

# Before you Begin: TensorFlow 2.0 and this Course

We've gotten a lot of questions about whether this course teaches TensorFlow 2.0 or 1.x.

The answer is: Both!

We've designed the curriculum for the early modules in this course to have code that's generic enough that it works with both versions, so you are welcome to try it with the 2.0 alpha. If you are using the colabs, you'll use the latest version of TensorFlow that Google Colaboratory supports, which, at time of writing is 1.13. You can replace this with 2.0 by adding a codeblock containing:

```
1 !pip install tensorflow==2.0.0-alpha0
```

...and you should be able to use TF2.0 alpha if you like!

Later modules may use specific versions of libraries, some of which may require TF2.0, and we will note them when you get there!

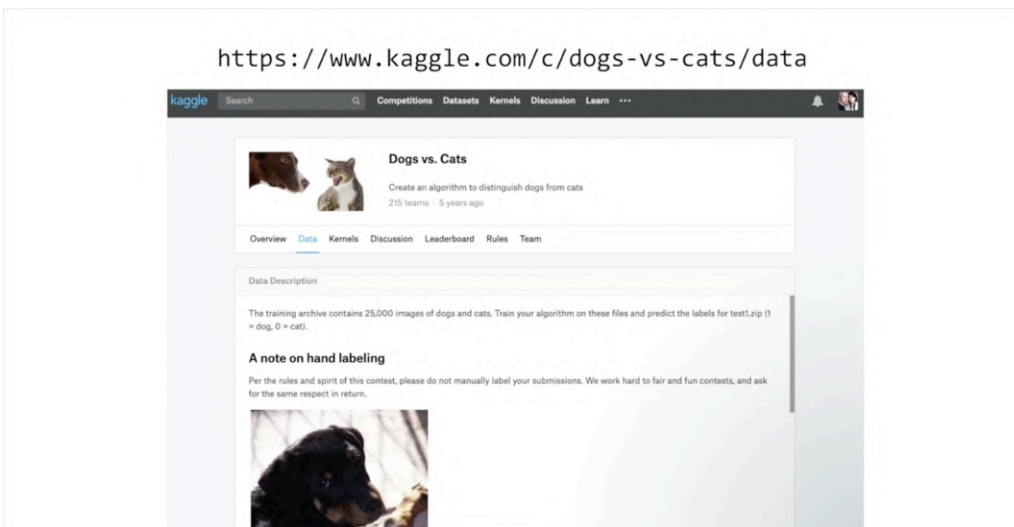
## The cats vs dogs dataset

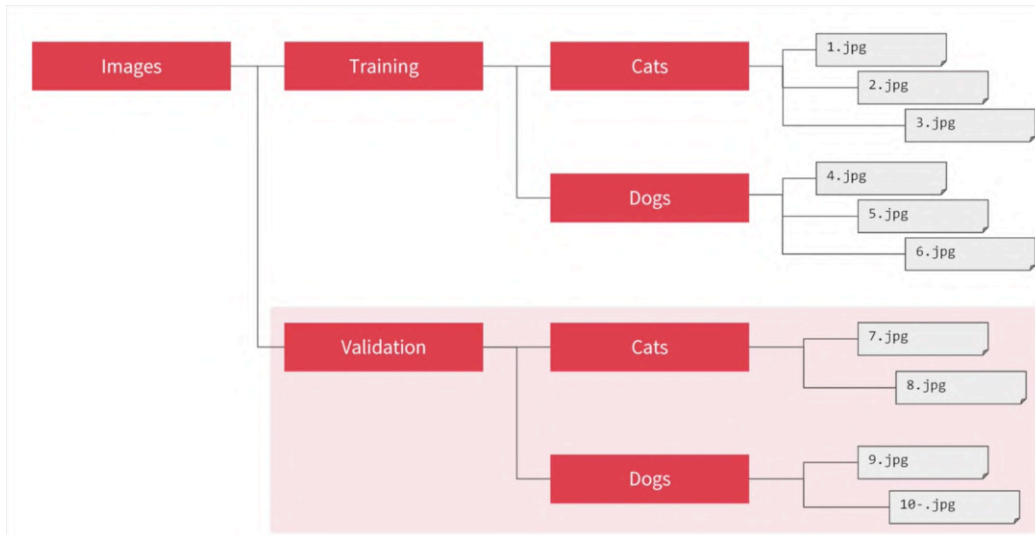
In the next video, you'll look at the famous Kaggle Dogs v Cats dataset: <https://www.kaggle.com/c/dogs-vs-cats>

This was originally a challenge in building a classifier aimed at the world's best Machine Learning and AI Practitioners, but the technology has advanced so quickly, you'll see how you can do it in just a few minutes with some simple Convolutional Neural Network programming.

It's also a nice exercise in looking at a larger dataset, downloading and preparing it for training, as well as handling some preprocessing of data. Even data like this which has been carefully curated for you can have errors -- as you'll notice with some corrupt images!

Also, you may notice some warnings about missing or corrupt EXIF data as the images are being loaded into the model for training. Don't worry about this -- it won't impact your model! :)





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

```
train_datagen = ImageDataGenerator(rescale=1./255)  
train_generator = train_datagen.flow_from_directory(  
    train_dir,  
    target_size=(150, 150),  
    batch_size=20,  
    class_mode='binary')
```

```
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

```
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

```
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

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

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

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

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

```
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')
```

```

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')
])

```

```

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')
])

```

```

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')
])

```

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 (MaxPooling2D)	(None, 36, 36, 32)	0
conv2d_2 (Conv2D)	(None, 34, 34, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(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'])

```

```

history = model.fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=15,
    validation_data=validation_generator,
    validation_steps=50,
    verbose=2)

```

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. And we'll see them in next week's lesson!

