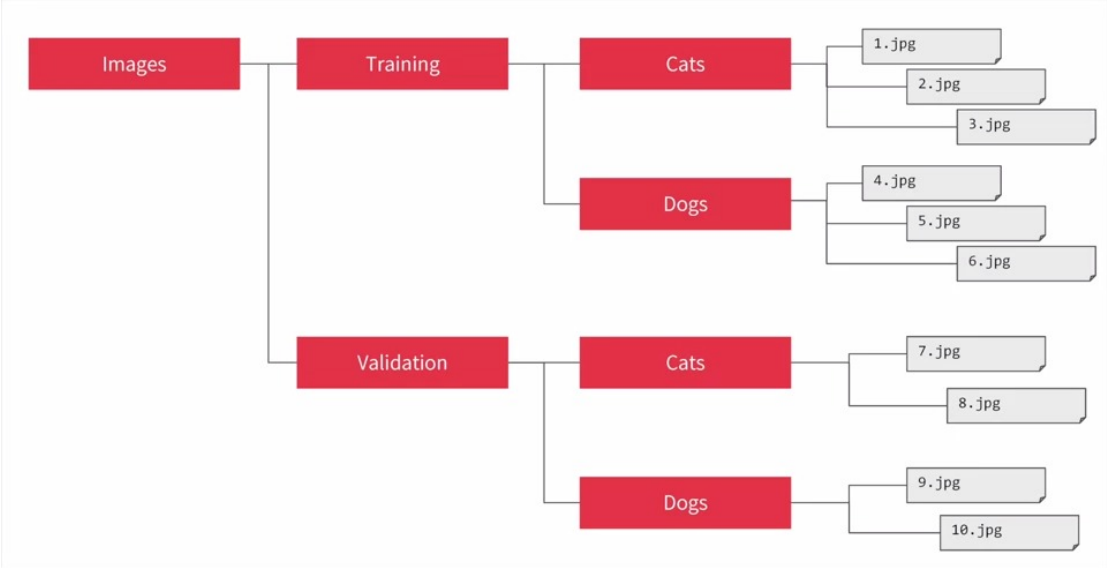
1.1 Training with the cats vs. dogs dataset

One of the nice things with TensorFlow and Keras is that if you put your images into named subdirectories, an image generated will auto label them for you. So the cats and dogs dataset you could actually do that and you've already got a massive head start in building the classifier. Then you can subdivide that into a training set and a validation set.
https://www.kaggle.com/c/dogs-vs-cats



```
test_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

deeplearning.ai

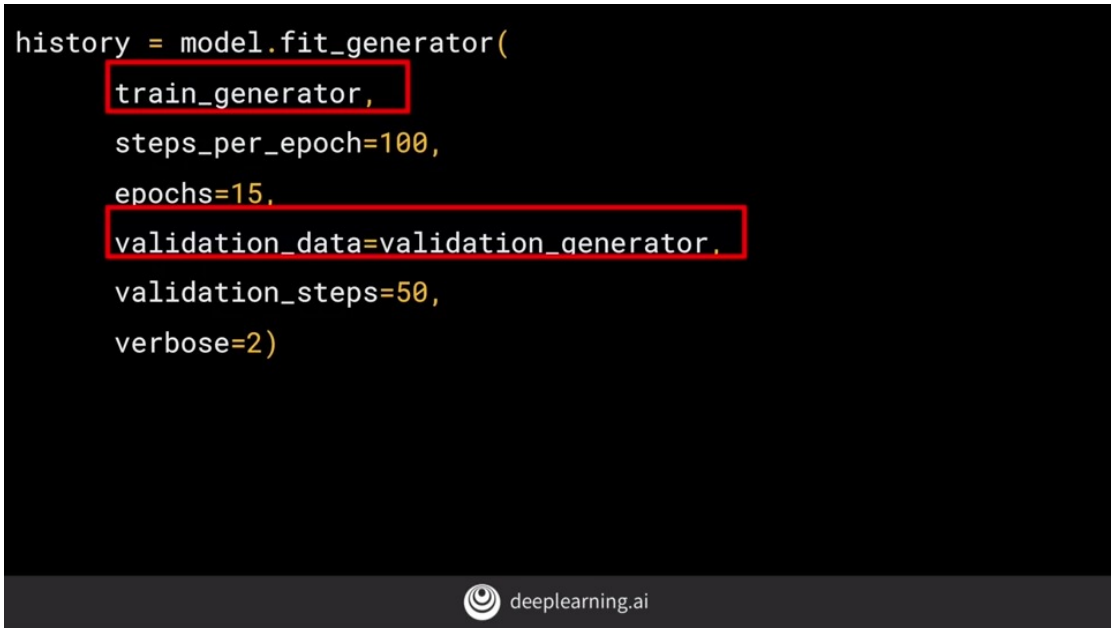
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu',
                           input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

deeplearning.ai

```
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
```



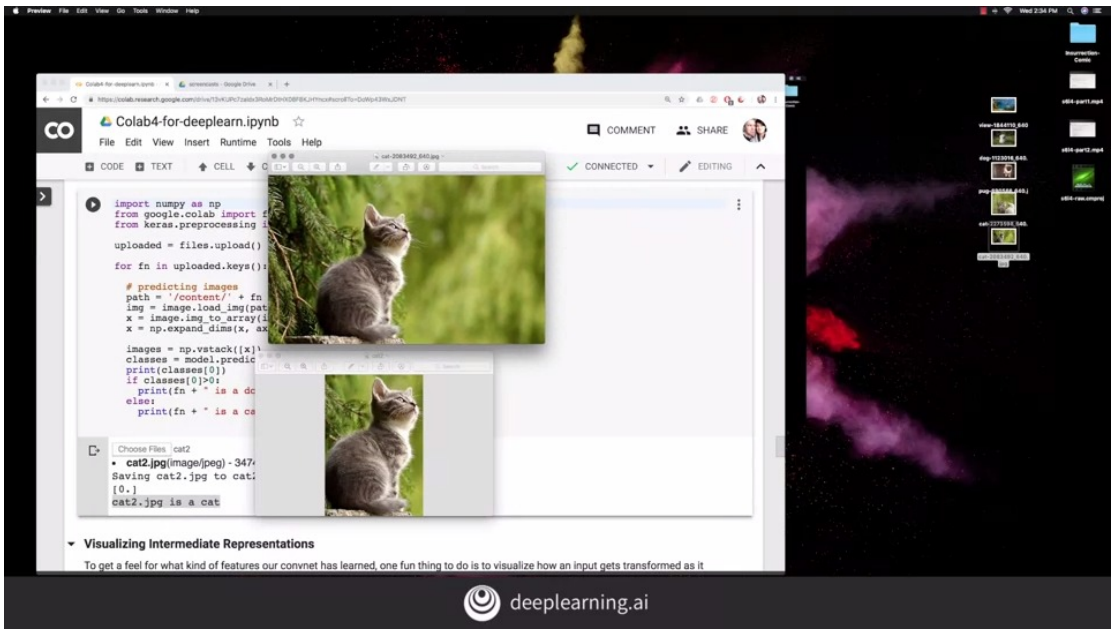
```
from tensorflow.keras.optimizers import RMSprop

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



1.2 Working through the notebook

Course 2 - Part 4 - Lesson 2 - Notebook.ipynb

1.3 Fixing through cropping



Let me show you what I did in the case of the cat my model thought was a dog.

I'll upload this image to see how it classifies. It's a crop of the cats, and lo and behold, it classifies as a cat. Let's open it, and compare it to the original image, and we'll see that just by cropping I was able to get it to change its classification. There must have been something in the uncropped image that matched features with a dog. Now I thought that was a very

interesting experiment, didn't you? Now what do you think the impact of cropping might've had on training? Would that have trained the model to show that this was a cat better than an uncropped image.

1.4 Visualizing the effect of the convolutions

Let's return to the notebook and take a look at the code that plots the outputs of the convolutions in max pulling layers.

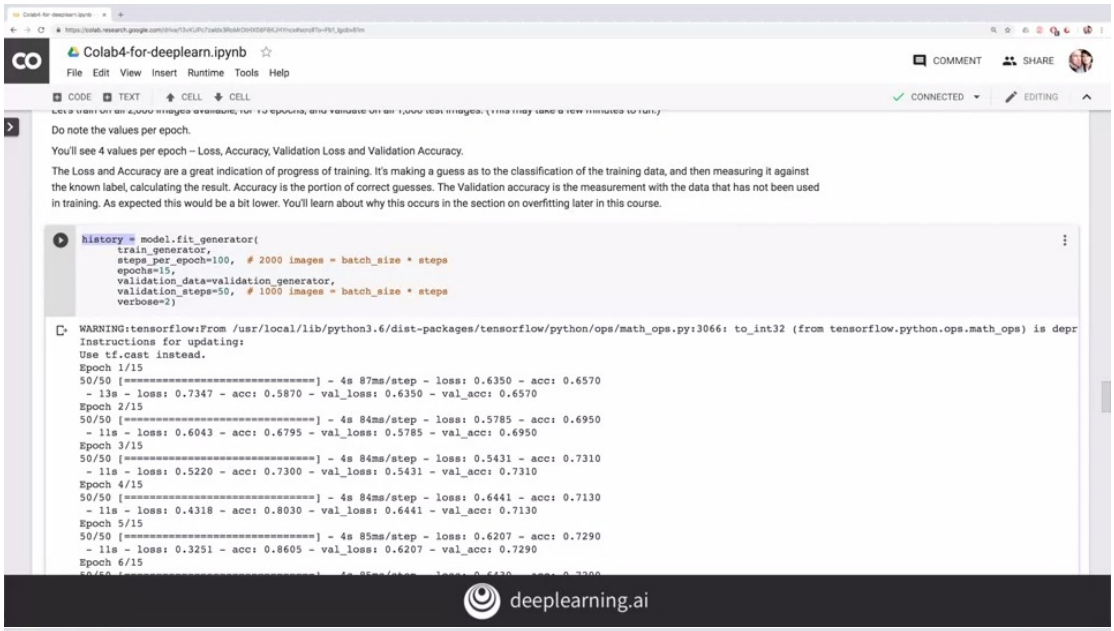
The key to this is understanding the model.layers API, which allows you to find the outputs and iterate through them, creating a visualization model for each one.

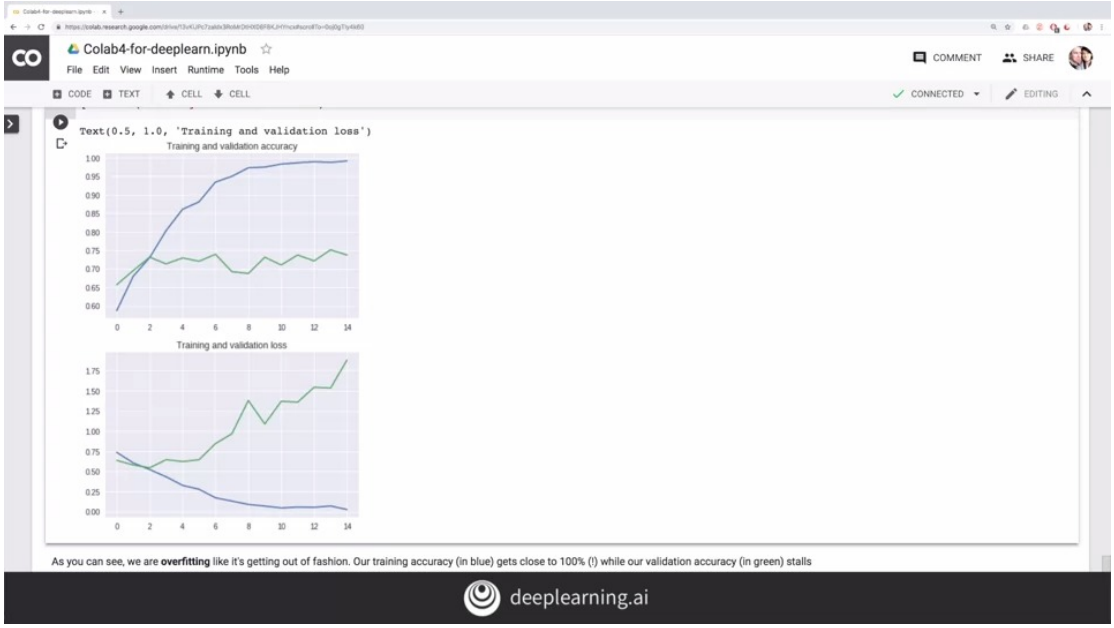
You can see images such as the dog's nose being highlighted in many of the images on the left.

1.5 Looking at accuracy and loss

let's have a quick look at plotting the learning history of this model. The object has training accuracy and loss values as well as validation accuracy and validation loss values. So let's iterate over these and plot them.

Now, if you look closely, we didn't just call **model.fit**, we said **history equals model.fit**. So we now have **a history object** that we can query for data.





At the end of the last video you saw how to explore the training history and discovered an interesting phenomenon: Even though the training data set's accuracy went very high, we saw that after only a few epochs, the validation set levelled out. This is a clear sign that we are overfitting again. Using more data should help with this, but there are some other techniques that you can use with smaller data sets too.