




Copyright Notice


These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>

https://www.kaggle.com/c/dogs-vs-cats/

 [Competitions](#) [Datasets](#) [Kernels](#) [Discussion](#) [Learn](#) [...](#)  



Dogs vs. Cats

Create an algorithm to distinguish dogs from cats

215 teams · 5 years ago

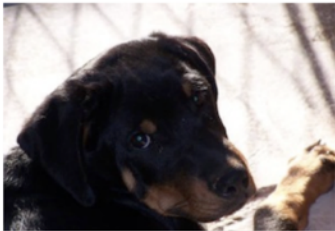
[Overview](#) [Data](#) [Kernels](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#)

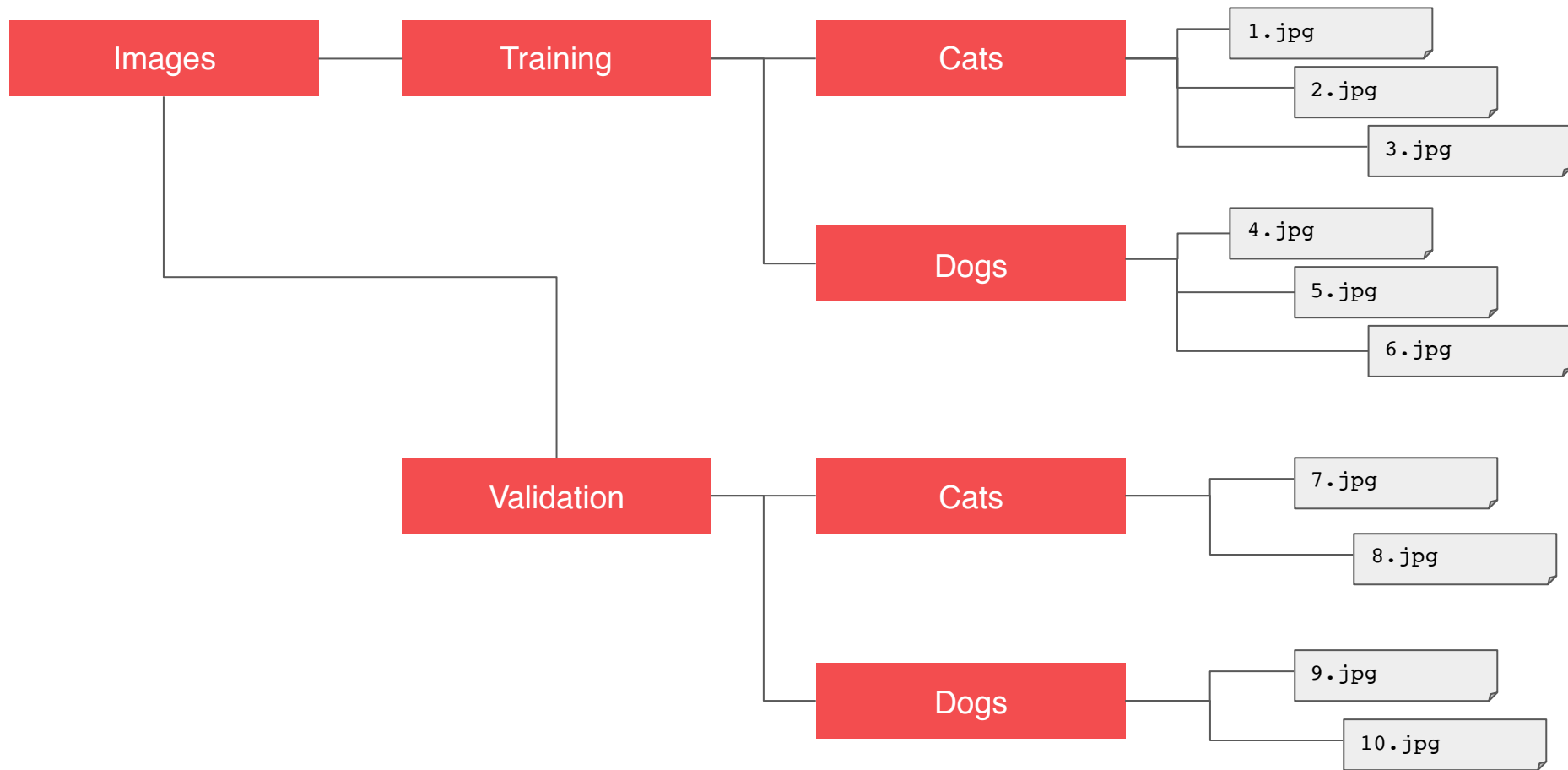
Data Description

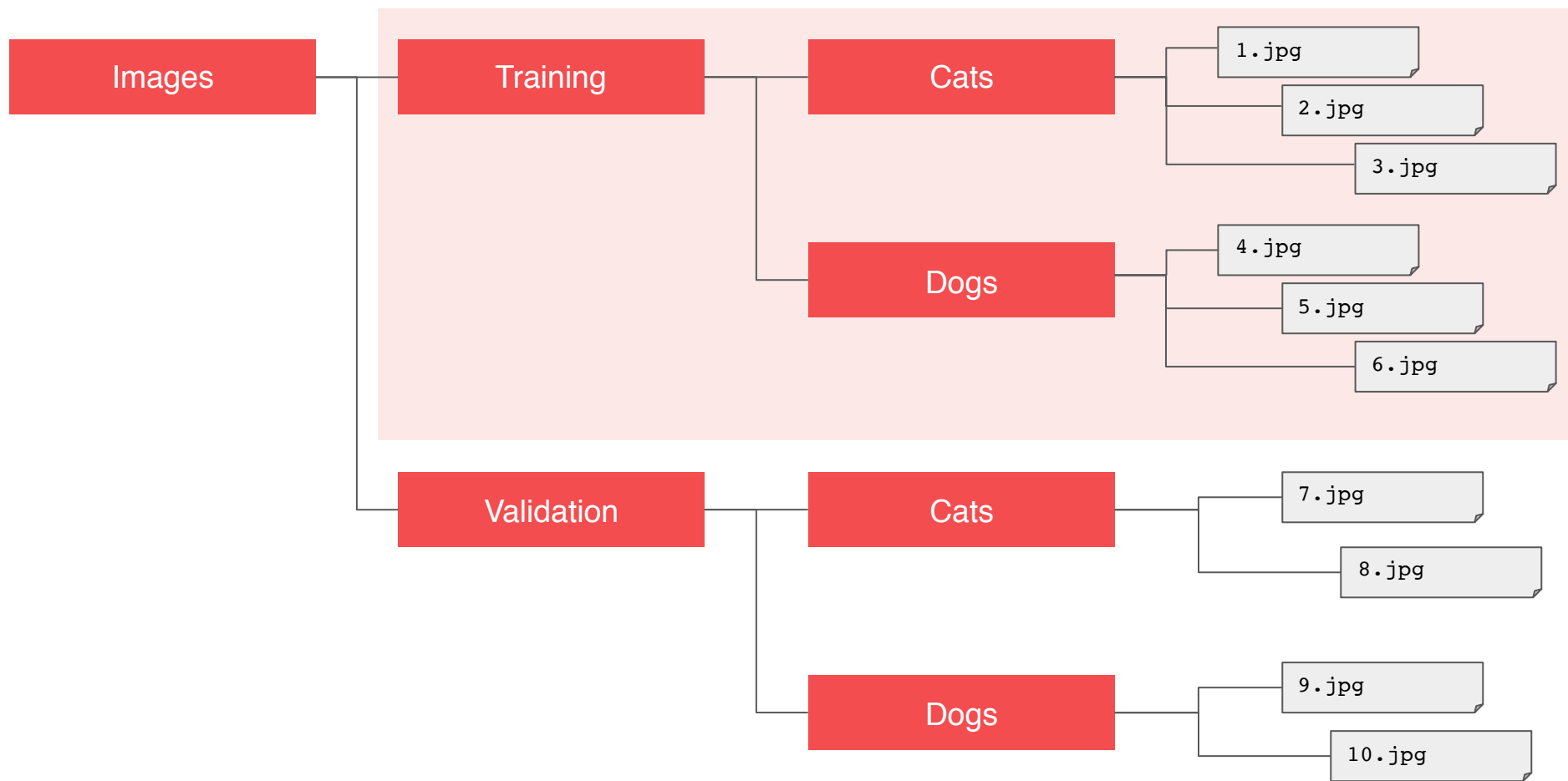
The training archive contains 25,000 images of dogs and cats. Train your algorithm on these files and predict the labels for test1.zip (1 = dog, 0 = cat).

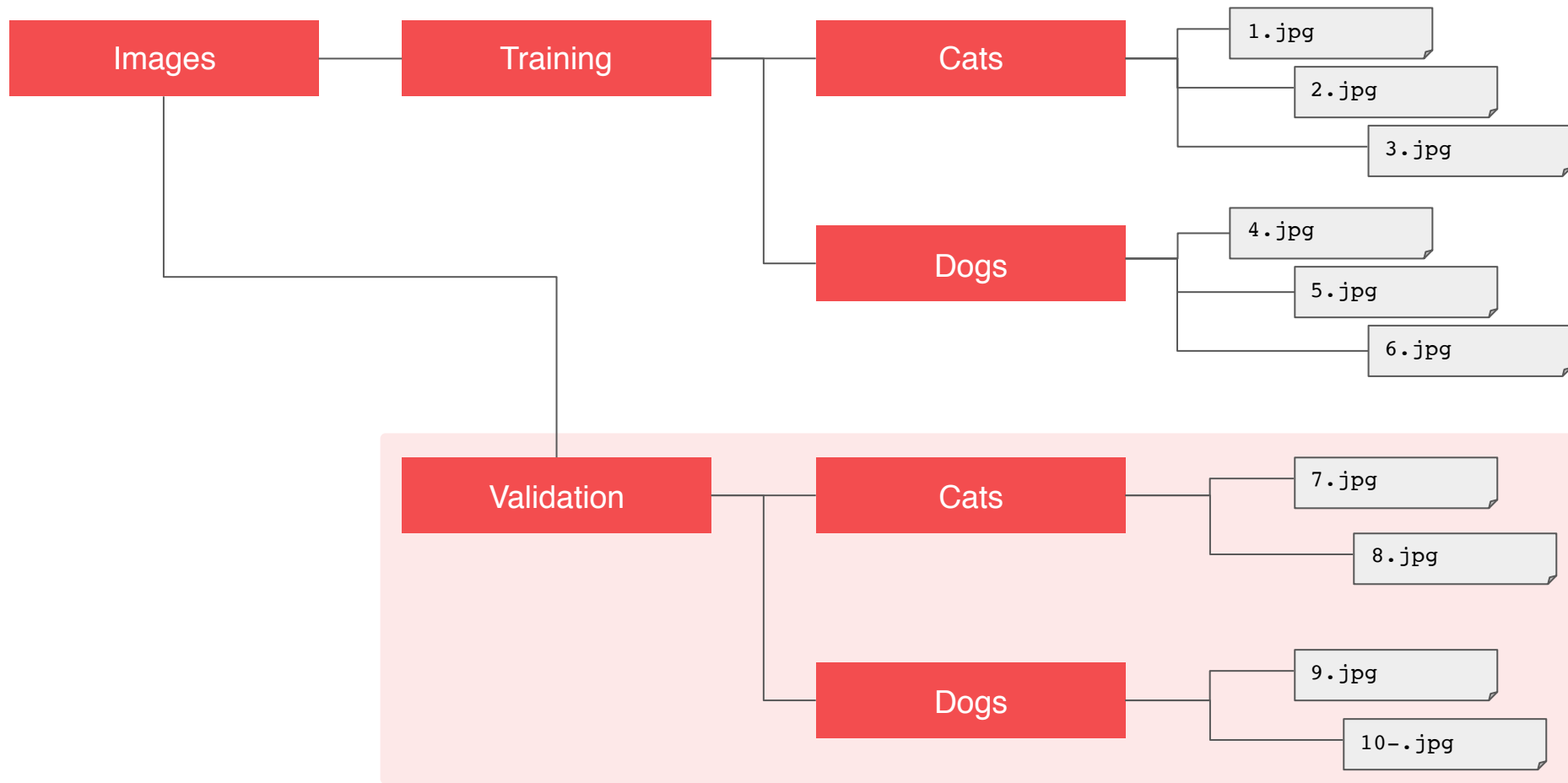
A note on hand labeling

Per the rules and spirit of this contest, please do not manually label your submissions. We work hard to fair and fun contests, and ask for the same respect in return.









```
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')
])
```



```
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 |
| <hr/> | | |
| max_pooling2d (MaxPooling2D) | (None, 74, 74, 16) | 0 |
| <hr/> | | |
| conv2d_1 (Conv2D) | (None, 72, 72, 32) | 4640 |
| <hr/> | | |
| max_pooling2d_1 (MaxPooling2D) | (None, 36, 36, 32) | 0 |
| <hr/> | | |
| conv2d_2 (Conv2D) | (None, 34, 34, 64) | 18496 |
| <hr/> | | |
| max_pooling2d_2 (MaxPooling2D) | (None, 17, 17, 64) | 0 |
| <hr/> | | |
| flatten (Flatten) | (None, 18496) | 0 |
| <hr/> | | |
| dense (Dense) | (None, 512) | 9470464 |
| <hr/> | | |
| 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(  
    train_generator,  
    steps_per_epoch=100,  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=50,  
    verbose=2)
```