

# 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/>

The screenshot shows the Kaggle interface for the "Dogs vs. Cats" competition. At the top, there's a navigation bar with links for Search, Competitions, Datasets, Kernels, Discussion, Learn, and more. A user profile icon is also present. Below the header, the competition title "Dogs vs. Cats" is displayed next to images of a dog and a cat. A subtitle reads "Create an algorithm to distinguish dogs from cats", followed by "215 teams · 5 years ago". A horizontal menu bar below the title includes links for Overview, Data (which is underlined), Kernels, Discussion, Leaderboard, Rules, and Team. The main content area is titled "Data Description" and contains text about the training archive: "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)." Below this, a section titled "A note on hand labeling" advises users not to manually label submissions. A small image of a black and tan dog is shown at the bottom left.

Dogs vs. Cats

Create an algorithm to distinguish dogs from cats

215 teams · 5 years ago

Overview Data Kernels Discussion Leaderboard Rules Team

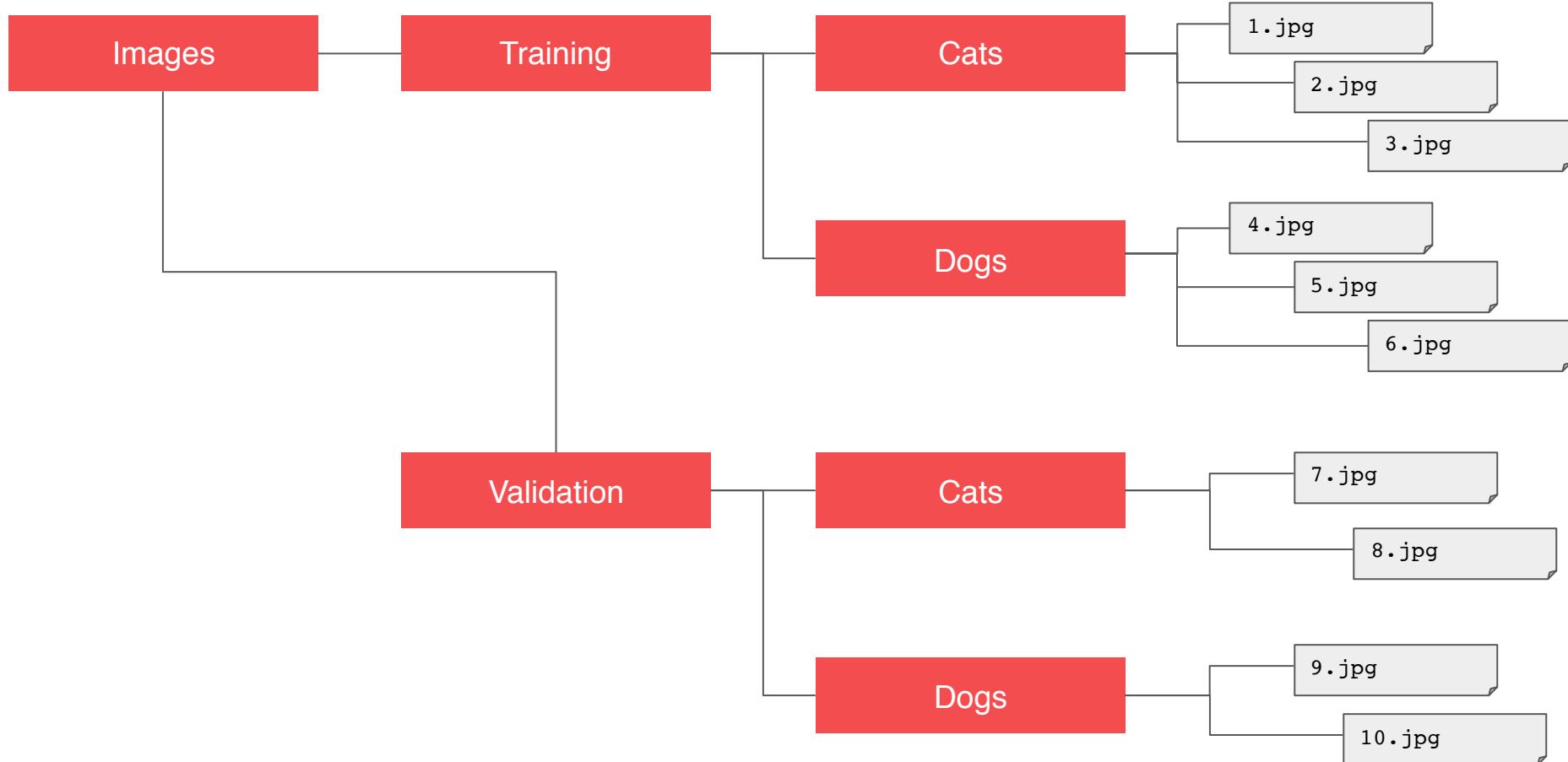
Data Description

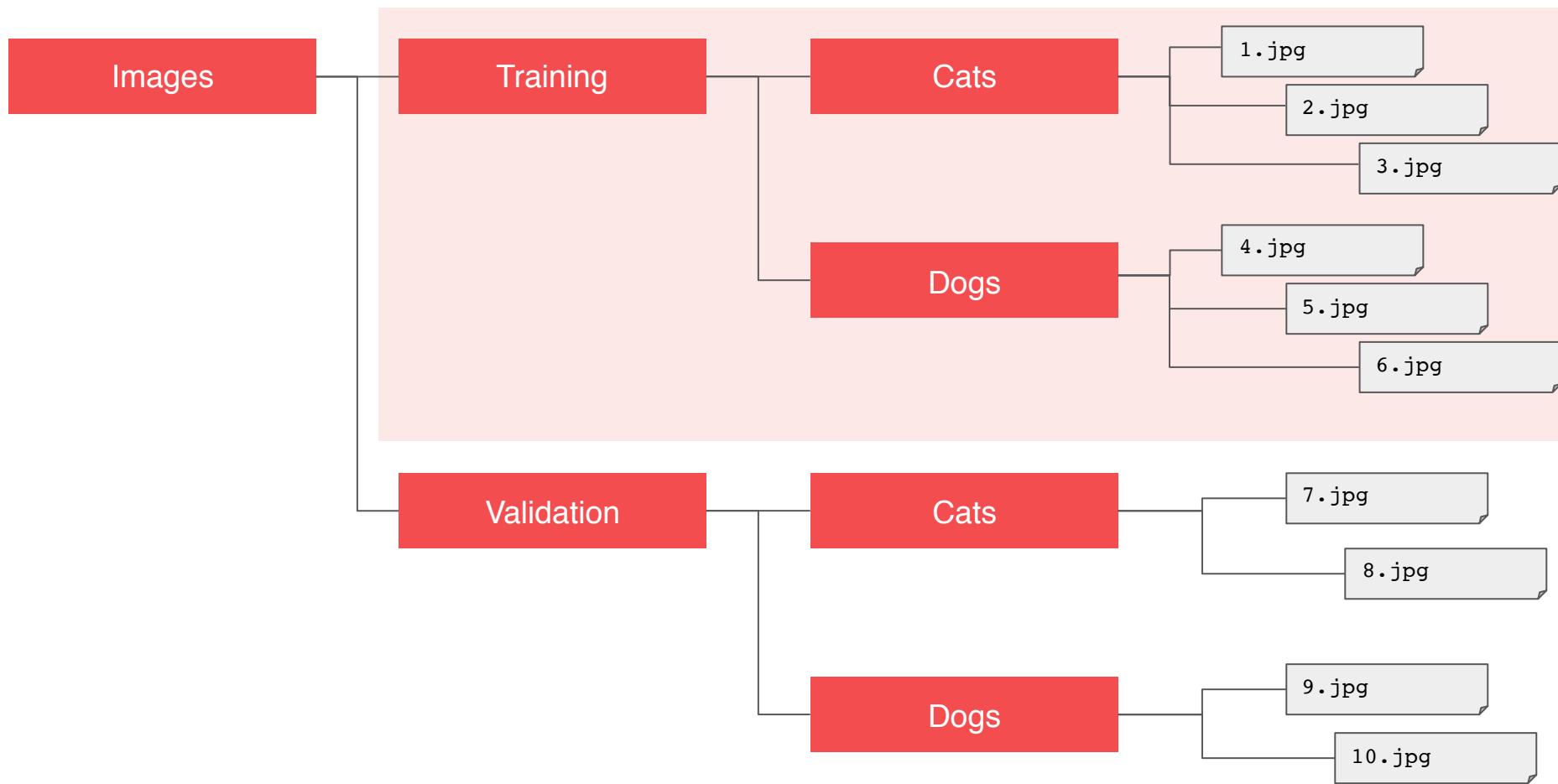
The training archive contains 25,000 images of dogs and cats. Train your algorithm on these files and predict the labels for test1.zip (1 = dog, 0 = cat).

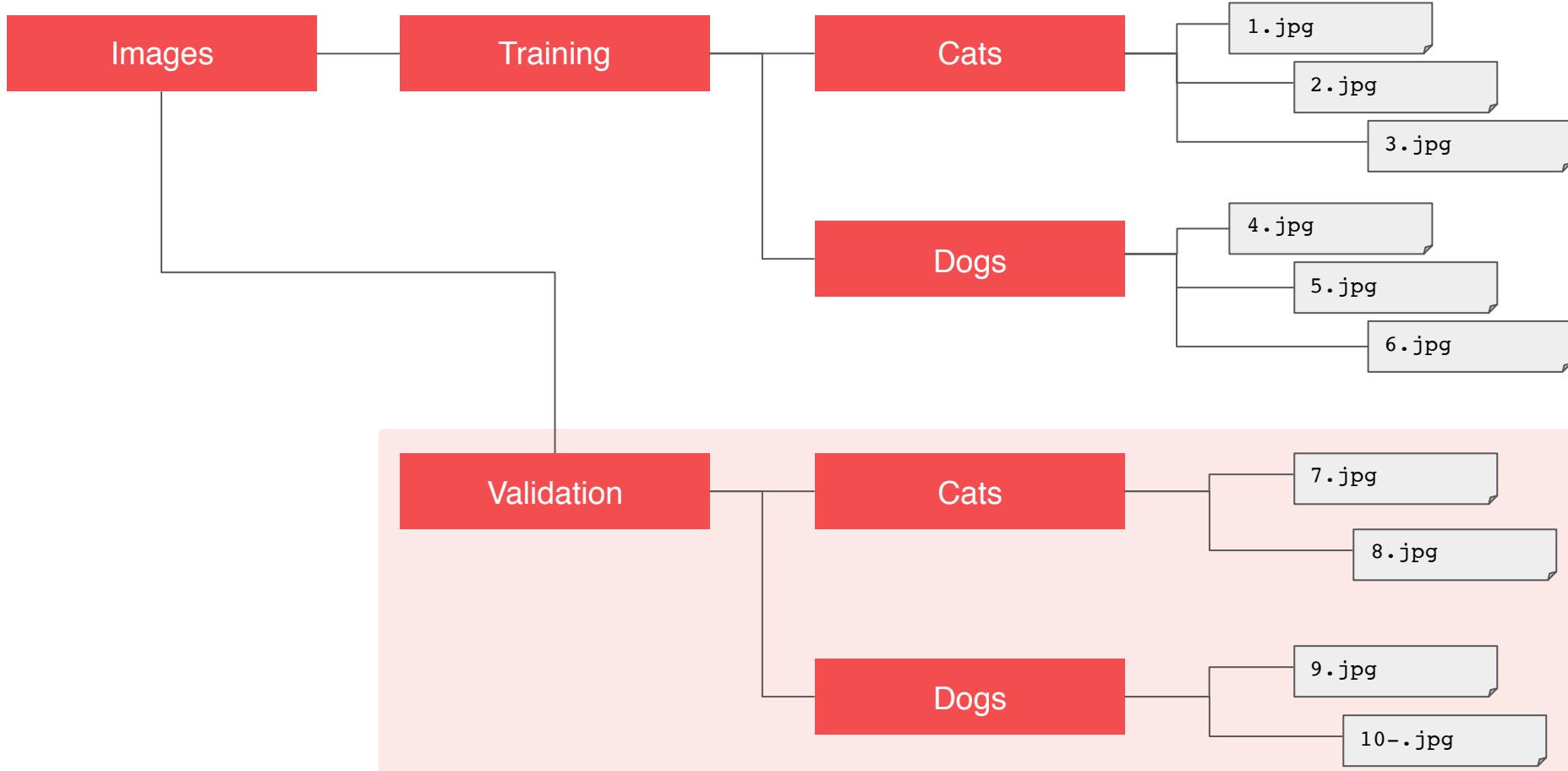
**A note on hand labeling**

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.

A close-up photograph of a black and tan dog's face, looking slightly to the side. The dog has dark eyes and a brown patch on its forehead.







```
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
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'])
```

```
history = model.fit(
```

```
    train_generator,
```

```
    steps_per_epoch=100,
```

```
    epochs=15,
```

```
    validation_data=validation_generator,
```

```
    validation_steps=50,
```

```
    verbose=2)
```

# Copyright Notice

These slides are distributed under the Creative Commons License.

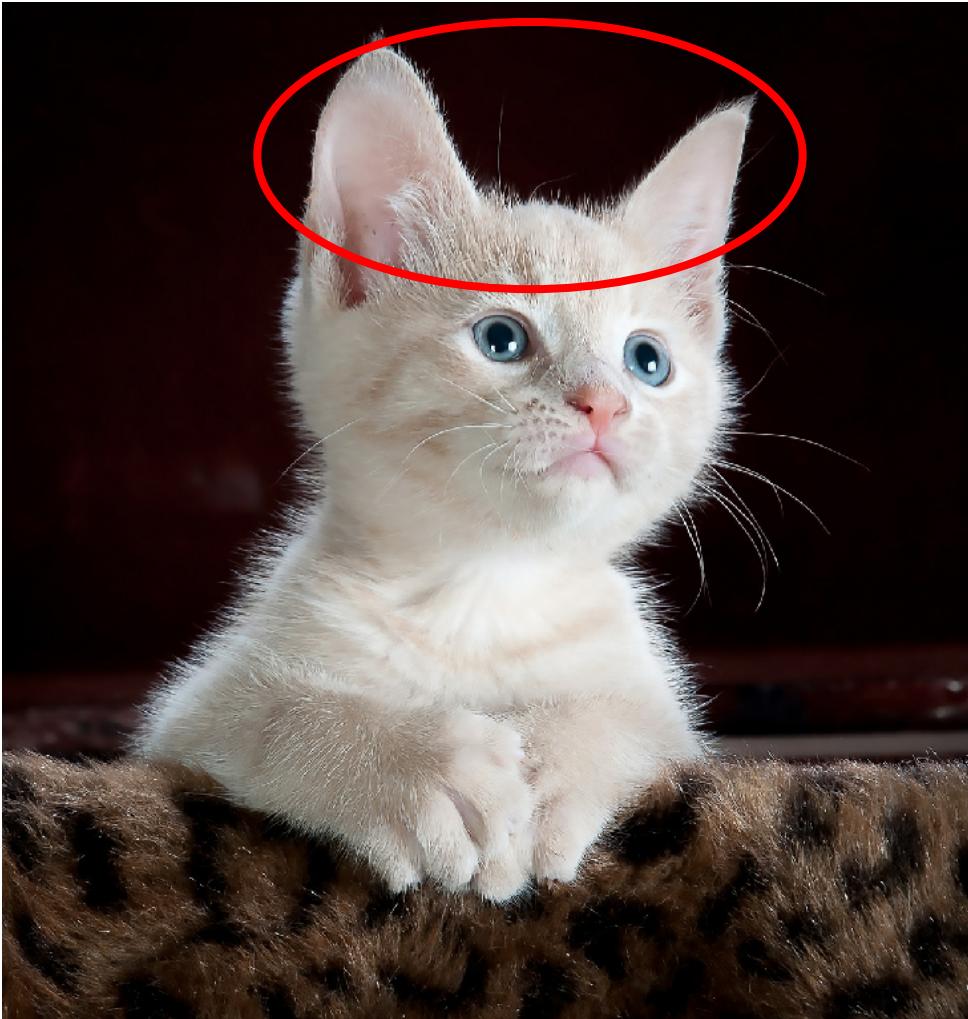
DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

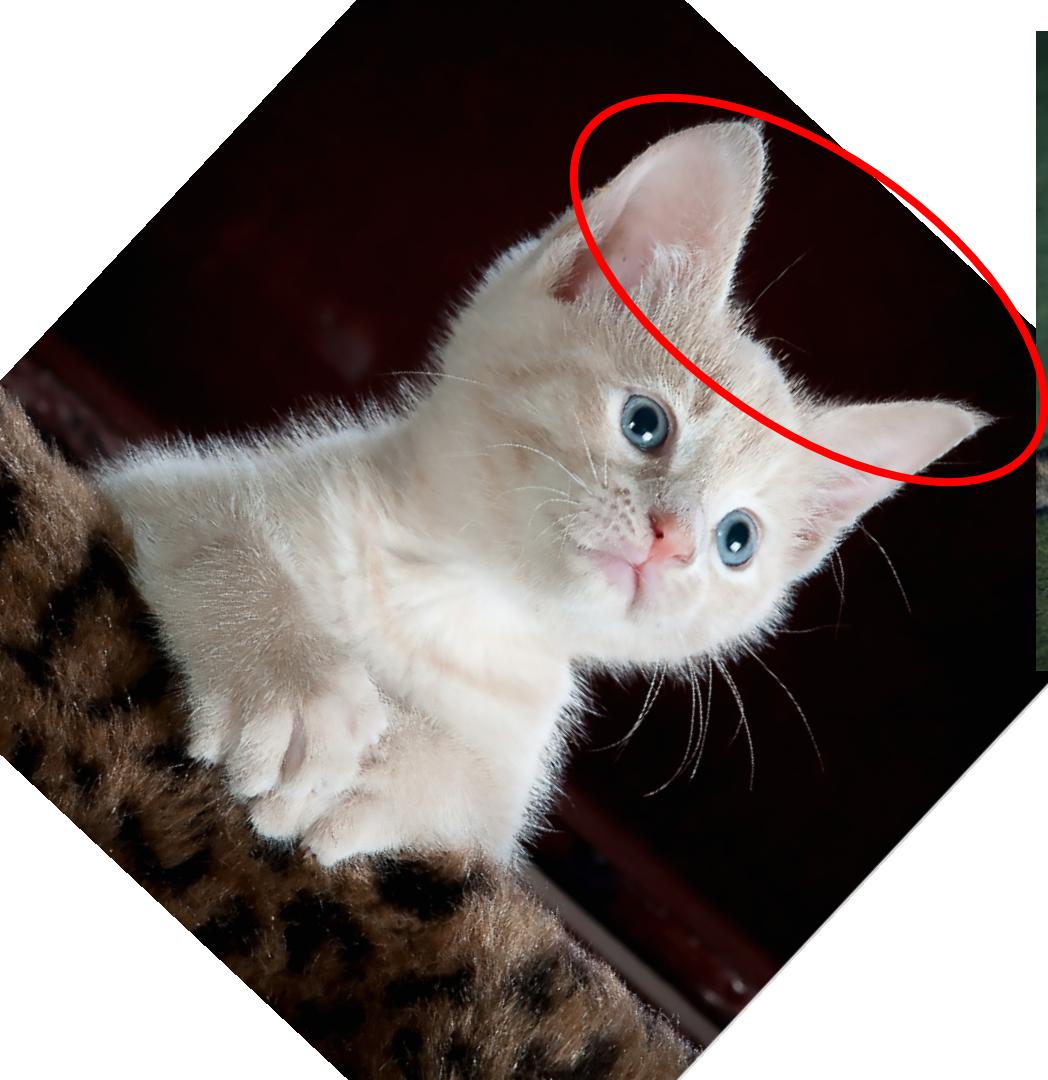
For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>











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

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

```
# Updated to do image augmentation
```

```
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
```

```
    rotation_range=40,
```

```
    width_shift_range=0.2,
```

```
    height_shift_range=0.2,
```

```
    shear_range=0.2,
```

```
    zoom_range=0.2,
```

```
    horizontal_flip=True,
```

```
    fill_mode='nearest')
```

```
# Updated to do image augmentation
```

```
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
```

```
    rotation_range=40,
```

```
    width_shift_range=0.2,
```

```
    height_shift_range=0.2,
```

```
    shear_range=0.2,
```

```
    zoom_range=0.2,
```

```
    horizontal_flip=True,
```

```
    fill_mode='nearest')
```

```
# Updated to do image augmentation
```

```
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
```

```
    rotation_range=40,
```

```
    width_shift_range=0.2,
```

```
    height_shift_range=0.2,
```

```
    shear_range=0.2,
```

```
    zoom_range=0.2,
```

```
    horizontal_flip=True,
```

```
    fill_mode='nearest')
```





```
# Updated to do image augmentation
```

```
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
```

```
    rotation_range=40,
```

```
    width_shift_range=0.2,
```

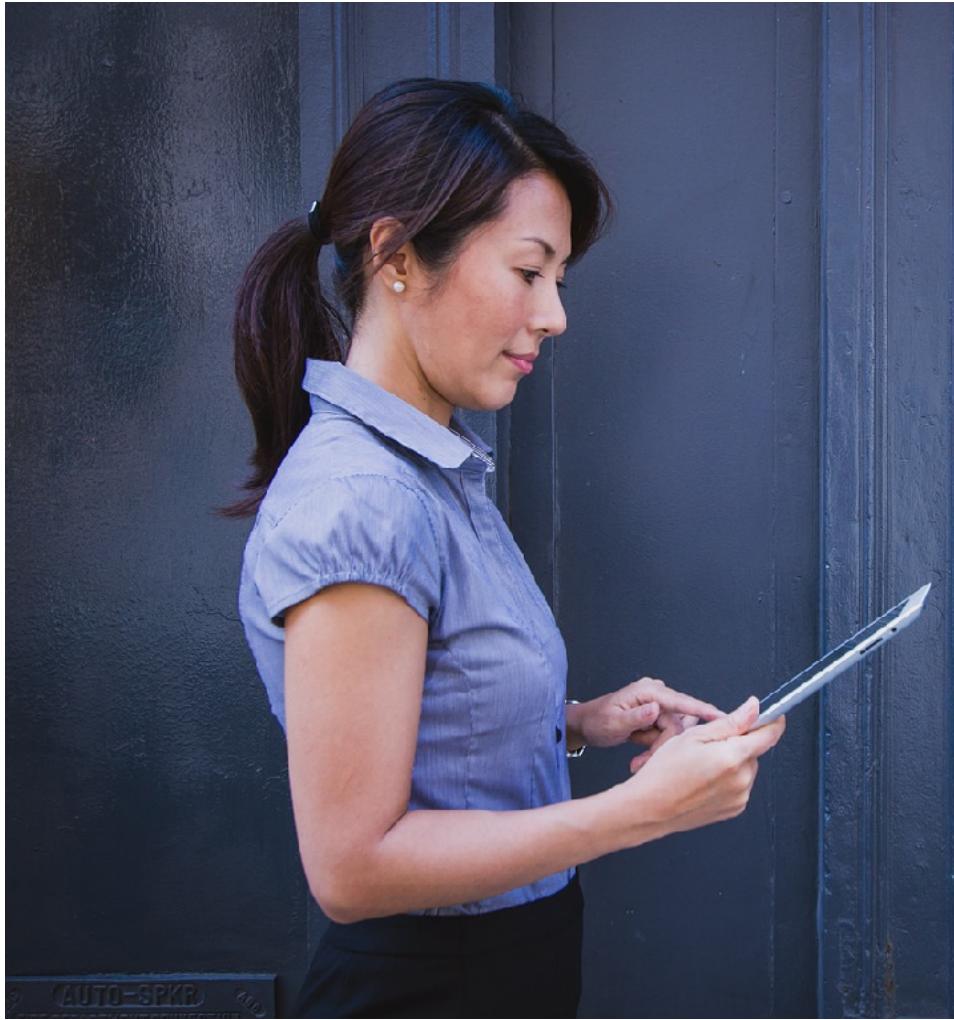
```
    height_shift_range=0.2,
```

```
    shear_range=0.2,
```

```
    zoom_range=0.2,
```

```
    horizontal_flip=True,
```

```
    fill_mode='nearest')
```





```
# Updated to do image augmentation
```

```
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
```

```
    rotation_range=40,
```

```
    width_shift_range=0.2,
```

```
    height_shift_range=0.2,
```

```
    shear_range=0.2,
```

```
    zoom_range=0.2,
```

```
    horizontal_flip=True,
```

```
    fill_mode='nearest')
```





```
# Updated to do image augmentation
```

```
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
```

```
    rotation_range=40,
```

```
    width_shift_range=0.2,
```

```
    height_shift_range=0.2,
```

```
    shear_range=0.2,
```

```
    zoom_range=0.2,
```

```
    horizontal_flip=True,
```

```
    fill_mode='nearest')
```

```
# Updated to do image augmentation
```

```
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
```

```
    rotation_range=40,
```

```
    width_shift_range=0.2,
```

```
    height_shift_range=0.2,
```

```
    shear_range=0.2,
```

```
    zoom_range=0.2,
```

```
    horizontal_flip=True,
```

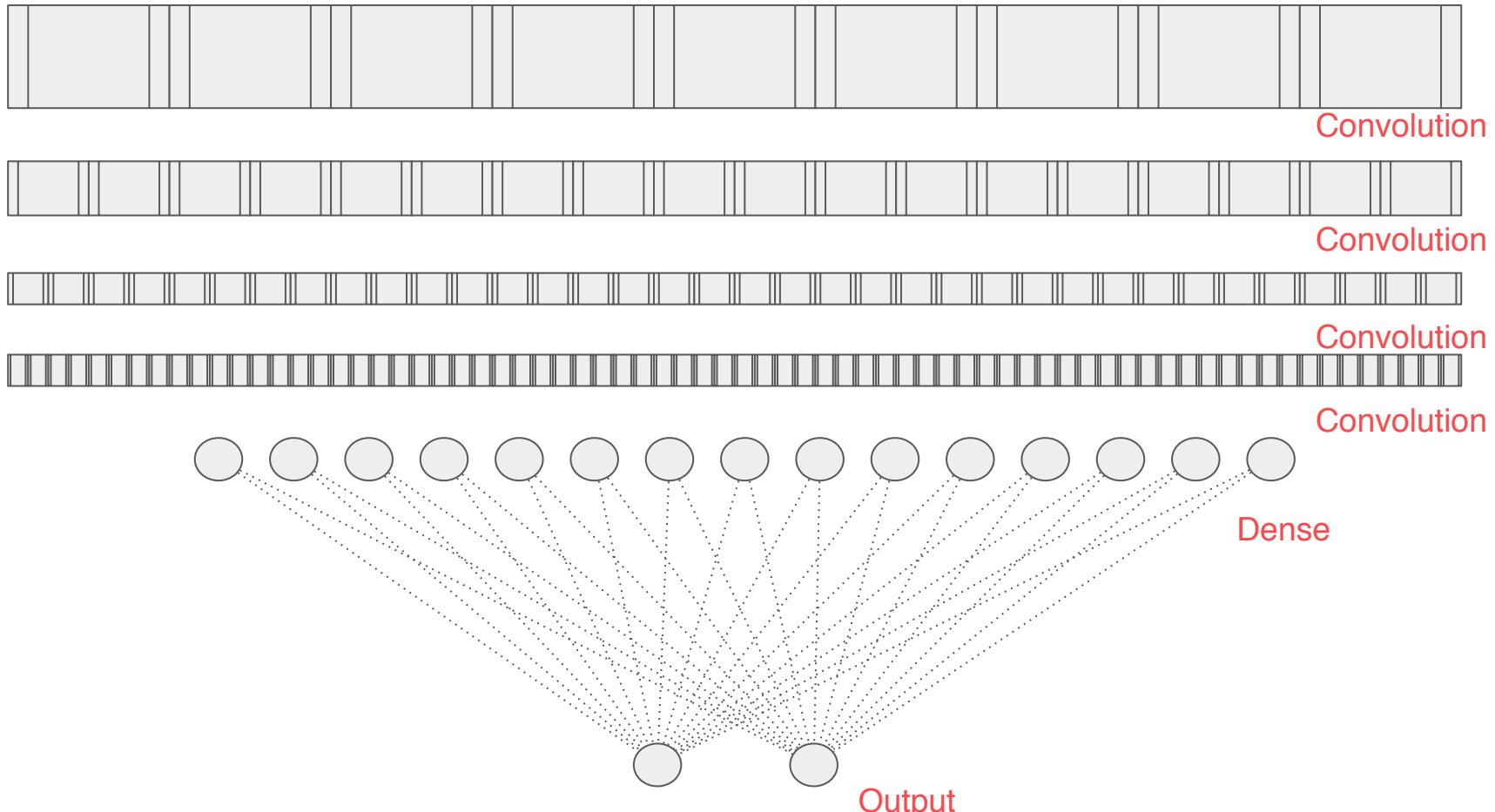
```
    fill_mode='nearest')
```

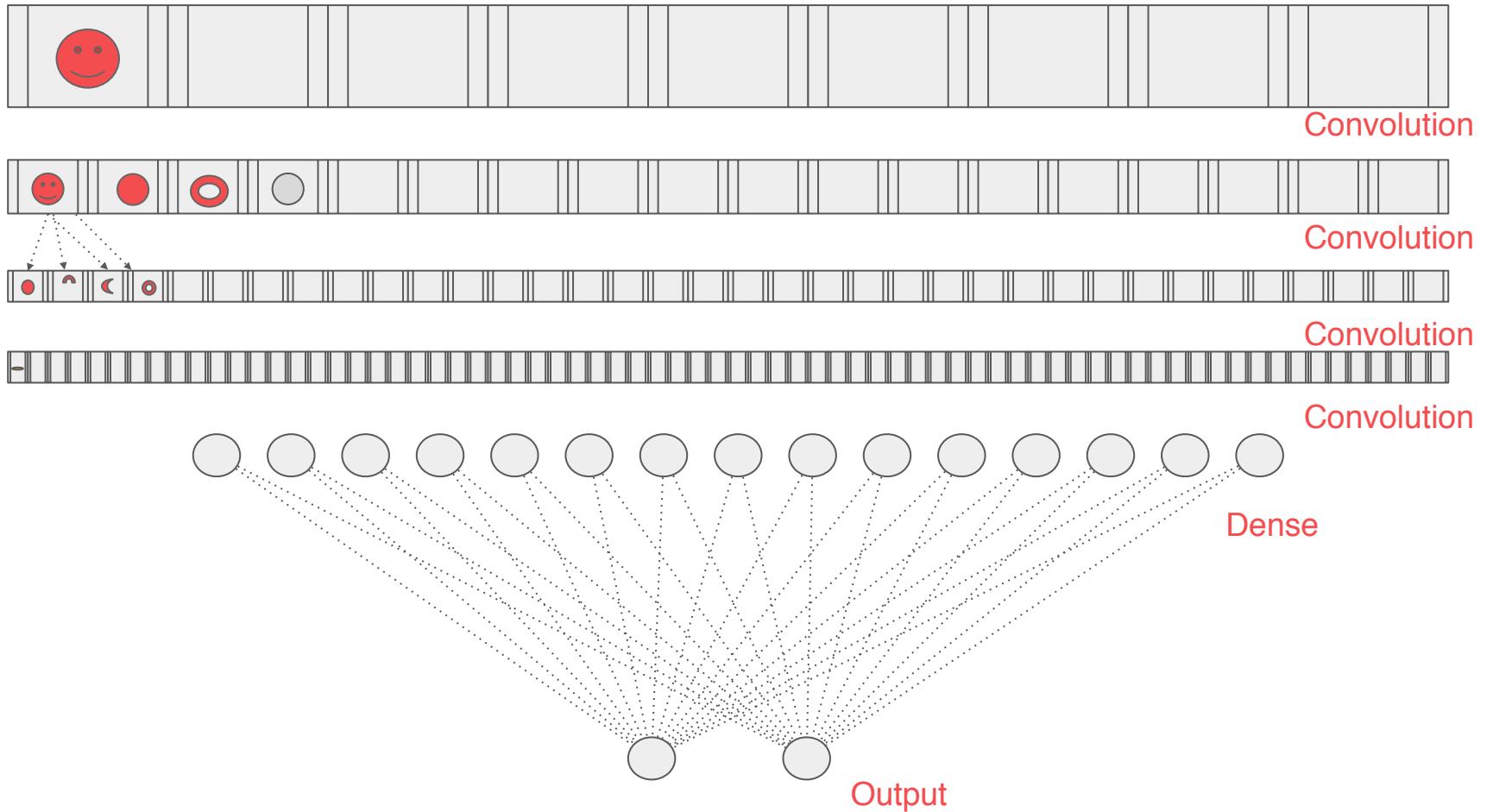
# Copyright Notice

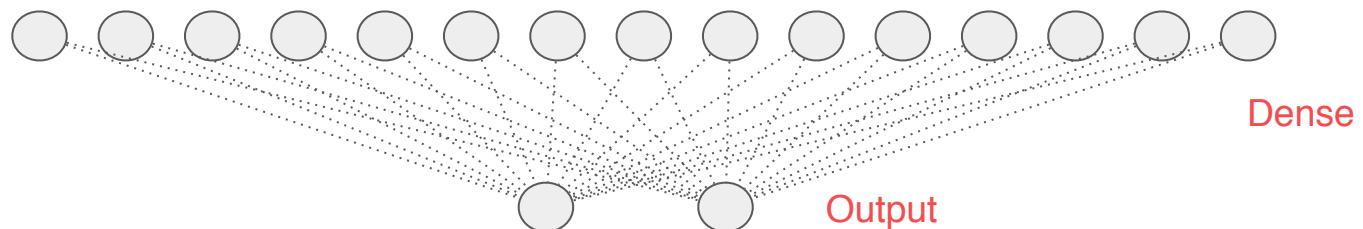
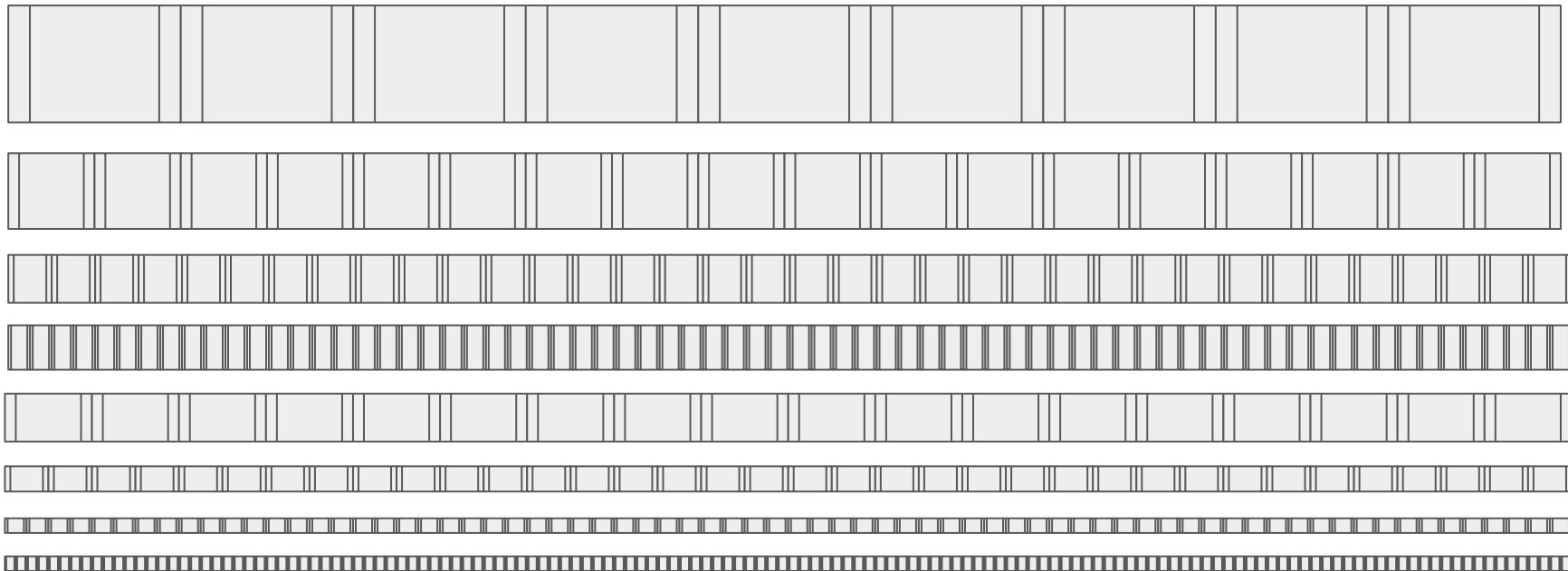
These slides are distributed under the Creative Commons License.

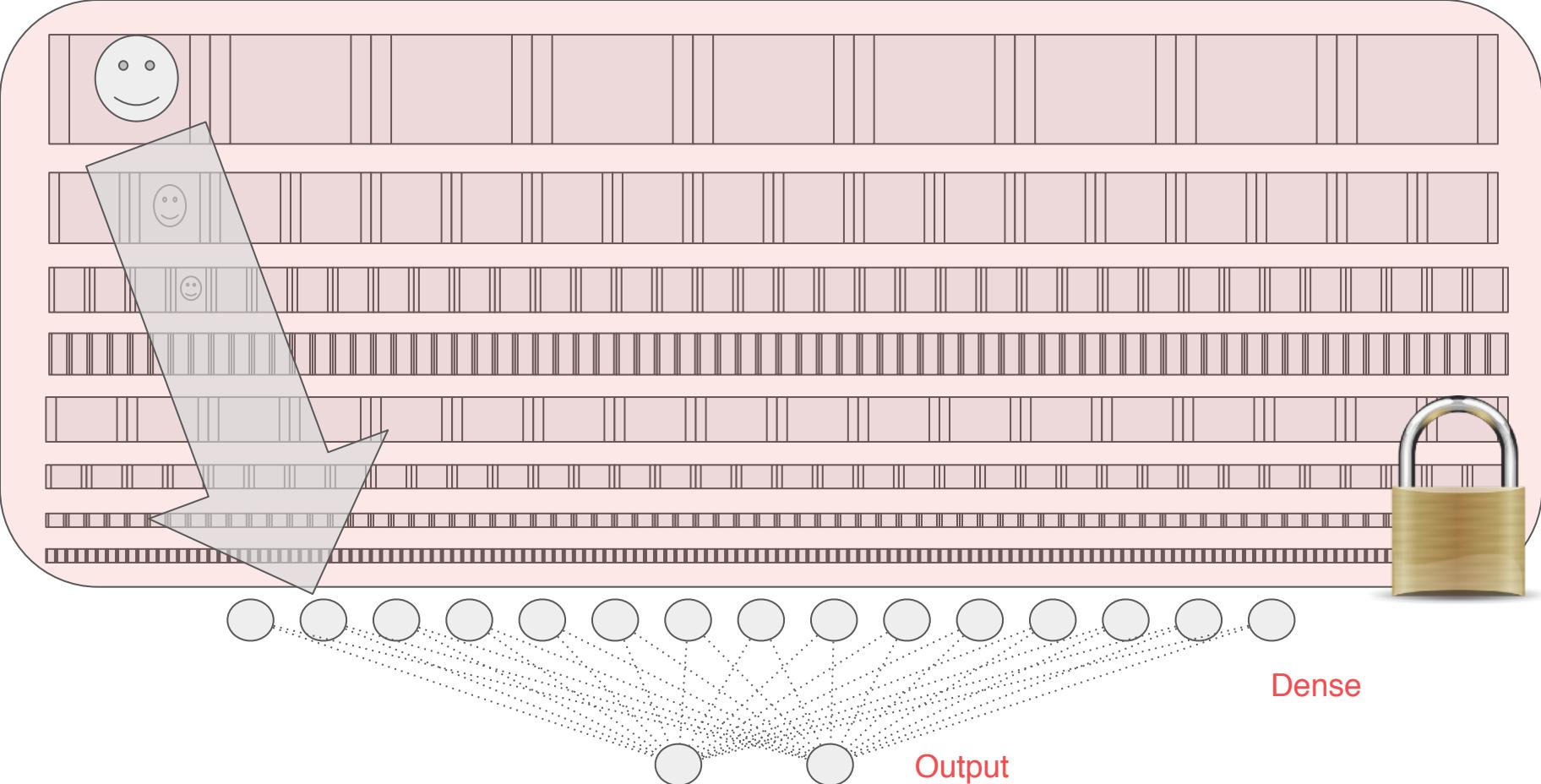
DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

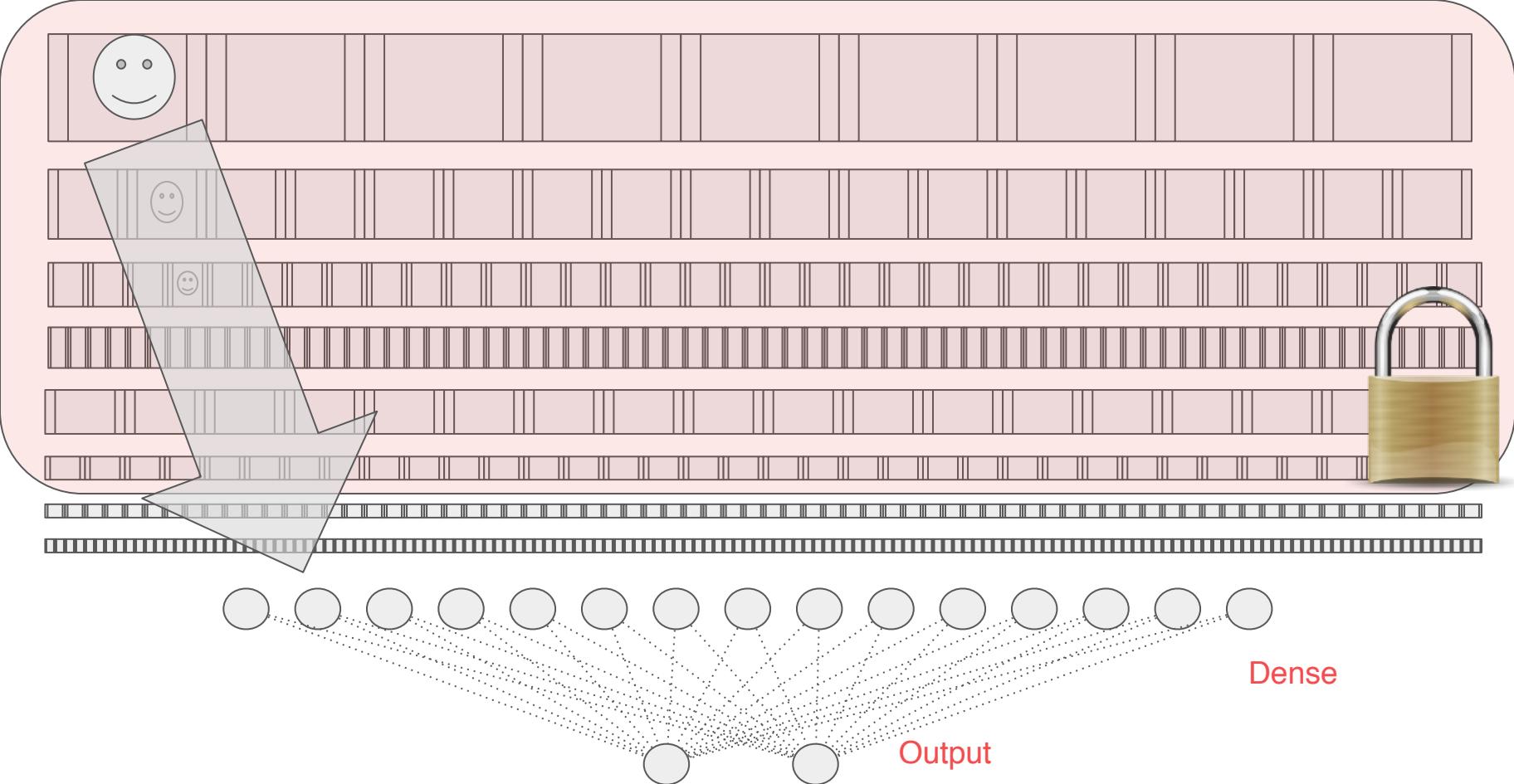
For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>













## Rethinking the Inception Architecture for Computer Vision

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna

(Submitted on 2 Dec 2015 ([v1](#)), last revised 11 Dec 2015 (this version, v3))

Convolutional networks are at the core of most state-of-the-art computer vision solutions for a wide variety of tasks. Since 2014 very deep convolutional networks started to become mainstream, yielding substantial gains in various benchmarks. Although increased model size and computational cost tend to translate to immediate quality gains for most tasks (as long as enough labeled data is provided for training), computational efficiency and low parameter count are still enabling factors for various use cases such as mobile vision and big-data scenarios. Here we explore ways to scale up networks in ways that aim at utilizing the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization. We benchmark our methods on the ILSVRC 2012 classification challenge validation set demonstrate substantial gains over the state of the art: 21.2% top-1 and 5.6% top-5 error for single frame evaluation using a network with a computational cost of 5 billion multiply-adds per inference and with using less than 25 million parameters. With an ensemble of 4 models and multi-crop evaluation, we report 3.5% top-5 error on the validation set (3.6% error on the test set) and 17.3% top-1 error on the validation set.

### Download:

- PDF
- Other formats

(license)

Current browse context:

cs.CV

[< prev](#) | [next >](#)  
[new](#) | [recent](#) | [1512](#)

Change to browse by:

[cs](#)

References & Citations

- [NASA ADS](#)

<http://image-net.org/>



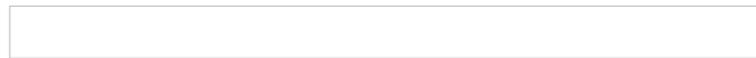
14,197,122 images, 21841 synsets indexed

[Explore](#) [Download](#) [Challenges](#) [Publications](#) [CoolStuff](#) [About](#)

Not logged in. [Login](#) | [Signup](#)

**ImageNet** is an image database organized according to the [WordNet](#) hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.

[Click here](#) to learn more about ImageNet, [Click here](#) to join the ImageNet mailing list.



What do these images have in common? *Find out!*

[Check out the ImageNet Challenge on Kaggle!](#)

```
import os  
  
from tensorflow.keras import layers  
from tensorflow.keras import Model
```

[https://storage.googleapis.com/mledu-datasets/  
inception\\_v3\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels](https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_kernels)

```
from tensorflow.keras.applications.inception_v3 import InceptionV3

local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                 include_top = False,
                                 weights = None)

pre_trained_model.load_weights(local_weights_file)
```

```
for layer in pre_trained_model.layers:  
    layer.trainable = False
```

```
pre_trained_model.summary()
```

```
last_layer = pre_trained_model.get_layer('mixed7')
```

```
last_output = last_layer.output
```

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

```
x = layers.Flatten()(last_output)
```

```
x = layers.Dense(1024, activation='relu')(x)
```

```
x = layers.Dense (1, activation='sigmoid')(x)
```

```
model = Model( pre_trained_model.input, x)
```

```
model.compile(optimizer = RMSprop(learning_rate=0.0001),
```

```
loss = 'binary_crossentropy',
```

```
metrics = ['acc'])
```

```
from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dense (1, activation='sigmoid')(x)

model = Model( pre_trained_model.input, x)
model.compile(optimizer = RMSprop(learning_rate=0.0001),
               loss = 'binary_crossentropy',
               metrics = ['acc'])
```

```
from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dense (1, activation='sigmoid')(x)

model = Model( pre_trained_model.input, x)
model.compile(optimizer = RMSprop(learning_rate=0.0001),
               loss = 'binary_crossentropy',
               metrics = ['acc'])
```

```
from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dense (1, activation='sigmoid')(x)

model = Model( pre_trained_model.input, x)

model.compile(optimizer = RMSprop(learning_rate=0.0001),
              loss = 'binary_crossentropy',
              metrics = ['acc'])
```

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

```
x = layers.Flatten()(last_output)
```

```
x = layers.Dense(1024, activation='relu')(x)
```

```
x = layers.Dense (1, activation='sigmoid')(x)
```

```
model = Model( pre_trained_model.input, x)
```

```
model.compile(optimizer = RMSprop(lr=0.0001),
```

```
    loss = 'binary_crossentropy',
```

```
    metrics = ['acc'])
```

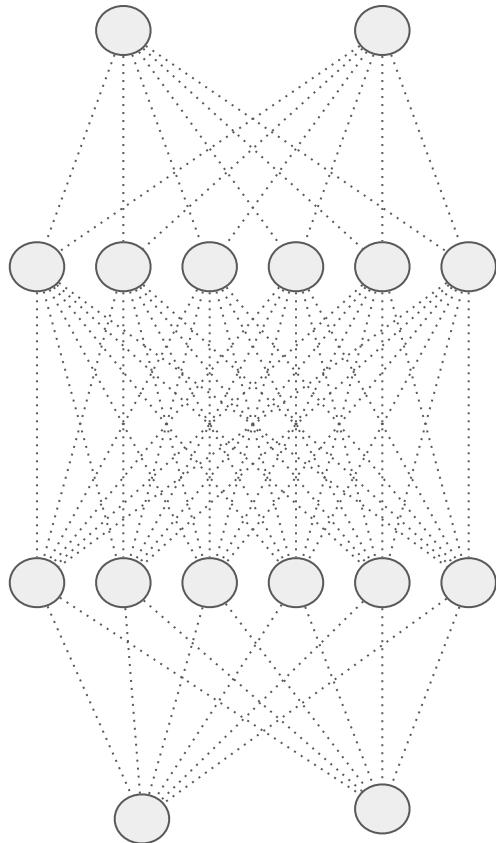


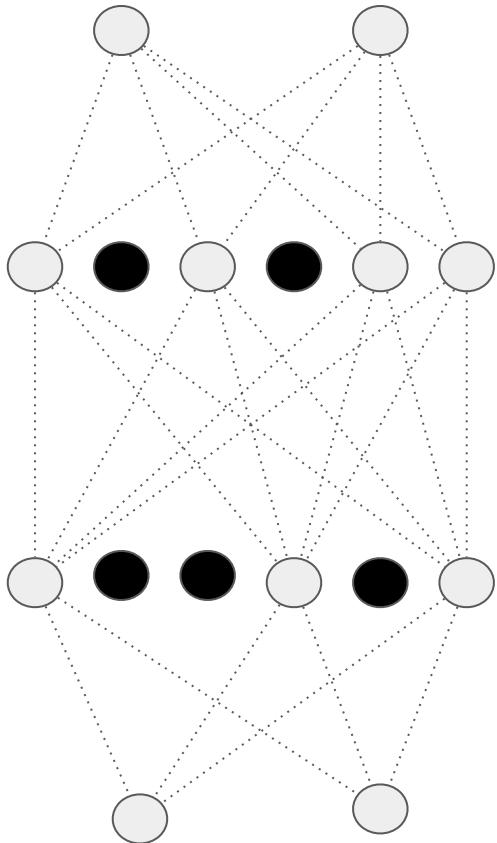
```
train_generator = train_datagen.flow_from_directory(  
    train_dir,  
    batch_size = 20,  
    class_mode = 'binary',  
    target_size = (150, 150))
```

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

### Training and validation accuracy,







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

```
x = layers.Flatten()(last_output)
```

```
x = layers.Dense(1024, activation='relu')(x)
```

```
x = layers.Dense (1, activation='sigmoid')(x)
```

```
model = Model( pre_trained_model.input, x)
```

```
model.compile(optimizer = RMSprop(lr=0.0001),
```

```
    loss = 'binary_crossentropy',
```

```
    metrics = ['acc'])
```

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

```
x = layers.Flatten()(last_output)
```

```
x = layers.Dense(1024, activation='relu')(x)
```

```
x = layers.Dropout(0.2)(x)
```

```
x = layers.Dense (1, activation='sigmoid')(x)
```

```
model = Model( pre_trained_model.input, x)
```

```
model.compile(optimizer = RMSprop(lr=0.0001),
```

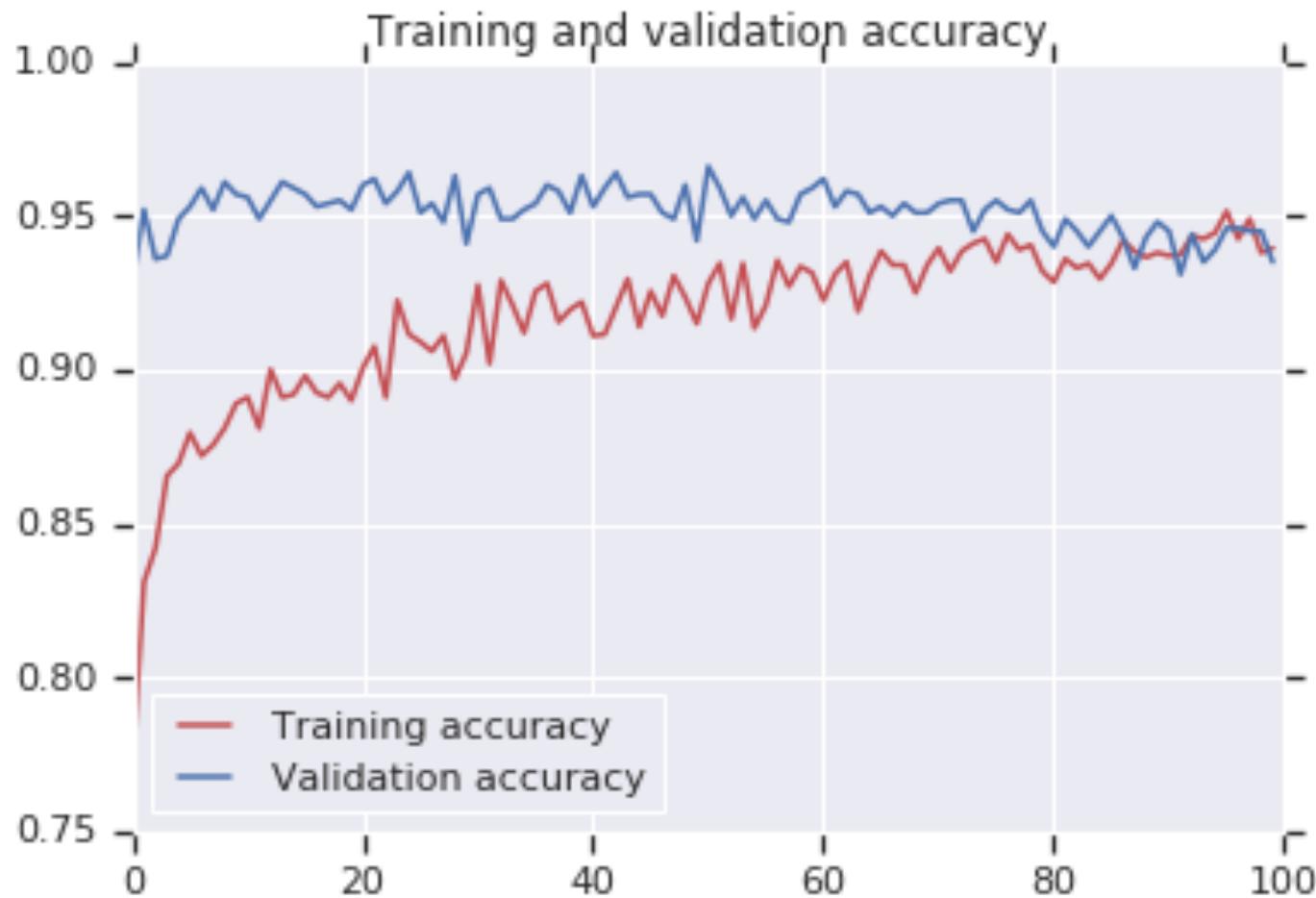
```
    loss = 'binary_crossentropy',
```

```
    metrics = ['acc'])
```

### Training and validation accuracy,



Training and validation accuracy

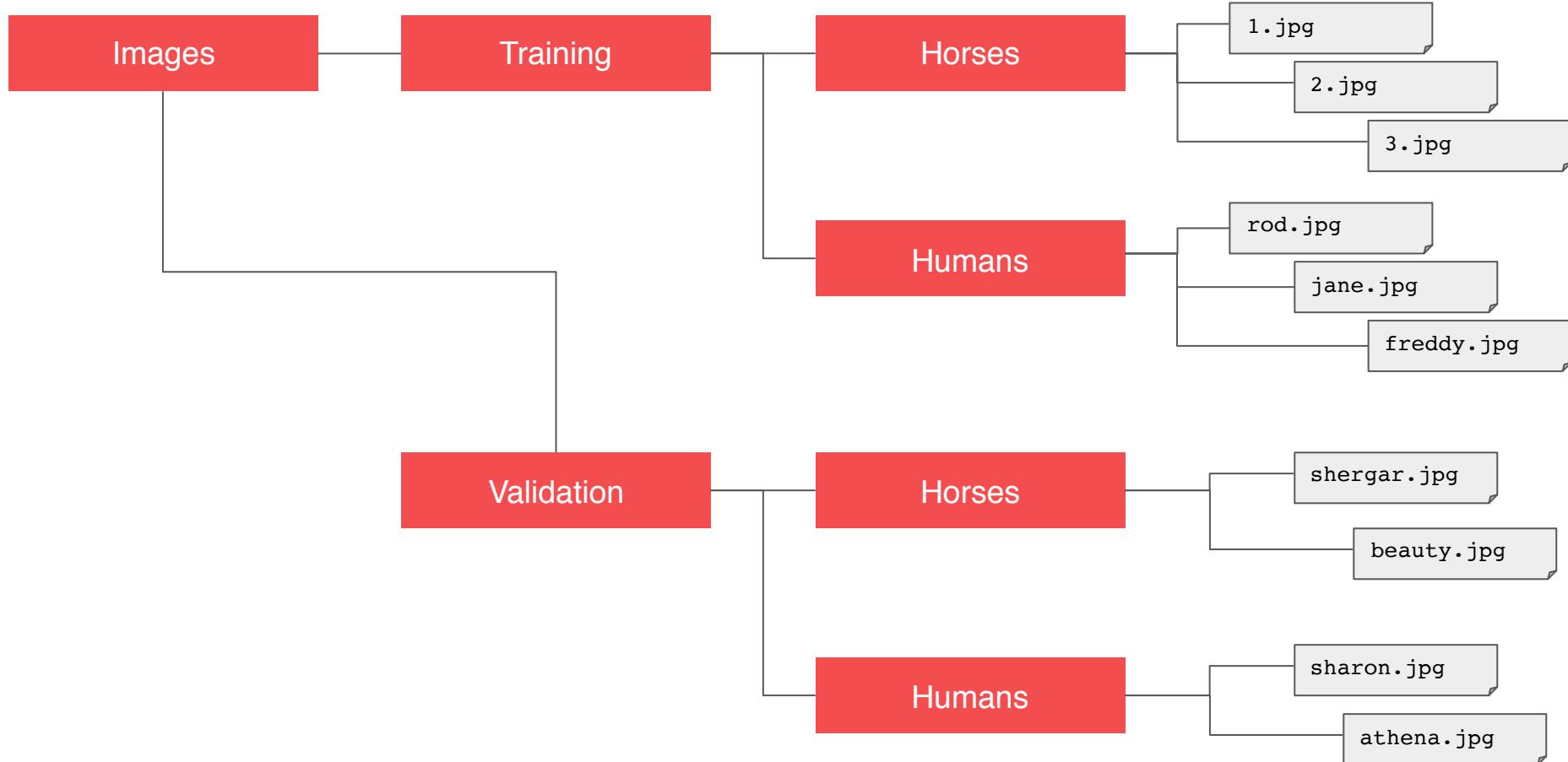


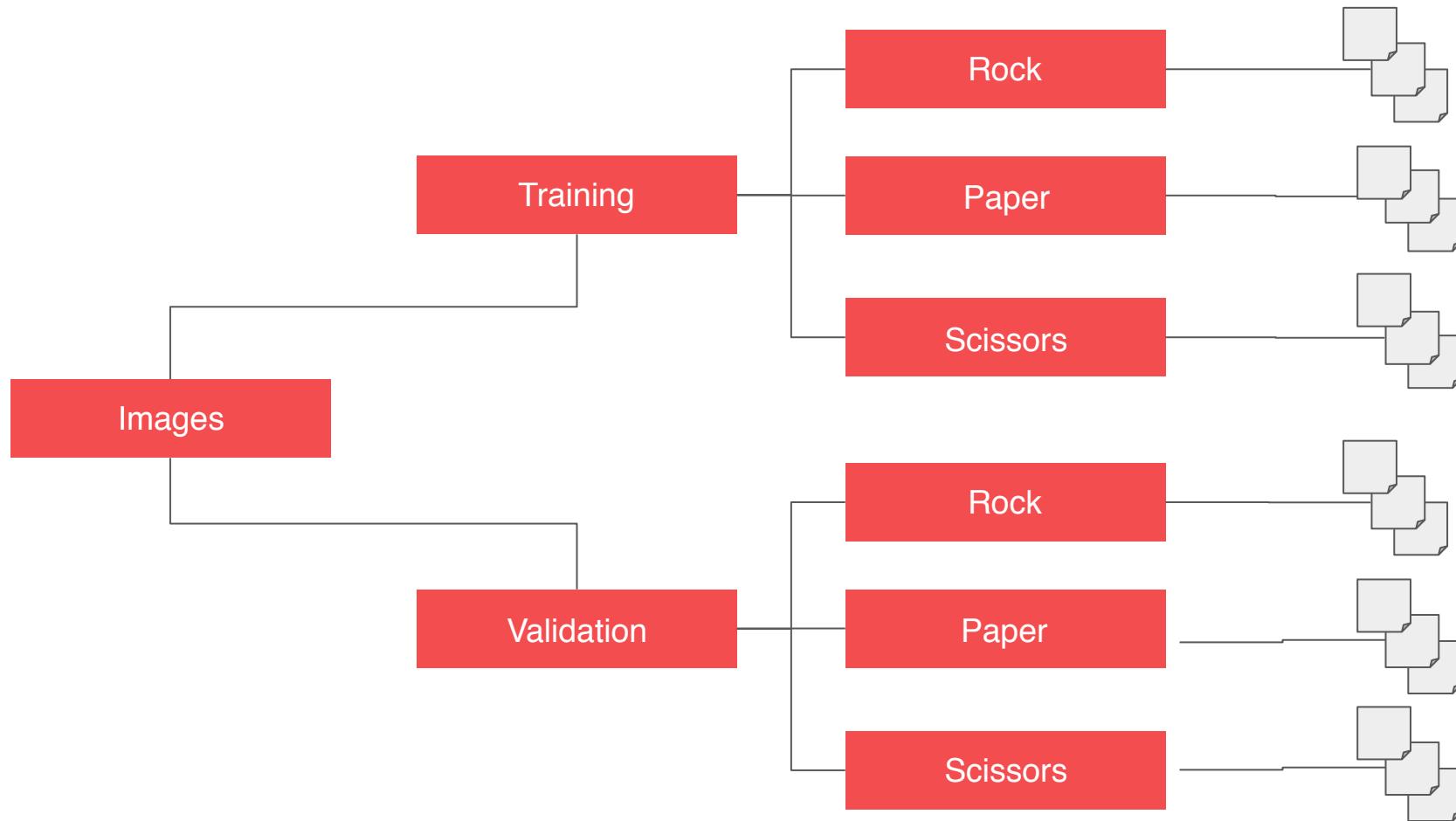
# Copyright Notice

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>







<http://www.laurencemoroney.com/rock-paper-scissors-dataset/>



## Rock Paper Scissors Dataset

# Introducing Rock Paper Scissors – A multi class learning dataset

## Abstract

Rock Paper Scissors is a dataset containing 2,892 images of diverse hands in Rock/Paper/Scissors poses. It is licensed [CC By 2.0](#) and available for all purposes, but its intent is primarily for learning and research.

## Overview

Rock Paper Scissors contains images from a variety of different hands, from different races, ages and genders, posed into Rock / Paper or Scissors and labelled as such. You can download the [training set here](#), and the [test set here](#). These images have all been generated using CGI techniques as an experiment in determining if a CGI-based dataset can be

## FOLLOW ME ON TWITTER

Laurence Moroney Retweeted



**TensorFlow** @TensorFlow

In the third episode of the Intro to Google Colaboratory series, @lmoroney covers how to quickly build a neural network for basic Breast Cancer classification.

Watch this #CodingTensorFlow → [bit.ly/2SPD4XC](https://bit.ly/2SPD4XC)

P.S. Don't forget to add your homework below!



```
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='categorical')
```

```
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(3, activation='softmax')
])
```



Rock: 0.001 Paper:  
0.647

Scissors:  
0.352

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

```
model.compile(loss='binary_crossentropy',
```

```
    optimizer=RMSprop(lr=0.001),
```

```
    metrics=['acc'])
```

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

```
model.compile(loss='categorical_crossentropy',
```

```
    optimizer=RMSprop(lr=0.001),
```

```
    metrics=['acc'])
```

