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

```
--2020-03-12 18:04:22--  https://storage.googleapis.com/laurencemoroney-blog.a
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.14.112, 2
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.14.112|:
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   101MB/s    in 1.4s

2020-03-12 18:04:24 (101 MB/s) - '/tmp/horse-or-human.zip' saved [149574867/14
```

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

```
--2020-03-12 18:04:30--  https://storage.googleapis.com/laurencemoroney-blog.a
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.11.176, 2
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.11.176|:
HTTP request sent, awaiting response... 200 OK
Length: 11480187 (11M) [application/zip]
Saving to: '/tmp/validation-horse-or-human.zip'

/tmp/validation-hor 100%[===================>]  10.95M  --.-KB/s    in 0.03s

2020-03-12 18:04:30 (376 MB/s) - '/tmp/validation-horse-or-human.zip' saved [1
```

The following python code will use the OS library to use Operating System libraries, giving you acc allowing you to unzip the data.

```
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()
```

The contents of the .zip are extracted to the base directory `/tmp/horse-or-human`, which in turn subdirectories.

In short: The training set is the data that is used to tell the neural network model that 'this is what a like' etc.

One thing to pay attention to in this sample: We do not explicitly label the images as horses or hun example earlier, we had labelled 'this is a 1', 'this is a 7' etc. Later you'll see something called an Im to read images from subdirectories, and automatically label them from the name of that subdirect directory containing a 'horses' directory and a 'humans' one. ImageGenerator will label the images

Let's define each of these directories:

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

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

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

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

Building a Small Model from Scratch

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

Step 1 will be to import tensorflow.

```
import tensorflow as tf
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%t`

We then add convolutional layers as in the previous example, and flatten the final result to feed int

Finally we add the densely connected layers.

Note that because we are facing a two-class classification problem, i.e. a *binary classification prob* activation, so that the output of our network will be a single scalar between 0 and 1, encoding the (as opposed to class 0).

```
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 by
    # This is the first convolution
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 15
    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)
```

```
    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 cla
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

The model.summary() method call prints a summary of the NN

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 18496)             0
_________________________________________________________________
dense (Dense)                (None, 512)               9470464
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

The "output shape" column shows how the size of your feature map evolves in each successive la
the feature maps by a bit due to padding, and each pooling layer halves the dimensions.

Next, we'll configure the specifications for model training. We will train our model with the `binary_` classification problem and our final activation is a sigmoid. (For a refresher on loss metrics, see th use the `rmsprop` optimizer with a learning rate of `0.001`. During training, we will want to monitor

NOTE: In this case, using the RMSprop optimization algorithm is preferable to stochastic gradient learning-rate tuning for us. (Other optimizers, such as Adam and Adagrad, also automatically adap work equally well here.)

```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['acc'])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

Data Preprocessing

Let's set up data generators that will read pictures in our source folders, convert them to `float32` our network. We'll have one generator for the training images and one for the validation images. Ou size 300x300 and their labels (binary).

As you may already know, data that goes into neural networks should usually be normalized in son processing by the network. (It is uncommon to feed raw pixels into a convnet.) In our case, we will pixel values to be in the `[0, 1]` range (originally all values are in the `[0, 255]` range).

In Keras this can be done via the `keras.preprocessing.image.ImageDataGenerator` class usin `ImageDataGenerator` class allows you to instantiate generators of augmented image batches (a `.flow_from_directory(directory)`. These generators can then be used with the Keras model m inputs: `fit_generator`, `evaluate_generator`, and `predict_generator`.

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

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1/255)
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 i
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=128,
        # Since we use binary_crossentropy 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
        target_size=(150, 150),  # 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.
```

Training

Let's train for 15 epochs -- this may take a few minutes to run.

Do note the values per epoch.

The Loss and Accuracy are a great indication of progress of training. It's making a guess as to the measuring it against the known label, calculating the result. Accuracy is the portion of correct gue

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=8,
      epochs=15,
      verbose=1,
      validation_data = validation_generator,
      validation_steps=8)
```

```
8/8 [==============================] - 1s 91ms/step - loss: 0.4881 - acc: 0.82
8/8 [==============================] - 9s 1s/step - loss: 2.6206 - acc: 0.5473
Epoch 2/15
6/8 [=====================>........] - ETA: 1s - loss: 0.4942 - acc: 0.7344Epo
8/8 [==============================] - 1s 133ms/step - loss: 0.6018 - acc: 0.7
8/8 [==============================] - 5s 674ms/step - loss: 0.4540 - acc: 0.7
Epoch 3/15
7/8 [=========================>....] - ETA: 0s - loss: 0.6742 - acc: 0.8392Epo
8/8 [==============================] - 1s 129ms/step - loss: 1.5070 - acc: 0.6
8/8 [==============================] - 5s 657ms/step - loss: 0.6106 - acc: 0.8
Epoch 4/15
7/8 [=========================>....] - ETA: 0s - loss: 0.1917 - acc: 0.9252Epo
8/8 [==============================] - 1s 136ms/step - loss: 1.3144 - acc: 0.7
8/8 [==============================] - 6s 739ms/step - loss: 0.1875 - acc: 0.9
Epoch 5/15
7/8 [=========================>....] - ETA: 0s - loss: 0.1785 - acc: 0.9053Epo
8/8 [==============================] - 1s 134ms/step - loss: 1.0523 - acc: 0.8
8/8 [==============================] - 5s 651ms/step - loss: 0.1678 - acc: 0.9
Epoch 6/15
7/8 [=========================>....] - ETA: 0s - loss: 0.0798 - acc: 0.9650Epo
8/8 [==============================] - 1s 130ms/step - loss: 1.0680 - acc: 0.8
8/8 [==============================] - 5s 653ms/step - loss: 0.0833 - acc: 0.9
Epoch 7/15
7/8 [=========================>....] - ETA: 0s - loss: 0.0744 - acc: 0.9650Epo
8/8 [==============================] - 1s 133ms/step - loss: 2.5921 - acc: 0.6
8/8 [==============================] - 5s 655ms/step - loss: 0.1387 - acc: 0.9
Epoch 8/15
7/8 [=========================>....] - ETA: 0s - loss: 0.2048 - acc: 0.8872Epo
8/8 [==============================] - 1s 127ms/step - loss: 1.4794 - acc: 0.8
8/8 [==============================] - 5s 565ms/step - loss: 0.1853 - acc: 0.8
Epoch 9/15
7/8 [=========================>....] - ETA: 0s - loss: 0.0703 - acc: 0.9710Epo
8/8 [==============================] - 1s 131ms/step - loss: 1.4551 - acc: 0.7
8/8 [==============================] - 6s 729ms/step - loss: 0.0681 - acc: 0.9
Epoch 10/15
7/8 [=========================>....] - ETA: 0s - loss: 0.0136 - acc: 1.0000Epo
8/8 [==============================] - 1s 127ms/step - loss: 1.9596 - acc: 0.7
8/8 [==============================] - 5s 661ms/step - loss: 0.0131 - acc: 1.0
Epoch 11/15
7/8 [=========================>....] - ETA: 0s - loss: 0.0135 - acc: 0.9974Epo
8/8 [==============================] - 1s 132ms/step - loss: 1.8790 - acc: 0.8
8/8 [==============================] - 5s 672ms/step - loss: 0.0123 - acc: 0.9
Epoch 12/15
7/8 [=========================>....] - ETA: 0s - loss: 0.0042 - acc: 1.0000Epo
8/8 [==============================] - 1s 134ms/step - loss: 2.1995 - acc: 0.8
8/8 [==============================] - 6s 728ms/step - loss: 0.0040 - acc: 1.0
Epoch 13/15
7/8 [=========================>....] - ETA: 0s - loss: 9.5277e-04 - acc: 1.000
8/8 [==============================] - 1s 134ms/step - loss: 2.4533 - acc: 0.8
8/8 [==============================] - 5s 575ms/step - loss: 9.6421e-04 - acc:
Epoch 14/15
7/8 [=========================>....] - ETA: 0s - loss: 0.3477 - acc: 0.8772Epo
8/8 [==============================] - 1s 132ms/step - loss: 1.0312 - acc: 0.7
8/8 [==============================] - 6s 732ms/step - loss: 0.3178 - acc: 0.8
Epoch 15/15
7/8 [=========================>....] - ETA: 0s - loss: 0.1619 - acc: 0.9416Epo
8/8 [==============================] - 1s 131ms/step - loss: 1.7745 - acc: 0.8
8/8 [==============================] - 5s 658ms/step - loss: 0.1443 - acc: 0.9
```

Running the Model

Let's now take a look at actually running a prediction using the model. This code will allow you to c
will then upload them, and run them through the model, giving an indication of whether the object

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(classes[0])
  if classes[0]>0.5:
    print(fn + " is a human")
  else:
    print(fn + " is a horse")

```

Choose Files jpgggg.jpg

- **jpgggg.jpg**(image/jpeg) - 28499 bytes, last modified: 28/11/2019 - 100% done

```
Saving jpgggg.jpg to jpgggg.jpg
[0.]
jpgggg.jpg is a horse
```

Visualizing Intermediate Representations

To get a feel for what kind of features our convnet has learned, one fun thing to do is to visualize h
through the convnet.

Let's pick a random image from the training set, and then generate a figure where each row is the c
a specific filter in that output feature map. Rerun this cell to generate intermediate representations

```
import numpy as np
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img

# Let's define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
successive_outputs = [layer.output for layer in model.layers[1:]]
```

```
#visualization_model = Model(img_input, successive_outputs)
visualization_model = tf.keras.models.Model(inputs = model.input, outputs = su
# Let's prepare a random input image from the training set.
horse_img_files = [os.path.join(train_horse_dir, f) for f in train_horse_names
human_img_files = [os.path.join(train_human_dir, f) for f in train_human_names
img_path = random.choice(horse_img_files + human_img_files)

img = load_img(img_path, target_size=(300, 300))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# Rescale by 1/255
x /= 255

# Let's run our image through our network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model.layers]

# Now let's display our representations
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
  if len(feature_map.shape) == 4:
    # Just do this for the conv / maxpool layers, not the fully-connected laye
    n_features = feature_map.shape[-1]  # number of features in feature map
    # The feature map has shape (1, size, size, n_features)
    size = feature_map.shape[1]
    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))
    for i in range(n_features):
      # Postprocess the feature to make it visually palatable
      x = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std()
      x *= 64
      x += 128
      x = np.clip(x, 0, 255).astype('uint8')
      # We'll tile each filter into this big horizontal grid
      display_grid[:, i * size : (i + 1) * size] = x
    # Display the grid
    scale = 20. / n_features
    plt.figure(figsize=(scale * n_features, scale))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

--NORMAL--

As you can see we go from the raw pixels of the images to increasingly abstract and compact repr downstream start highlighting what the network pays attention to, and they show fewer and fewer zero. This is called "sparsity." Representation sparsity is a key feature of deep learning.

These representations carry increasingly less information about the original pixels of the image, bu

Clean Up

Before running the next exercise, run the following cell to terminate the kernel and free memory re:

```
import os, signal
os.kill(os.getpid(), signal.SIGKILL)
```