

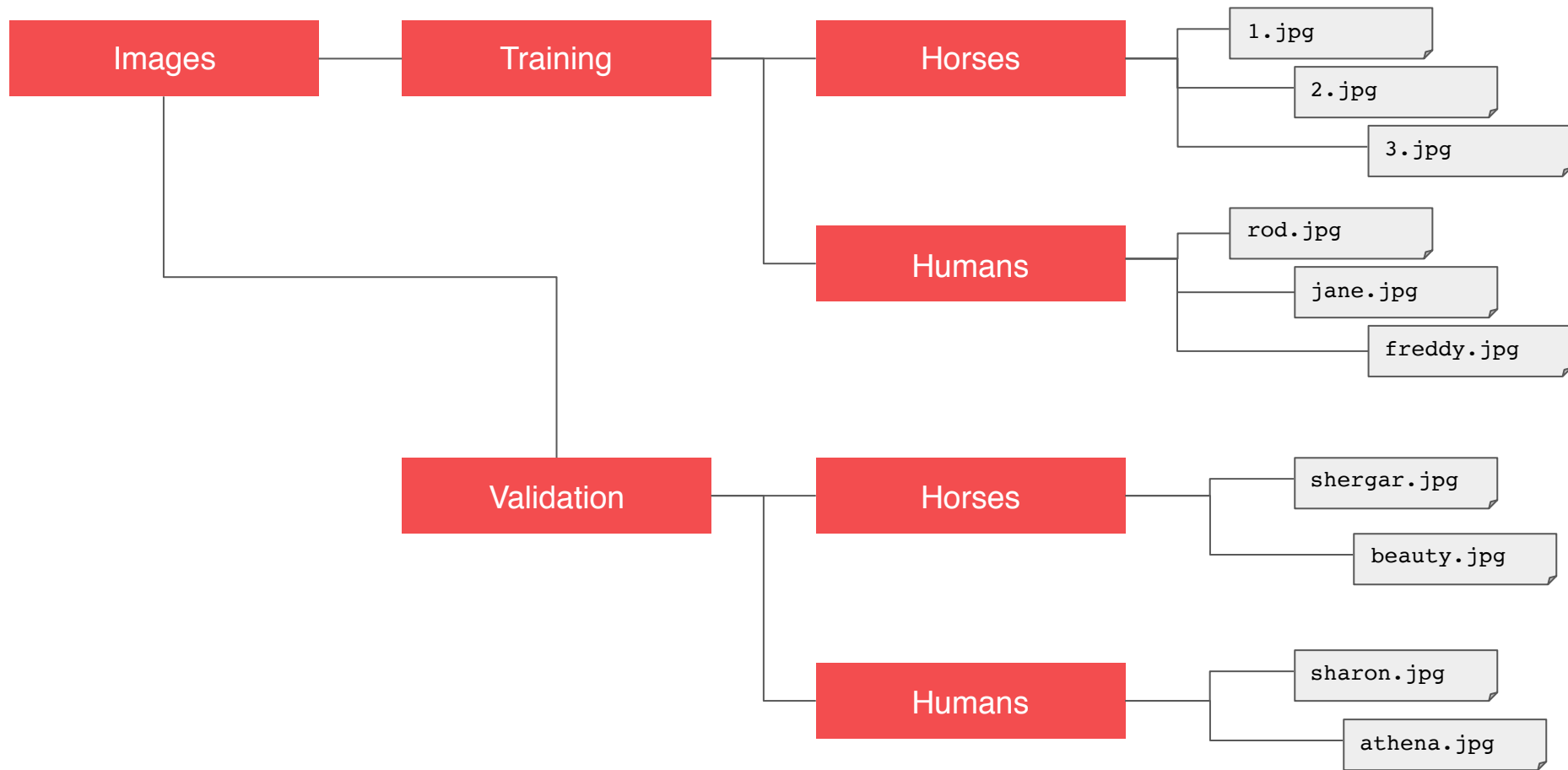
Copyright Notice

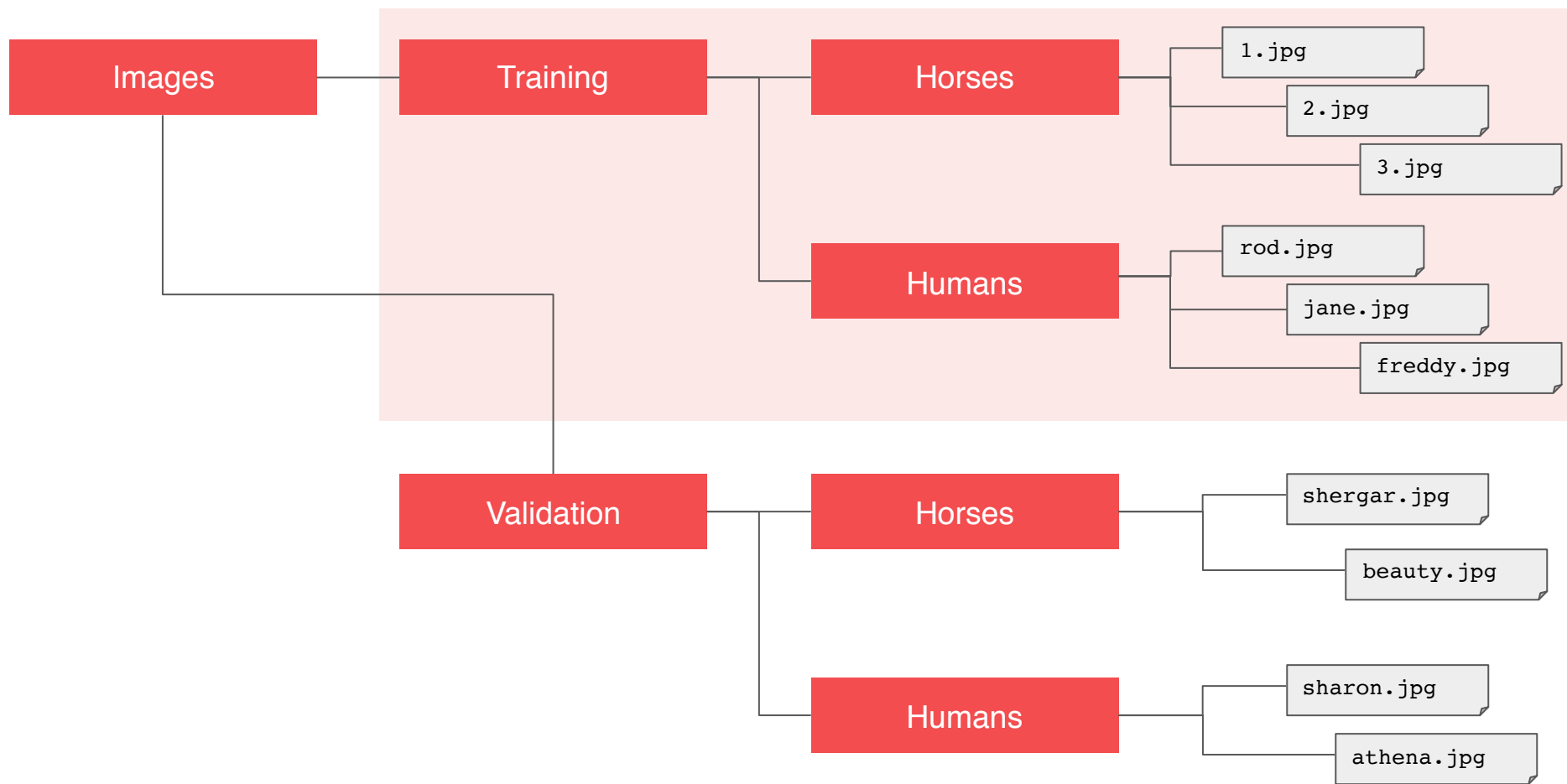
These slides are distributed under the Creative Commons License.

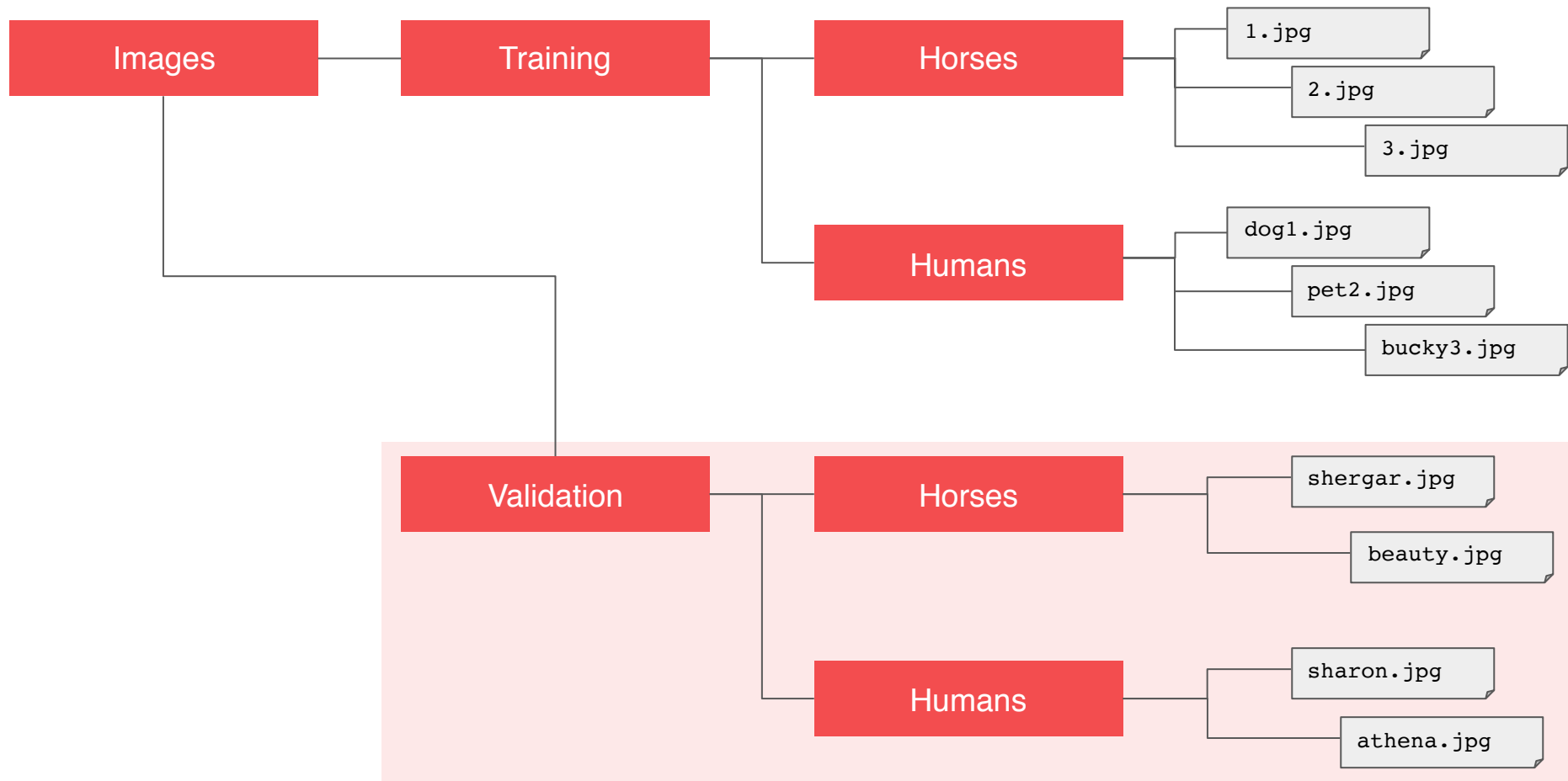
[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

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









```
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.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.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.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.Flatten(),  
    tf.keras.layers.Dense(512, activation='relu'),  
    tf.keras.layers.Dense(1, activation='sigmoid')  
])
```



Layer (type)	Output Shape	Param #
=====		
conv2d_5 (Conv2D)	(None, 298, 298, 16)	448
<hr/>		
max_pooling2d_5 (MaxPooling2)	(None, 149, 149, 16)	0
<hr/>		
conv2d_6 (Conv2D)	(None, 147, 147, 32)	4640
<hr/>		
max_pooling2d_6 (MaxPooling2)	(None, 73, 73, 32)	0
<hr/>		
conv2d_7 (Conv2D)	(None, 71, 71, 64)	18496
<hr/>		
max_pooling2d_7 (MaxPooling2)	(None, 35, 35, 64)	0
<hr/>		
flatten_1 (Flatten)	(None, 78400)	0
<hr/>		
dense_2 (Dense)	(None, 512)	40141312
<hr/>		
dense_3 (Dense)	(None, 1)	513
=====		
Total params: 40,165,409		
Trainable params: 40,165,409		
Non-trainable params: 0		

```
from tensorflow.keras.optimizers import RMSprop
```

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

<https://youtu.be/zLRB4oupj6g>



Machine Learning



0:06 / 8:58



2.1.4 Gradient Descent in Practice II Learning Rate by Andrew Ng

```
history = model.fit(  
    train_generator,  
    steps_per_epoch=8,  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=8,  
    verbose=2)
```



```
history = model.fit(  
    train_generator,  
    steps_per_epoch=8,  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=8,  
    verbose=2)
```

```
history = model.fit(  
    train_generator,  
    steps_per_epoch=8,  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=8,  
    verbose=2)
```

```
history = model.fit(  
    train_generator,  
    steps_per_epoch=8,  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=8,  
    verbose=2)
```



```
history = model.fit(  
    train_generator,  
    steps_per_epoch=8,  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=8,  
    verbose=2)
```

```
history = model.fit(  
    train_generator,  
    steps_per_epoch=8,  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=8,  
    verbose=2)
```

```
history = model.fit(  
    train_generator,  
    steps_per_epoch=8,  
    epochs=15,  
    validation_data=validation_generator,  
    validation_steps=8,  
    verbose=2)
```

```
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=(300, 300))
```

```
    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")
```

```
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=(300, 300))
```

```
    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")
```

```
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=(300, 300))
```

```
    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")
```

```
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=(300, 300))
```

```
    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")
```