```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/horse-or-human.zip \
    -O /tmp/horse-or-human.zip

!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/validation-horse-or-human
    -O /tmp/validation-horse-or-human.zip

import os
import zipfile

local_zip = '/tmp/horse-or-human.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp/horse-or-human')
local_zip = '/tmp/validation-horse-or-human.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp/validation-horse-or-human')
zip_ref.close()
# Directory with our training horse pictures
train_horse_dir = os.path.join('/tmp/horse-or-human/horses')

# Directory with our training human pictures
/horse-or-human/humans')

# Directory with our training horse pictures
validation_horse_dir = os.path.join('/tmp/validation-horse-or-human/horses')

# Directory with our training human pictures
validation_human_dir = os.path.join('/tmp/validation-horse-or-human/humans')
```

Saved successfully!

```
--2020-09-16 19:25:15--  https://storage.googleapis.com/laurencemoroney-blog.appspot.co
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.126.128, 108.177.1
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.126.128|:443... c
HTTP request sent, awaiting response... 200 OK
Length: 149574867 (143M) [application/zip]
Saving to: '/tmp/horse-or-human.zip'

/tmp/horse-or-human 100%[===================>] 142.65M  45.1MB/s    in 3.2s
```

Building a Small Model from Scratch

But before we continue, let's start defining the model:

Step 1 will be to import tensorflow.

```
Length: 11480187 (11M) [application/zip]
```

```
import tensorflow as tf
```

```
/tmp/validation-hor 100%[===================>]  10.95M  32.5MB/s    in 0.3s
```

We then add convolutional layers as in the previous example, and flatten the final result to feed into the densely connected layers.

Finally we add the densely connected layers.

Note that because we are facing a two-class classification problem, i.e. a *binary classification problem*, we will end our network with a [*sigmoid* activation](), so that the output of our network will be a single scalar between 0 and 1, encoding the probability that the current image is class 1 (as opposed to class 0).

Saved successfully!

```
    # Note the input shape is the desired size of the image 300x300 with 3 bytes color
    # This is the first convolution
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(300, 300, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fourth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fifth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
```

```
    # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('horses') a
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=1e-4),
              metrics=['accuracy'])
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1/255)

# Flow training images in batches of 128 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        '/tmp/horse-or-human/',  # This is the source directory for training images
                                   ll images will be resized to 150x150
Saved successfully!
                                   ntropy loss, we need binary labels
        class_mode='binary')

# Flow training images in batches of 128 using train_datagen generator
validation_generator = validation_datagen.flow_from_directory(
        '/tmp/validation-horse-or-human/',  # This is the source directory for training image
        target_size=(300, 300),  # All images will be resized to 150x150
        batch_size=32,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')
```

```
Found 1027 images belonging to 2 classes.
Found 256 images belonging to 2 classes.
```

```
history = model.fit(
      train_generator,
      steps_per_epoch=8,
      epochs=100,
      verbose=1,
      validation_data = validation_generator,
```

```
      validation_steps=8)
```

Saved successfully!

```
Epoch 72/100
8/8 [==============================] - 20s 2s/step - loss: 0.2194 - accuracy: 0.9110 - v
Epoch 73/100
8/8 [==============================] - 20s 2s/step - loss: 0.3002 - accuracy: 0.8721 - v
Epoch 74/100
8/8 [==============================] - 20s 2s/step - loss: 0.2068 - accuracy: 0.9121 - v
Epoch 75/100
8/8 [==============================] - 20s 2s/step - loss: 0.2701 - accuracy: 0.8754 - v
Epoch 76/100
8/8 [==============================] - 20s 3s/step - loss: 0.2128 - accuracy: 0.9244 - v
Epoch 77/100
8/8 [==============================] - 20s 3s/step - loss: 0.1955 - accuracy: 0.9221 - v
Epoch 78/100
8/8 [==============================] - 20s 3s/step - loss: 0.2653 - accuracy: 0.8865 - v
Epoch 79/100
8/8 [==============================] - 20s 2s/step - loss: 0.1775 - accuracy: 0.9399 - v
Epoch 80/100
8/8 [==============================] - 20s 2s/step - loss: 0.2421 - accuracy: 0.9010 - v
Epoch 81/100
8/8 [==============================] - 20s 2s/step - loss: 0.2065 - accuracy: 0.9199 - v
Epoch 82/100
8/8 [==============================] - 20s 2s/step - loss: 0.1818 - accuracy: 0.9255 - v
Epoch 83/100
8/8 [==============================] - 20s 2s/step - loss: 0.2288 - accuracy: 0.9043 - v
Epoch 84/100
8/8 [==============================] - 22s 3s/step - loss: 0.2646 - accuracy: 0.8999 - v
Epoch 85/100
8/8 [==============================] - 22s 3s/step - loss: 0.1663 - accuracy: 0.9297 - v
Epoch 86/100
8/8 [==============================] - 20s 2s/step - loss: 0.1539 - accuracy: 0.9444 - v
Epoch 87/100
8/8 [==============================] - 22s 3s/step - loss: 0.2130 - accuracy: 0.9150 - v
Epoch 88/100
====] - 20s 2s/step - loss: 0.2143 - accuracy: 0.9166 - v
8/8 [==============================] - 20s 2s/step - loss: 0.2218 - accuracy: 0.9099 - v
Epoch 90/100
8/8 [==============================] - 20s 2s/step - loss: 0.1674 - accuracy: 0.9388 - v
Epoch 91/100
8/8 [==============================] - 20s 2s/step - loss: 0.2285 - accuracy: 0.9088 - v
Epoch 92/100
8/8 [==============================] - 20s 2s/step - loss: 0.2083 - accuracy: 0.9232 - v
Epoch 93/100
8/8 [==============================] - 20s 2s/step - loss: 0.1607 - accuracy: 0.9388 - v
Epoch 94/100
8/8 [==============================] - 20s 2s/step - loss: 0.2297 - accuracy: 0.9088 - v
Epoch 95/100
8/8 [==============================] - 20s 2s/step - loss: 0.3468 - accuracy: 0.8932 - v
Epoch 96/100
8/8 [==============================] - 20s 2s/step - loss: 0.1528 - accuracy: 0.9377 - v
Epoch 97/100
8/8 [==============================] - 22s 3s/step - loss: 0.1412 - accuracy: 0.9492 - v
Epoch 98/100
8/8 [==============================] - 20s 2s/step - loss: 0.2245 - accuracy: 0.9032 - v
Epoch 99/100
8/8 [==============================] - 22s 3s/step - loss: 0.1212 - accuracy: 0.9611 - v
Epoch 100/100
```

Saved successfully!

```
8/8 [==============================] - 22s 3s/step - loss: 0.1902 - accuracy: 0.9229 - v
```

Saved successfully!