	block_2_expand_B…
block_2_depthwise (DepthwiseConv2D)	(None, 40, 40, 144)	1,296	block_2_expand_r…
block_2_depthwise_… (BatchNormalizatio…	(None, 40, 40, 144)	576	block_2_depthwis…
block_2_depthwise_… (ReLU)	(None, 40, 40, 144)	0	block_2_depthwis…
block_2_project (Conv2D)	(None, 40, 40, 24)	3,456	block_2_depthwis…
block_2_project_BN (BatchNormalizatio…	(None, 40, 40, 24)	96	block_2_project[…
block_2_add (Add)	(None, 40, 40, 24)	0	block_1_project_… block_2_project_…
block_3_expand (Conv2D)	(None, 40, 40, 144)	3,456	block_2_add[0][0]

block_3_expand_BN (BatchNormalizatio…	(None, 40, 40, 144)	576	block_3_expand[0…
block_3_expand_relu (ReLU)	(None, 40, 40, 144)	0	block_3_expand_B…
block_3_pad (ZeroPadding2D)	(None, 41, 41, 144)	0	block_3_expand_r…
block_3_depthwise (DepthwiseConv2D)	(None, 20, 20, 144)	1,296	block_3_pad[0][0]
block_3_depthwise_… (BatchNormalizatio…	(None, 20, 20, 144)	576	block_3_depthwis…
block_3_depthwise_… (ReLU)	(None, 20, 20, 144)	0	block_3_depthwis…
block_3_project (Conv2D)	(None, 20, 20, 32)	4,608	block_3_depthwis…
block_3_project_BN (BatchNormalizatio…	(None, 20, 20, 32)	128	block_3_project[…
block_4_expand (Conv2D)	(None, 20, 20, 192)	6,144	block_3_project_…
block_4_expand_BN (BatchNormalizatio…	(None, 20, 20, 192)	768	block_4_expand[0…
block_4_expand_relu (ReLU)	(None, 20, 20, 192)	0	block_4_expand_B…
block_4_depthwise (DepthwiseConv2D)	(None, 20, 20, 192)	1,728	block_4_expand_r…
block_4_depthwise_… (BatchNormalizatio…	(None, 20, 20, 192)	768	block_4_depthwis…
block_4_depthwise_… (ReLU)	(None, 20, 20, 192)	0	block_4_depthwis…
block_4_project (Conv2D)	(None, 20, 20, 32)	6,144	block_4_depthwis…
block_4_project_BN (BatchNormalizatio…	(None, 20, 20, 32)	128	block_4_project[…
block_4_add (Add)	(None, 20, 20, 32)	0	block_3_project_… block_4_project_…
block_5_expand (Conv2D)	(None, 20, 20, 192)	6,144	block_4_add[0][0]
block_5_expand_BN (BatchNormalizatio…	(None, 20, 20, 192)	768	block_5_expand[0…
block_5_expand_relu (ReLU)	(None, 20, 20, 192)	0	block_5_expand_B…
block_5_depthwise (DepthwiseConv2D)	(None, 20, 20, 192)	1,728	block_5_expand_r…
block_5_depthwise_… (BatchNormalizatio…	(None, 20, 20, 192)	768	block_5_depthwis…
block_5_depthwise_… (ReLU)	(None, 20, 20, 192)	0	block_5_depthwis…
block_5_project (Conv2D)	(None, 20, 20, 32)	6,144	block_5_depthwis…
block_5_project_BN (BatchNormalizatio…	(None, 20, 20, 32)	128	block_5_project[…
block_5_add (Add)	(None, 20, 20, 32)	0	block_4_add[0][0… block_5_project_…
block_6_expand (Conv2D)	(None, 20, 20, 192)	6,144	block_5_add[0][0]
block_6_expand_BN (BatchNormalizatio…	(None, 20, 20, 192)	768	block_6_expand[0…
block_6_expand_relu (ReLU)	(None, 20, 20, 192)	0	block_6_expand_B…
block_6_pad	(None, 21, 21,	0	block_6_expand_r…

(ZeroPadding2D)	192)		
block_6_depthwise (DepthwiseConv2D)	(None, 10, 10, 192)	1,728	block_6_pad[0][0]
block_6_depthwise_… (BatchNormalizatio…	(None, 10, 10, 192)	768	block_6_depthwis…
block_6_depthwise_… (ReLU)	(None, 10, 10, 192)	0	block_6_depthwis…
block_6_project (Conv2D)	(None, 10, 10, 64)	12,288	block_6_depthwis…
block_6_project_BN (BatchNormalizatio…	(None, 10, 10, 64)	256	block_6_project[…
block_7_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_6_project_…
block_7_expand_BN (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_7_expand[0…
block_7_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_7_expand_B…
block_7_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_7_expand_r…
block_7_depthwise_… (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_7_depthwis…
block_7_depthwise_… (ReLU)	(None, 10, 10, 384)	0	block_7_depthwis…
block_7_project (Conv2D)	(None, 10, 10, 64)	24,576	block_7_depthwis…
block_7_project_BN (BatchNormalizatio…	(None, 10, 10, 64)	256	block_7_project[…
block_7_add (Add)	(None, 10, 10, 64)	0	block_6_project_… block_7_project_…
block_8_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_7_add[0][0]
block_8_expand_BN (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_8_expand[0…
block_8_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_8_expand_B…
block_8_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_8_expand_r…
block_8_depthwise_… (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_8_depthwis…
block_8_depthwise_… (ReLU)	(None, 10, 10, 384)	0	block_8_depthwis…
block_8_project (Conv2D)	(None, 10, 10, 64)	24,576	block_8_depthwis…
block_8_project_BN (BatchNormalizatio…	(None, 10, 10, 64)	256	block_8_project[…
block_8_add (Add)	(None, 10, 10, 64)	0	block_7_add[0][0… block_8_project_…
block_9_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_8_add[0][0]
block_9_expand_BN (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_9_expand[0…
block_9_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_9_expand_B…
block_9_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_9_expand_r…
block_9_depthwise_… (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_9_depthwis…
block_9_depthwise_… (ReLU)	(None, 10, 10, 384)	0	block_9_depthwis…
block_9_project	(None, 10, 10	24,576	block_9_depthwis…

block_9_project (Conv2D)	(None, 10, 10, 64)	24,576	block_9_depthwis…
block_9_project_BN (BatchNormalizatio…	(None, 10, 10, 64)	256	block_9_project[…
block_9_add (Add)	(None, 10, 10, 64)	0	block_8_add[0][0]… block_9_project_…
block_10_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_9_add[0][0]
block_10_expand_BN (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_10_expand[…
block_10_expand_re… (ReLU)	(None, 10, 10, 384)	0	block_10_expand_…
block_10_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_10_expand_…
block_10_depthwise… (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_10_depthwi…
block_10_depthwise… (ReLU)	(None, 10, 10, 384)	0	block_10_depthwi…
block_10_project (Conv2D)	(None, 10, 10, 96)	36,864	block_10_depthwi…
block_10_project_BN (BatchNormalizatio…	(None, 10, 10, 96)	384	block_10_project…
block_11_expand (Conv2D)	(None, 10, 10, 576)	55,296	block_10_project…
block_11_expand_BN (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_11_expand[…
block_11_expand_re… (ReLU)	(None, 10, 10, 576)	0	block_11_expand_…
block_11_depthwise (DepthwiseConv2D)	(None, 10, 10, 576)	5,184	block_11_expand_…
block_11_depthwise… (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_11_depthwi…
block_11_depthwise… (ReLU)	(None, 10, 10, 576)	0	block_11_depthwi…
block_11_project (Conv2D)	(None, 10, 10, 96)	55,296	block_11_depthwi…
block_11_project_BN (BatchNormalizatio…	(None, 10, 10, 96)	384	block_11_project…
block_11_add (Add)	(None, 10, 10, 96)	0	block_10_project… block_11_project…
block_12_expand (Conv2D)	(None, 10, 10, 576)	55,296	block_11_add[0][…
block_12_expand_BN (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_12_expand[…
block_12_expand_re… (ReLU)	(None, 10, 10, 576)	0	block_12_expand_…
block_12_depthwise (DepthwiseConv2D)	(None, 10, 10, 576)	5,184	block_12_expand_…
block_12_depthwise… (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_12_depthwi…
block_12_depthwise… (ReLU)	(None, 10, 10, 576)	0	block_12_depthwi…
block_12_project (Conv2D)	(None, 10, 10, 96)	55,296	block_12_depthwi…
block_12_project_BN (BatchNormalizatio…	(None, 10, 10, 96)	384	block_12_project…
block_12_add (Add)	(None, 10, 10, 96)	0	block_11_add[0][… block_12_project…
block_13_expand (Conv2D)	(None, 10, 10, 576)	55,296	block_12_add[0][…

block_13_expand_BN (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_13_expand[…
block_13_expand_re… (ReLU)	(None, 10, 10, 576)	0	block_13_expand_…
block_13_pad (ZeroPadding2D)	(None, 11, 11, 576)	0	block_13_expand_…
block_13_depthwise (DepthwiseConv2D)	(None, 5, 5, 576)	5,184	block_13_pad[0][…
block_13_depthwise… (BatchNormalizatio…	(None, 5, 5, 576)	2,304	block_13_depthwi…
block_13_depthwise… (ReLU)	(None, 5, 5, 576)	0	block_13_depthwi…
block_13_project (Conv2D)	(None, 5, 5, 160)	92,160	block_13_depthwi…
block_13_project_BN (BatchNormalizatio…	(None, 5, 5, 160)	640	block_13_project…
block_14_expand (Conv2D)	(None, 5, 5, 960)	153,600	block_13_project…
block_14_expand_BN (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_14_expand[…
block_14_expand_re… (ReLU)	(None, 5, 5, 960)	0	block_14_expand_…
block_14_depthwise (DepthwiseConv2D)	(None, 5, 5, 960)	8,640	block_14_expand_…
block_14_depthwise… (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_14_depthwi…
block_14_depthwise… (ReLU)	(None, 5, 5, 960)	0	block_14_depthwi…
block_14_project (Conv2D)	(None, 5, 5, 160)	153,600	block_14_depthwi…
block_14_project_BN (BatchNormalizatio…	(None, 5, 5, 160)	640	block_14_project…
block_14_add (Add)	(None, 5, 5, 160)	0	block_13_project… block_14_project…
block_15_expand (Conv2D)	(None, 5, 5, 960)	153,600	block_14_add[0][…
block_15_expand_BN (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_15_expand[…
block_15_expand_re… (ReLU)	(None, 5, 5, 960)	0	block_15_expand_…
block_15_depthwise (DepthwiseConv2D)	(None, 5, 5, 960)	8,640	block_15_expand_…
block_15_depthwise… (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_15_depthwi…
block_15_depthwise… (ReLU)	(None, 5, 5, 960)	0	block_15_depthwi…
block_15_project (Conv2D)	(None, 5, 5, 160)	153,600	block_15_depthwi…
block_15_project_BN (BatchNormalizatio…	(None, 5, 5, 160)	640	block_15_project…
block_15_add (Add)	(None, 5, 5, 160)	0	block_14_add[0][… block_15_project…
block_16_expand (Conv2D)	(None, 5, 5, 960)	153,600	block_15_add[0][…
block_16_expand_BN (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_16_expand[…
block_16_expand_re… (ReLU)	(None, 5, 5, 960)	0	block_16_expand_…

```
rainable params: 0 (0.00 B)
```

```
on-trainable params: 2,257,984 (8.61 MB)
```

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` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling2D) layer to convert the features to a single 1280-element vector per image.

```
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` (https://www.tensorflow.org/api_docs/python/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.

```
prediction_layer = tf.keras.layers.Dense(1, activation='sigmoid')
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). As previously mentioned, use `training=False` as our model contains a `BatchNormalization` layer.

```
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)
```

```
model.summary()
```

```
del: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 160, 160, 3)	0
sequential (Sequential)	(None, 160, 160, 3)	0
true_divide (TrueDivide)	(None, 160, 160, 3)	0
subtract (Subtract)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functional)	(None, 5, 5, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1,281

otal params: 2,259,265 (8.62 MB)

rainable params: 1,281 (5.00 KB)

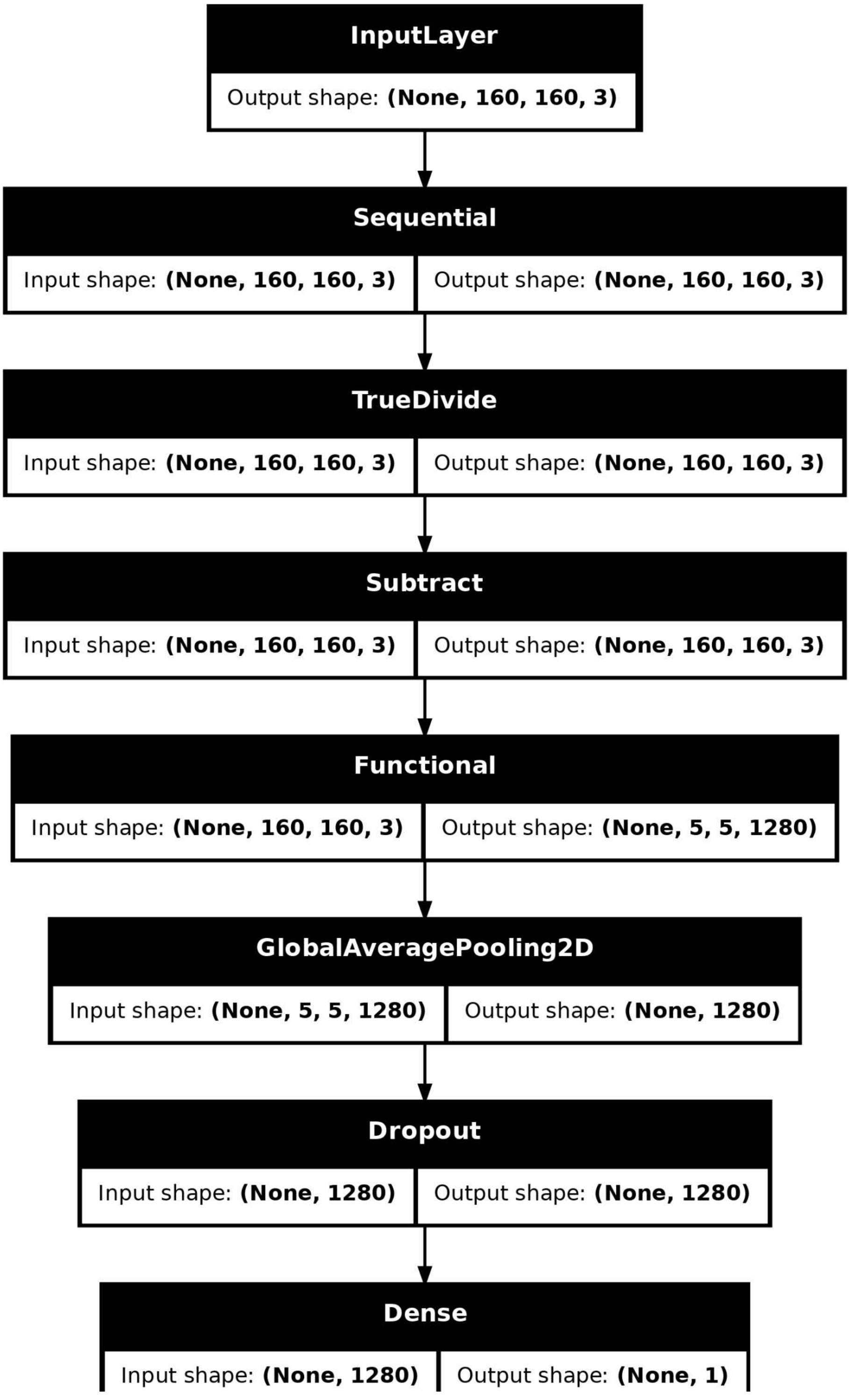
on-trainable params: 2,257,984 (8.61 MB)

The 8+ 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`** (https://www.tensorflow.org/api_docs/python/tf/Variable) objects, the weights and biases.

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

2

```
tf.keras.utils.plot_model(model, show_shapes=True)
```


InputLayer
Output shape: (None, 160, 160, 3)
Sequential
Input shape: (None, 160, 160, 3)
Output shape: (None, 160, 160, 3)
TrueDivide
Input shape: (None, 160, 160, 3)
Output shape: (None, 160, 160, 3)
Subtract
Input shape: (None, 160, 160, 3)
Output shape: (None, 160, 160, 3)
Functional
Input shape: (None, 160, 160, 3)
Output shape: (None, 5, 5, 1280)
GlobalAveragePooling2D
Input shape: (None, 5, 5, 1280)
Output shape: (None, 1280)
Dropout
Input shape: (None, 1280)
Output shape: (None, 1280)
Dense
Input shape: (None, 1280)
Output shape: (None, 1)

Compile the model

Compile the model before training it. Since there are two classes and a sigmoid oputput, use the `BinaryAccuracy`.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(threshold=0.5, name='accuracy')])
```

Train the model

After training for 10 epochs, you should see ~96% accuracy on the validation set.

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
W0000 00:00:1723777695.929524  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.931125  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.932523  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.933950  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.935401  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.937180  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.938849  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.940353  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.941907  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.943408  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.944914  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.946697  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777695.964661  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
```

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

```
initial loss: 0.70
initial accuracy: 0.56
```

```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
61/63 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.5997 - loss: 0.6990
W0000 00:00:1723777704.397737  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.398671  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.399497  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.400322  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.401154  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.402015  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.402938  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.403723  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.404513  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.405339  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777704.406274  124620 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
```

Learning curves

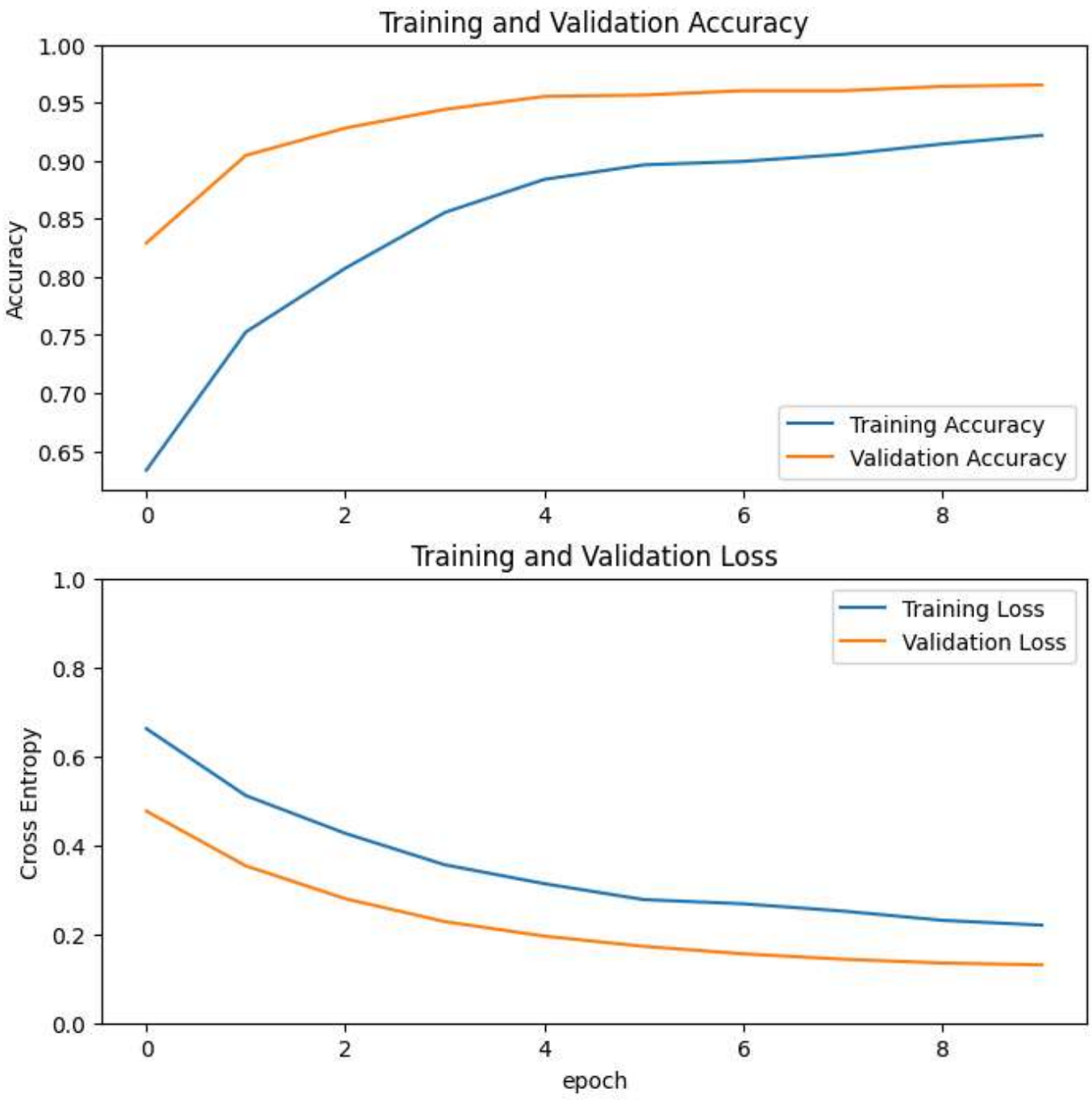
Let's take a look at the learning curves of the training and validation accuracy/loss when using the MobileNetV2 base model as a fixed feature extractor.

```
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 main factor is because layers like **`tf.keras.layers.BatchNormalization`** (https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization) and **`tf.keras.layers.Dropout`** (https://www.tensorflow.org/api_docs/python/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.

```
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:  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.

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=[tf.keras.metrics.BinaryAccuracy(threshold=0.5, name='accuracy')])
```

```
model.summary()
```

del: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 160, 160, 3)	0
sequential (Sequential)	(None, 160, 160, 3)	0
true_divide (TrueDivide)	(None, 160, 160, 3)	0
subtract (Subtract)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functional)	(None, 5, 5, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1,281

otal params: 2,259,265 (8.62 MB)

rainable params: 1,862,721 (7.11 MB)

on-trainable params: 396,544 (1.51 MB)

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

```
56
```

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_dataset,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset)
```

```
Epoch 11/20
W0000 00:00:1723777737.423707  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.425731  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
```

```
W0000 00:00:1723777737.427640  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.429540  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.433756  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.435535  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.438729  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.440841  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.442751  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.444746  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.446945  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
W0000 00:00:1723777737.449153  124625 gpu_timer.cc:114] Skipping the delay kernel, measurement accuracy will b
```

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.

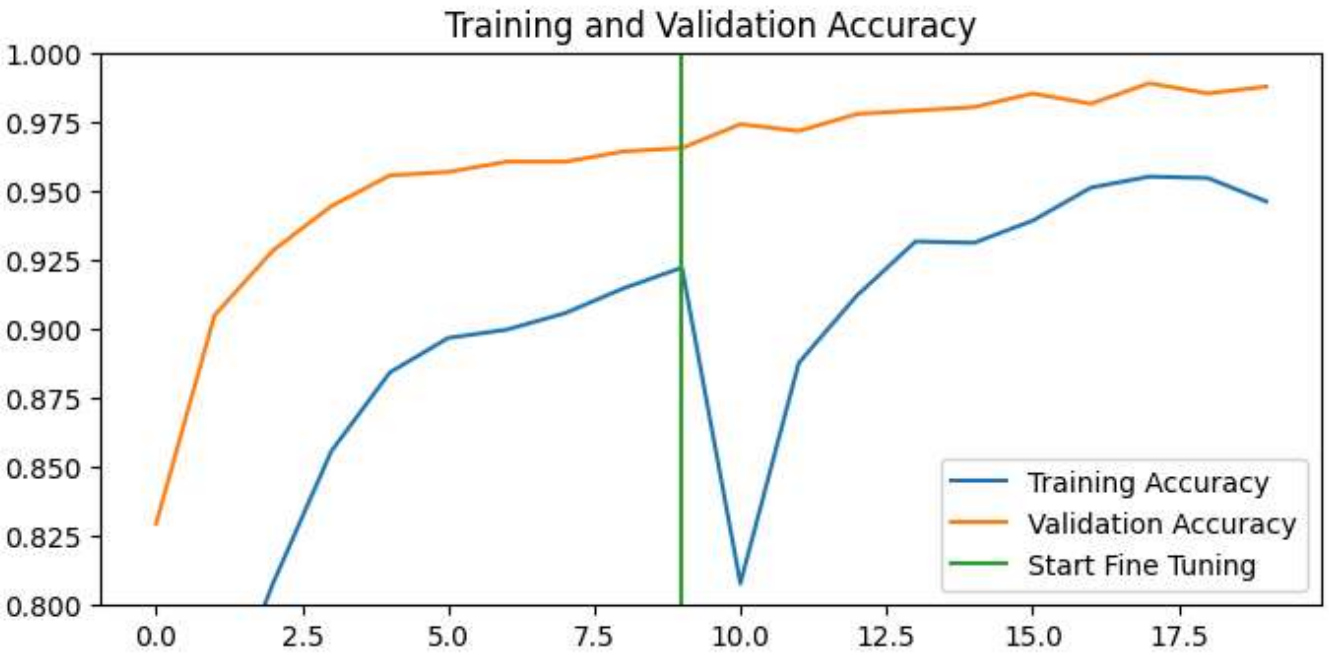
After fine tuning the model nearly reaches 98% accuracy on the validation set.

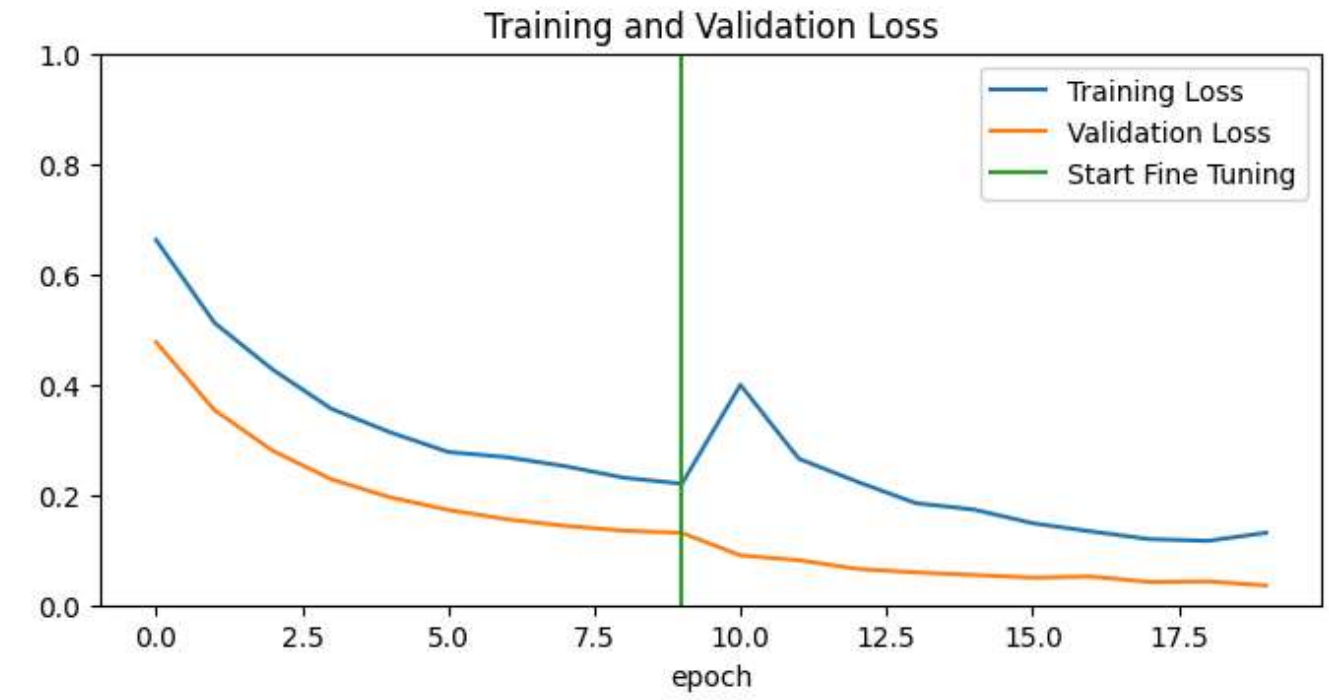
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.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.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()
```





Evaluation and prediction

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

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.9600 - loss: 0.0837
Test accuracy : 0.96875
```

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

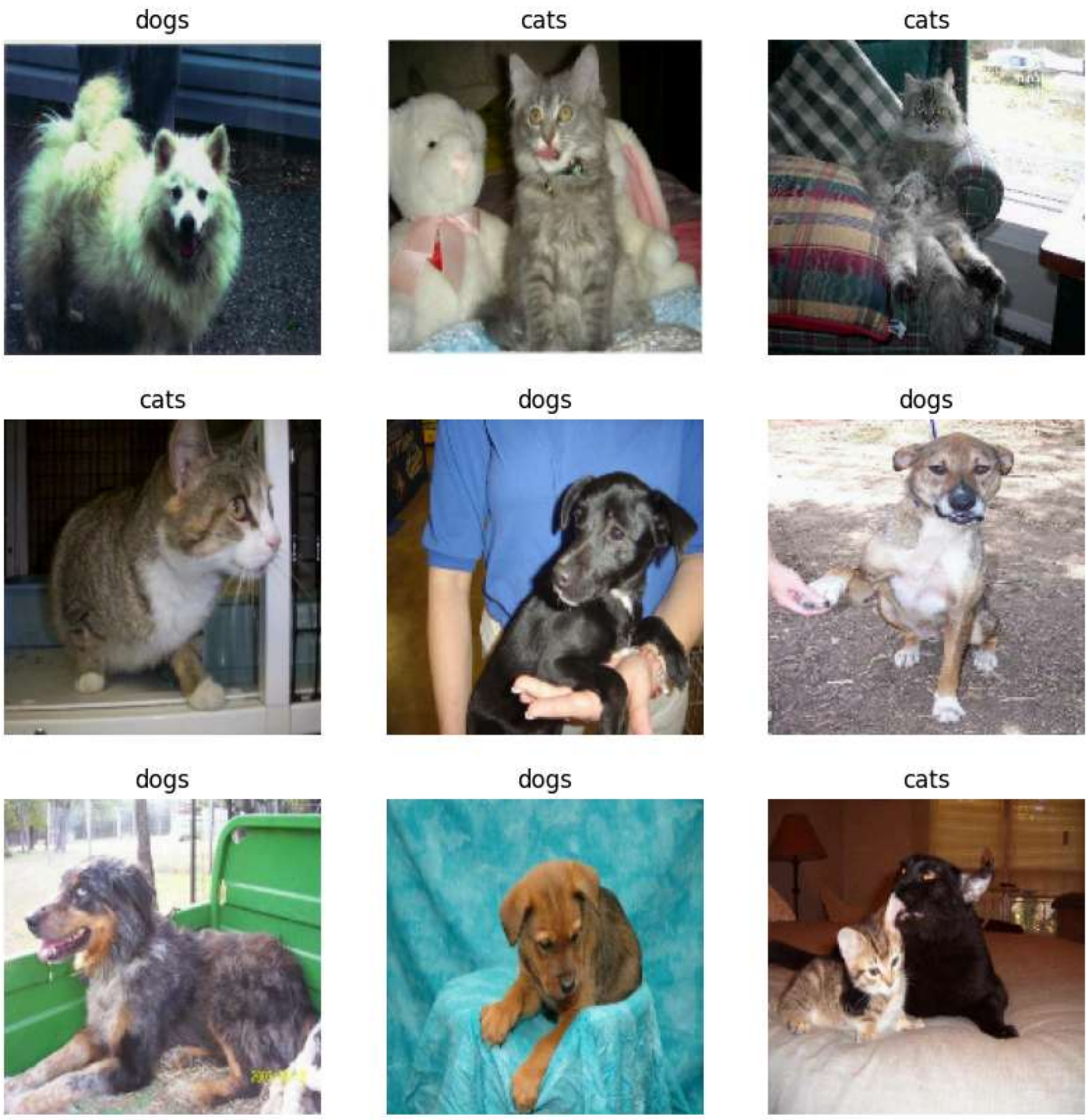
```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()
predictions = tf.where(predictions < 0.5, 0, 1)

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:
 [1 0 0 0 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0]
```

```
Labels:
 [1 0 0 0 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 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).

```
# MIT License
#
# Copyright (c) 2017 François Chollet
#
# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
```

```
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.
```


Last updated 2024-08-16 UTC.