Introduction

This example shows how to do image classification from scratch, starting from JPEG image files on disk, without leveraging pre-trained weights or a pre-made Keras Application model. We demonstrate the workflow on the Kaggle Cats vs Dogs binary classification dataset.

We use the `image_dataset_from_directory` utility to generate the datasets, and we use Keras image preprocessing layers for image standardization and data augmentation.

Setup

In [1]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from warnings import filterwarnings
filterwarnings('ignore')
```

Load the data: the Cats vs Dogs dataset

In [5]:

```
!curl -O https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanc
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  786M  100  786M    0     0  85.6M      0  0:00:09  0:00:09 --:--:-- 81.3M
```

In [6]:

```
!unzip -q kagglecatsanddogs_3367a.zip
!ls
```

```
 kagglecatsanddogs_3367a.zip   PetImages         sample_data
'MSR-LA - 3467.docx'       'readme[1].txt'   x
```

In [7]:

```
!ls PetImages
```

```
Cat  Dog
```

Filter out corrupted images

When working with lots of real-world image data, corrupted images are a common occurence. Let's filter out badly-encoded images that do not feature the string "JFIF" in their header.

In [8]:

```
import os

num_skipped = 0
for folder_name in ("Cat", "Dog"):
    folder_path = os.path.join("PetImages", folder_name)
    for fname in os.listdir(folder_path):
        fpath = os.path.join(folder_path, fname)
        try:
            fobj = open(fpath, "rb")
            is_jfif = tf.compat.as_bytes("JFIF") in fobj.peek(10)
        finally:
            fobj.close()

        if not is_jfif:
            num_skipped += 1
            # Delete corrupted image
            os.remove(fpath)

print("Deleted %d images" % num_skipped)
```

```
Deleted 1590 images
```

Generate a `Dataset`

In [9]:

```
image_size = (180, 180)
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "PetImages",
    validation_split=0.2,
```

```
    subset="training",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "PetImages",
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
```

```
Found 23410 files belonging to 2 classes.
Using 18728 files for training.
Found 23410 files belonging to 2 classes.
Using 4682 files for validation.
```

Visualize the data

Here are the first 9 images in the training dataset. As you can see, label 1 is "dog" and label 0 is "cat".

In [10]:

```
import matplotlib.pyplot as plt


def vis(ds):
  plt.figure(figsize=(10, 10))
  for images, labels in ds.take(100):
    for i in range(9):
      ax = plt.subplot(3, 3, i + 1)
      plt.imshow(images[i].numpy().astype("uint8"))
      plt.title(int(labels[i]))
      plt.axis("off")

vis(train_ds)
```

Using image 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 random horizontal flipping or small random rotations. This helps expose the model to different aspects of the training data while slowing down overfitting.

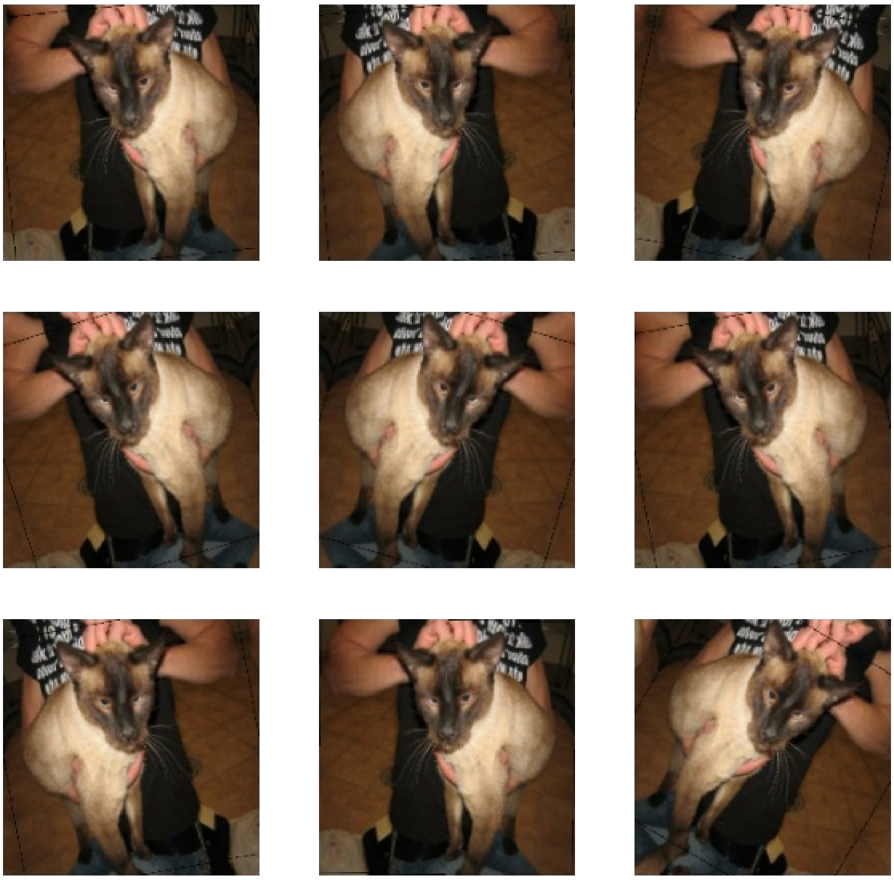
In [11]:

```
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomRotation(0.1),
    ]
)
```

Let's visualize what the augmented samples look like, by applying `data_augmentation` repeatedly to the first image in the dataset:

In [12]:

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```



Standardizing the data

Our image are already in a standard size (180x180), as they are being yielded as contiguous `float32` batches by our dataset. However, their RGB channel values are in the `[0, 255]` range. This is not ideal for a neural network; in general you should seek to make your input values small. Here, we will standardize values to be in the `[0, 1]` by using a `Rescaling` layer at the start of our model.

Two options to preprocess the data

Option 1: Make it part of the model , like this:

```
inputs = keras.Input(shape=input_shape)
x = data_augmentation(inputs)
x = layers.experimental.preprocessing.Rescaling(1./255)(x)
...  # Rest of the model
```

With this option, your data augmentation will happen *on device*, synchronously with the rest of the model execution, meaning that it will benefit from GPU acceleration.

Note that data augmentation is inactive at test time, so the input samples will only be augmented during `fit()` , not when calling `evaluate()` or `predict()` .

If you're training on GPU, this is the better option.

Option 2: apply it to the dataset , so as to obtain a dataset that yields batches of augmented images, like this:

```
augmented_train_ds = train_ds.map(
  lambda x, y: (data_augmentation(x, training=True), y))
```

With this option, your data augmentation will happen **on CPU**, asynchronously, and will be buffered before going into the model.

If you're training on CPU, this is the better option, since it makes data augmentation asynchronous and non-blocking.

In our case, we'll go with the first option.

Configure the dataset for performance

Let's make sure to use buffered prefetching so we can yield data from disk without having I/O becoming blocking:

In [13]:

```
train_ds = train_ds.prefetch(buffer_size=32)
val_ds = val_ds.prefetch(buffer_size=32)
```

Build a model

We'll build a small version of the Xception network. We haven't particularly tried to optimize the architecture; if you want to do a systematic search for the best model configuration, consider using Keras Tuner.

Note that:

- We start the model with the `data_augmentation` preprocessor, followed by a `Rescaling` layer.
- We include a `Dropout` layer before the final classification layer.

In [14]:

```
def MyModel(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = data_augmentation(inputs)

    # Entry block
    x = layers.experimental.preprocessing.Rescaling(1.0 / 255)(x)
    x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = tf.nn.relu6(x)

    x = layers.Conv2D(64, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = tf.nn.relu6(x)

    previous_block_activation = x  # Set aside residual

    for size in [128, 256, 512, 728]:
        x = tf.nn.relu6(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = tf.nn.relu6(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)
```

```python
        # Project residual
        residual = layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    x = layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = tf.nn.relu6(x)

    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


model = MyModel(input_shape=image_size + (3,), num_classes=2)
# keras.utils.plot_model(model, show_shapes=True)
```

In [15]:

```python
model.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 180, 180, 3) 0
__________________________________________________________________________________________________
sequential (Sequential)         (None, 180, 180, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
rescaling (Rescaling)           (None, 180, 180, 3)  0           sequential[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 90, 90, 32)   896         rescaling[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 90, 90, 32)   128         conv2d[0][0]
__________________________________________________________________________________________________
tf_op_layer_Relu6 (TensorFlowOp [(None, 90, 90, 32)] 0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 90, 90, 64)   18496       tf_op_layer_Relu6[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 90, 90, 64)   256         conv2d_1[0][0]
__________________________________________________________________________________________________
tf_op_layer_Relu6_1 (TensorFlow [(None, 90, 90, 64)] 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
tf_op_layer_Relu6_2 (TensorFlow [(None, 90, 90, 64)] 0           tf_op_layer_Relu6_1[0][0]
__________________________________________________________________________________________________
separable_conv2d (SeparableConv (None, 90, 90, 128)  8896        tf_op_layer_Relu6_2[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 90, 90, 128)  512         separable_conv2d[0][0]
__________________________________________________________________________________________________
tf_op_layer_Relu6_3 (TensorFlow [(None, 90, 90, 128) 0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
separable_conv2d_1 (SeparableCo (None, 90, 90, 128)  17664       tf_op_layer_Relu6_3[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 90, 90, 128)  512         separable_conv2d_1[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 45, 45, 128)  0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 45, 45, 128)  8320        tf_op_layer_Relu6_1[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 45, 45, 128)  0           max_pooling2d[0][0]
                                                                 conv2d_2[0][0]
__________________________________________________________________________________________________
tf_op_layer_Relu6_4 (TensorFlow [(None, 45, 45, 128) 0           add[0][0]
__________________________________________________________________________________________________
separable_conv2d_2 (SeparableCo (None, 45, 45, 256)  34176       tf_op_layer_Relu6_4[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 45, 45, 256)  1024        separable_conv2d_2[0][0]
__________________________________________________________________________________________________
tf_op_layer_Relu6_5 (TensorFlow [(None, 45, 45, 256) 0           batch_normalization_4[0][0]
```

```
________________________________________________________________________________________________
separable_conv2d_3 (SeparableCo (None, 45, 45, 256)  68096       tf_op_layer_Relu6_5[0][0]
________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 45, 45, 256)  1024        separable_conv2d_3[0][0]
________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 23, 23, 256)  0           batch_normalization_5[0][0]
________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 23, 23, 256)  33024       add[0][0]
________________________________________________________________________________________________
add_1 (Add)                     (None, 23, 23, 256)  0           max_pooling2d_1[0][0]
                                                                 conv2d_3[0][0]
________________________________________________________________________________________________
tf_op_layer_Relu6_6 (TensorFlow [(None, 23, 23, 256) 0           add_1[0][0]
________________________________________________________________________________________________
separable_conv2d_4 (SeparableCo (None, 23, 23, 512)  133888      tf_op_layer_Relu6_6[0][0]
________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 23, 23, 512)  2048        separable_conv2d_4[0][0]
________________________________________________________________________________________________
tf_op_layer_Relu6_7 (TensorFlow [(None, 23, 23, 512) 0           batch_normalization_6[0][0]
________________________________________________________________________________________________
separable_conv2d_5 (SeparableCo (None, 23, 23, 512)  267264      tf_op_layer_Relu6_7[0][0]
________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 23, 23, 512)  2048        separable_conv2d_5[0][0]
________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 12, 12, 512)  0           batch_normalization_7[0][0]
________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 12, 12, 512)  131584      add_1[0][0]
________________________________________________________________________________________________
add_2 (Add)                     (None, 12, 12, 512)  0           max_pooling2d_2[0][0]
                                                                 conv2d_4[0][0]
________________________________________________________________________________________________
tf_op_layer_Relu6_8 (TensorFlow [(None, 12, 12, 512) 0           add_2[0][0]
________________________________________________________________________________________________
separable_conv2d_6 (SeparableCo (None, 12, 12, 728)  378072      tf_op_layer_Relu6_8[0][0]
________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 12, 12, 728)  2912        separable_conv2d_6[0][0]
________________________________________________________________________________________________
tf_op_layer_Relu6_9 (TensorFlow [(None, 12, 12, 728) 0           batch_normalization_8[0][0]
________________________________________________________________________________________________
separable_conv2d_7 (SeparableCo (None, 12, 12, 728)  537264      tf_op_layer_Relu6_9[0][0]
________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 12, 12, 728)  2912        separable_conv2d_7[0][0]
________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 6, 6, 728)    0           batch_normalization_9[0][0]
________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 6, 6, 728)    373464      add_2[0][0]
________________________________________________________________________________________________
add_3 (Add)                     (None, 6, 6, 728)    0           max_pooling2d_3[0][0]
                                                                 conv2d_5[0][0]
________________________________________________________________________________________________
separable_conv2d_8 (SeparableCo (None, 6, 6, 1024)   753048      add_3[0][0]
________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 6, 6, 1024)   4096        separable_conv2d_8[0][0]
________________________________________________________________________________________________
tf_op_layer_Relu6_10 (TensorFlo [(None, 6, 6, 1024)] 0           batch_normalization_10[0][0]
________________________________________________________________________________________________
global_average_pooling2d (Globa (None, 1024)         0           tf_op_layer_Relu6_10[0][0]
________________________________________________________________________________________________
dropout (Dropout)               (None, 1024)         0           global_average_pooling2d[0][0]
________________________________________________________________________________________________
dense (Dense)                   (None, 1)            1025        dropout[0][0]
================================================================================================
Total params: 2,782,649
Trainable params: 2,773,913
Non-trainable params: 8,736
________________________________________________________________________________________________
```

Train the model

In [18]:

```
epochs = 50

callbacks = [
    keras.callbacks.ModelCheckpoint("checkpoint_model.h5", save_best_only=True),
    keras.callbacks.EarlyStopping(patience=7, restore_best_weights=True, verbose=1),
    keras.callbacks.ReduceLROnPlateau(verbose=1, min_lr=0.000001, patience=5)
```

```
]
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["binary_accuracy"],
)
model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)
```

```
Epoch 1/50
  2/586 [..............................] - ETA: 2:16 - loss: 0.8191 - binary_accuracy: 
0.5469WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time (batch 
time: 0.1212s vs `on_train_batch_end` time: 0.3494s). Check your callbacks.
586/586 [==============================] - 278s 475ms/step - loss: 0.6400 - binary_accuracy: 0.6539 - va
l_loss: 0.6422 - val_binary_accuracy: 0.6335
Epoch 2/50
586/586 [==============================] - 277s 473ms/step - loss: 0.5172 - binary_accuracy: 0.7479 - va
l_loss: 0.4836 - val_binary_accuracy: 0.7680
Epoch 3/50
586/586 [==============================] - 279s 476ms/step - loss: 0.4185 - binary_accuracy: 0.8092 - va
l_loss: 0.3724 - val_binary_accuracy: 0.8370
Epoch 4/50
586/586 [==============================] - 280s 479ms/step - loss: 0.3387 - binary_accuracy: 0.8541 - va
l_loss: 0.2619 - val_binary_accuracy: 0.8849
Epoch 5/50
586/586 [==============================] - 277s 473ms/step - loss: 0.2835 - binary_accuracy: 0.8800 - va
l_loss: 0.2185 - val_binary_accuracy: 0.9039
Epoch 6/50
586/586 [==============================] - 275s 469ms/step - loss: 0.2388 - binary_accuracy: 0.9013 - va
l_loss: 0.4023 - val_binary_accuracy: 0.8313
Epoch 7/50
586/586 [==============================] - 276s 471ms/step - loss: 0.2099 - binary_accuracy: 0.9150 - va
l_loss: 0.2465 - val_binary_accuracy: 0.8996
Epoch 8/50
586/586 [==============================] - 277s 472ms/step - loss: 0.1901 - binary_accuracy: 0.9213 - va
l_loss: 0.1560 - val_binary_accuracy: 0.9372
Epoch 9/50
586/586 [==============================] - 276s 472ms/step - loss: 0.1766 - binary_accuracy: 0.9265 - va
l_loss: 0.2296 - val_binary_accuracy: 0.9028
Epoch 10/50
586/586 [==============================] - 277s 472ms/step - loss: 0.1691 - binary_accuracy: 0.9322 - va
l_loss: 0.1395 - val_binary_accuracy: 0.9415
Epoch 11/50
586/586 [==============================] - 276s 471ms/step - loss: 0.1602 - binary_accuracy: 0.9356 - va
l_loss: 0.2196 - val_binary_accuracy: 0.9124
Epoch 12/50
586/586 [==============================] - 276s 471ms/step - loss: 0.1484 - binary_accuracy: 0.9406 - va
l_loss: 0.2722 - val_binary_accuracy: 0.8971
Epoch 13/50
586/586 [==============================] - 275s 470ms/step - loss: 0.1427 - binary_accuracy: 0.9423 - va
l_loss: 0.1578 - val_binary_accuracy: 0.9329
Epoch 14/50
586/586 [==============================] - 276s 471ms/step - loss: 0.1415 - binary_accuracy: 0.9426 - va
l_loss: 0.1829 - val_binary_accuracy: 0.9267
Epoch 15/50
586/586 [==============================] - ETA: 0s - loss: 0.1356 - binary_accuracy: 0.9453
Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
586/586 [==============================] - 276s 470ms/step - loss: 0.1356 - binary_accuracy: 0.9453 - va
l_loss: 0.1504 - val_binary_accuracy: 0.9383
Epoch 16/50
586/586 [==============================] - 277s 473ms/step - loss: 0.0957 - binary_accuracy: 0.9623 - va
l_loss: 0.0925 - val_binary_accuracy: 0.9628
Epoch 17/50
586/586 [==============================] - 277s 473ms/step - loss: 0.0793 - binary_accuracy: 0.9695 - va
l_loss: 0.1016 - val_binary_accuracy: 0.9643
Epoch 18/50
586/586 [==============================] - 277s 473ms/step - loss: 0.0787 - binary_accuracy: 0.9689 - va
l_loss: 0.0832 - val_binary_accuracy: 0.9656
Epoch 19/50
586/586 [==============================] - 277s 473ms/step - loss: 0.0714 - binary_accuracy: 0.9720 - va
l_loss: 0.0909 - val_binary_accuracy: 0.9663
Epoch 20/50
586/586 [==============================] - 277s 472ms/step - loss: 0.0746 - binary_accuracy: 0.9708 - va
l_loss: 0.0904 - val_binary_accuracy: 0.9660
Epoch 21/50
586/586 [==============================] - 276s 471ms/step - loss: 0.0692 - binary_accuracy: 0.9731 - va
l_loss: 0.0842 - val_binary_accuracy: 0.9680
```

```
l_loss: 0.0842 - val_binary_accuracy: 0.9680
Epoch 22/50
586/586 [==============================] - 277s 473ms/step - loss: 0.0653 - binary_accuracy: 0.9745 - va
l_loss: 0.0912 - val_binary_accuracy: 0.9660
Epoch 23/50
586/586 [==============================] - ETA: 0s - loss: 0.0617 - binary_accuracy: 0.9762
Epoch 00023: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
586/586 [==============================] - 277s 472ms/step - loss: 0.0617 - binary_accuracy: 0.9762 - va
l_loss: 0.0926 - val_binary_accuracy: 0.9652
Epoch 24/50
586/586 [==============================] - 279s 477ms/step - loss: 0.0571 - binary_accuracy: 0.9777 - va
l_loss: 0.0810 - val_binary_accuracy: 0.9677
Epoch 25/50
586/586 [==============================] - 279s 476ms/step - loss: 0.0543 - binary_accuracy: 0.9791 - va
l_loss: 0.0820 - val_binary_accuracy: 0.9671
Epoch 26/50
586/586 [==============================] - 279s 475ms/step - loss: 0.0531 - binary_accuracy: 0.9809 - va
l_loss: 0.0813 - val_binary_accuracy: 0.9680
Epoch 27/50
586/586 [==============================] - 279s 477ms/step - loss: 0.0565 - binary_accuracy: 0.9787 - va
l_loss: 0.0827 - val_binary_accuracy: 0.9684
Epoch 28/50
586/586 [==============================] - 279s 476ms/step - loss: 0.0553 - binary_accuracy: 0.9794 - va
l_loss: 0.0812 - val_binary_accuracy: 0.9692
Epoch 29/50
586/586 [==============================] - ETA: 0s - loss: 0.0549 - binary_accuracy: 0.9785
Epoch 00029: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.
586/586 [==============================] - 280s 477ms/step - loss: 0.0549 - binary_accuracy: 0.9785 - va
l_loss: 0.0834 - val_binary_accuracy: 0.9682
Epoch 30/50
586/586 [==============================] - 278s 475ms/step - loss: 0.0559 - binary_accuracy: 0.9800 - va
l_loss: 0.0821 - val_binary_accuracy: 0.9686
Epoch 31/50
586/586 [==============================] - ETA: 0s - loss: 0.0524 - binary_accuracy: 0.9799Restoring
model weights from the end of the best epoch.
586/586 [==============================] - 280s 478ms/step - loss: 0.0524 - binary_accuracy: 0.9799 - va
l_loss: 0.0822 - val_binary_accuracy: 0.9688
Epoch 00031: early stopping
```

Out[18]:

```
<tensorflow.python.keras.callbacks.History at 0x7fbf3d5486d8>
```

We get to ~97% validation accuracy after training for 30 epochs on the full dataset.

In [21]:

```python
loss, acc = model.evaluate(train_ds, verbose=0)
print("Training Loss: {:.3} \t Training Accuracy: {:.3}".format(loss, acc))
loss, acc = model.evaluate(val_ds, verbose=0)
print("Training Loss: {:.3} \t Training Accuracy: {:.3}".format(loss, acc))
```

```
Training Loss: 0.0389   Training Accuracy: 0.987
Training Loss: 0.081   Training Accuracy: 0.968
```

In [22]:

```python
clf = tf.keras.models.load_model('checkpoint_model.h5')
loss, acc = clf.evaluate(train_ds, verbose=0)
print('Training Loss: {:.2} \t Training Accuracy {:.3}'.format(loss, acc))
loss, acc = clf.evaluate(val_ds, verbose=0)
print('Testing Loss: {:.2} \t Testing Accuracy {:.3}'.format(loss, acc))
```

```
Training Loss: 0.039   Training Accuracy 0.987
Testing Loss: 0.081   Testing Accuracy 0.968
```

In [24]:

```python
json_model = model.to_json()
with open('model.json', 'w') as json_file:
  json_file.write(json_model)


yaml_model = model.to_yaml()
with open('model.yaml', 'w') as yaml_file:
  yaml_file.write(yaml_model)

model.save('model.h5')
model.save_weights('model_weights.h5')
print('Done')
```

```
Done
```

Run inference on new data

Note that data augmentation and dropout are inactive at inference time.

In [25]:

```
img = keras.preprocessing.image.load_img(
    "PetImages/Cat/6779.jpg", target_size=image_size
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
score = predictions[0]
print(
    "This image is %.2f percent cat and %.2f percent dog."
    % (100 * (1 - score), 100 * score)
)
```

```
This image is 89.60 percent cat and 10.40 percent dog.
```

In [ ]: