Working with Images

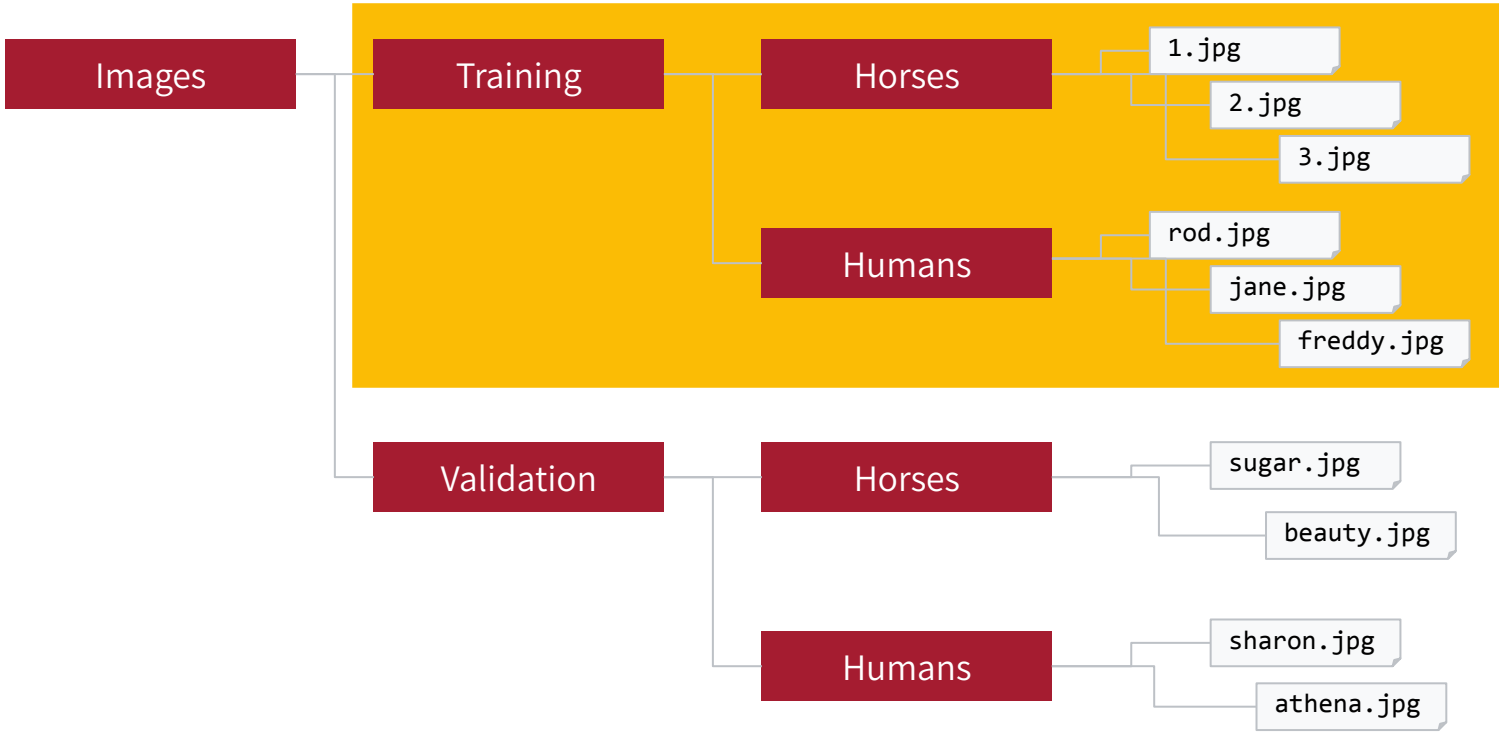
Training a CNN with complex images

Laurence Moroney, Google

```
mnist = tf.keras.datasets.fashion_mnist

(training_images, training_labels),
(val_images, val_labels) = mnist.load_data()

training_images=training_images / 255.0
val_images=val_images / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(20, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```







Images
Training
Horses
1.jpg
2.jpg
3.jpg
Humans
rod.jpg
jane.jpg
freddy.jpg
Validation
Horses
sugar.jpg
beauty.jpg
Humans
sharon.jpg
athena.jpg

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

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

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300, 300),
        batch_size=128,
        class_mode='binary')
```

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

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300, 300),
        batch_size=128,
        class_mode='binary')
```

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

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300, 300),
        batch_size=128,
        class_mode='binary')
```

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

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300, 300),
        batch_size=128,
        class_mode='binary')
```

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

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300, 300),
        batch_size=128,
        class_mode='binary')
```

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

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300, 300),
        batch_size=128,
        class_mode='binary')
```

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

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(300, 300),
        batch_size=32,
        class_mode='binary')
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu',
                           input_shape=(300, 300, 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.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=(300, 300, 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.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.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')
])
```

```
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=8,
      epochs=15,
      validation_data=validation_generator,
      validation_steps=8)
```

```
Epoch 10/15
8/8 [==============================] - 11s 1s/step - loss: 0.0486 - acc: 0.9878 - val_loss: 0.0252 - val_acc: 0.9893
Epoch 11/15
8/8 [==============================] - 11s 1s/step - loss: 0.0168 - acc: 0.9944 - val_loss: 0.0154 - val_acc: 0.9942
Epoch 12/15
8/8 [==============================] - 11s 1s/step - loss: 0.7284 - acc: 0.8721 - val_loss: 0.4953 - val_acc: 0.7692
Epoch 13/15
8/8 [==============================] - 11s 1s/step - loss: 0.1421 - acc: 0.9388 - val_loss: 0.0380 - val_acc: 0.9873
Epoch 14/15
8/8 [==============================] - 11s 1s/step - loss: 0.0219 - acc: 0.9944 - val_loss: 0.0120 - val_acc: 0.9990
Epoch 15/15
8/8 [==============================] - 12s 1s/step - loss: 0.0108 - acc: 0.9989 - val_loss: 0.0069 - val_acc: 1.0000
```

Your turn!