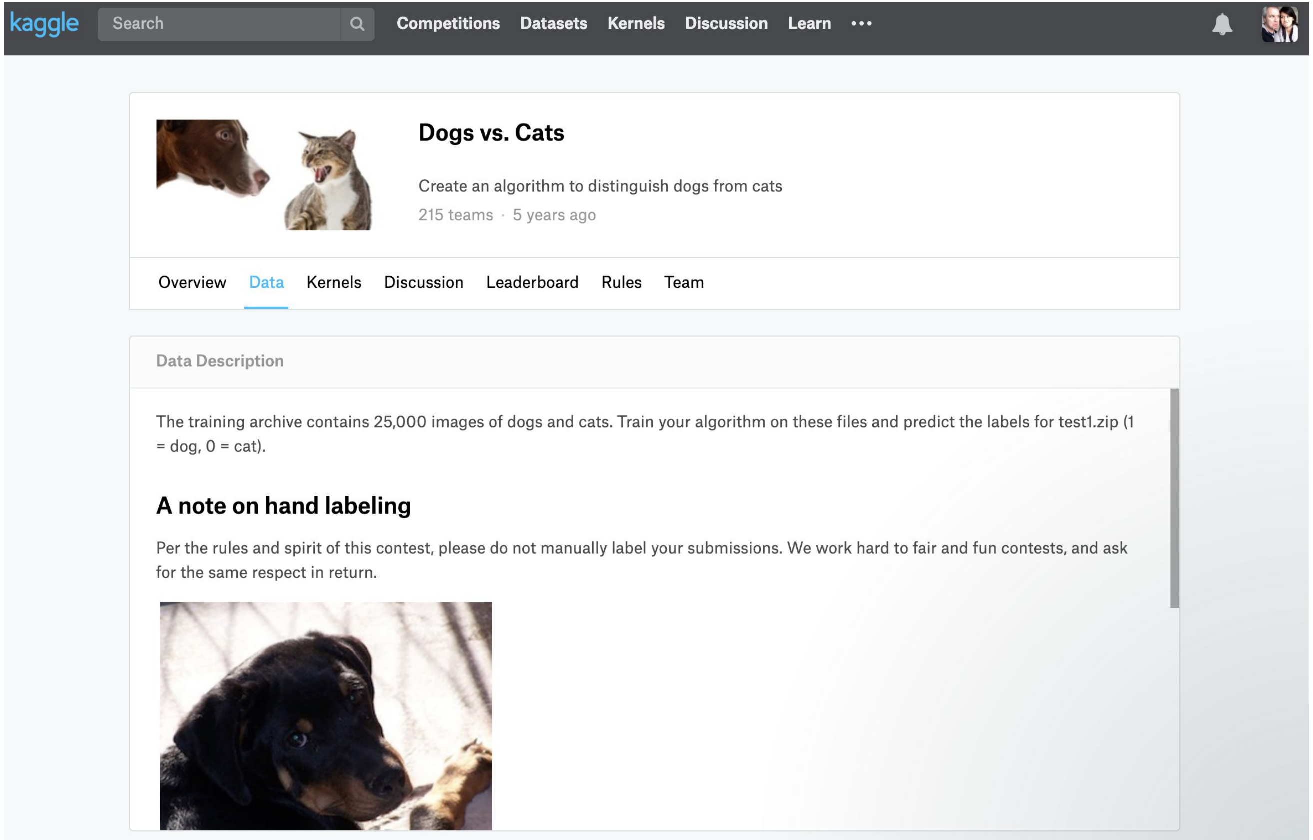
Copyright Notice

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



Dogs vs. Cats

Create an algorithm to distinguish dogs from cats

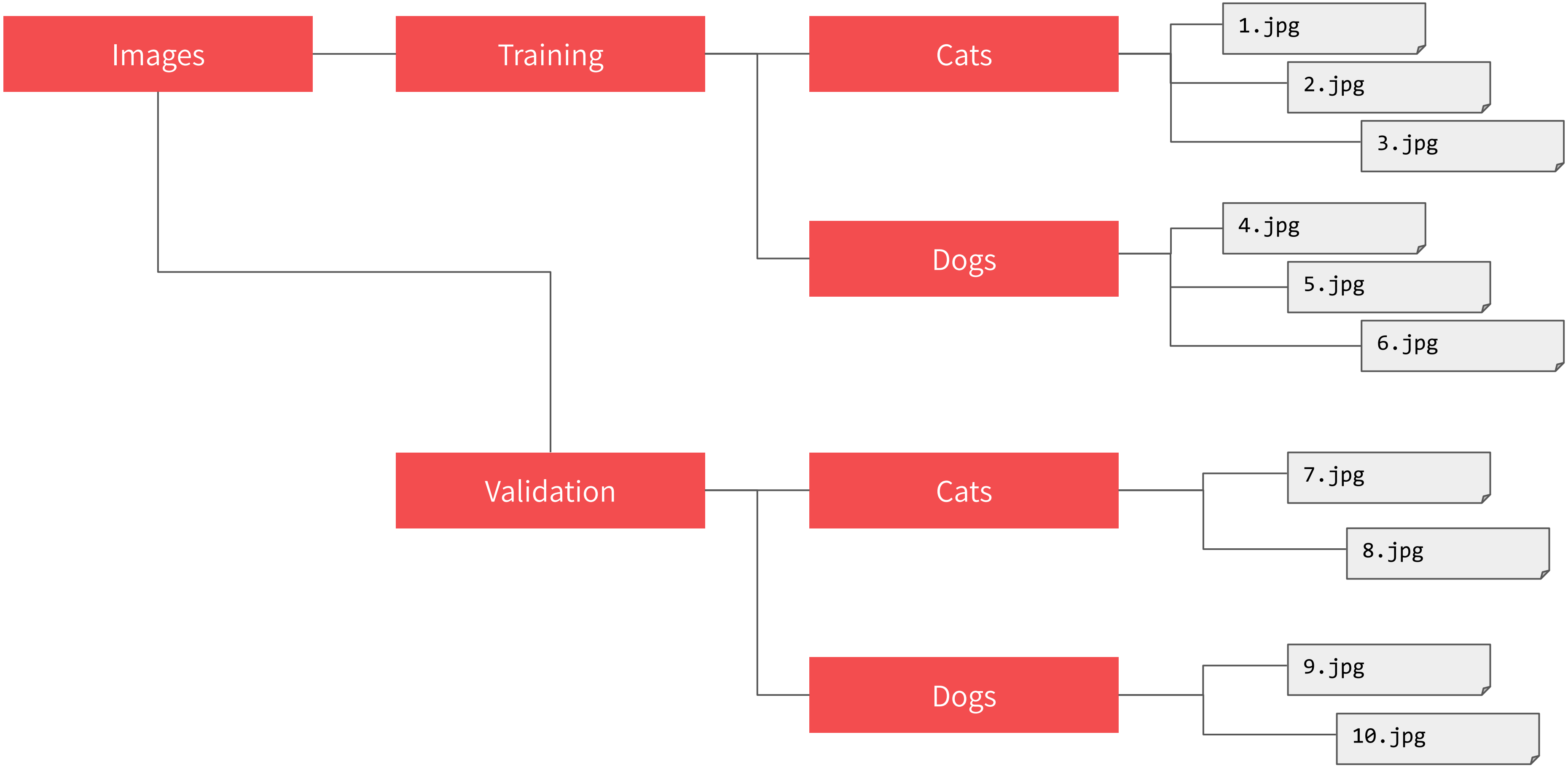
215 teams · 5 years ago

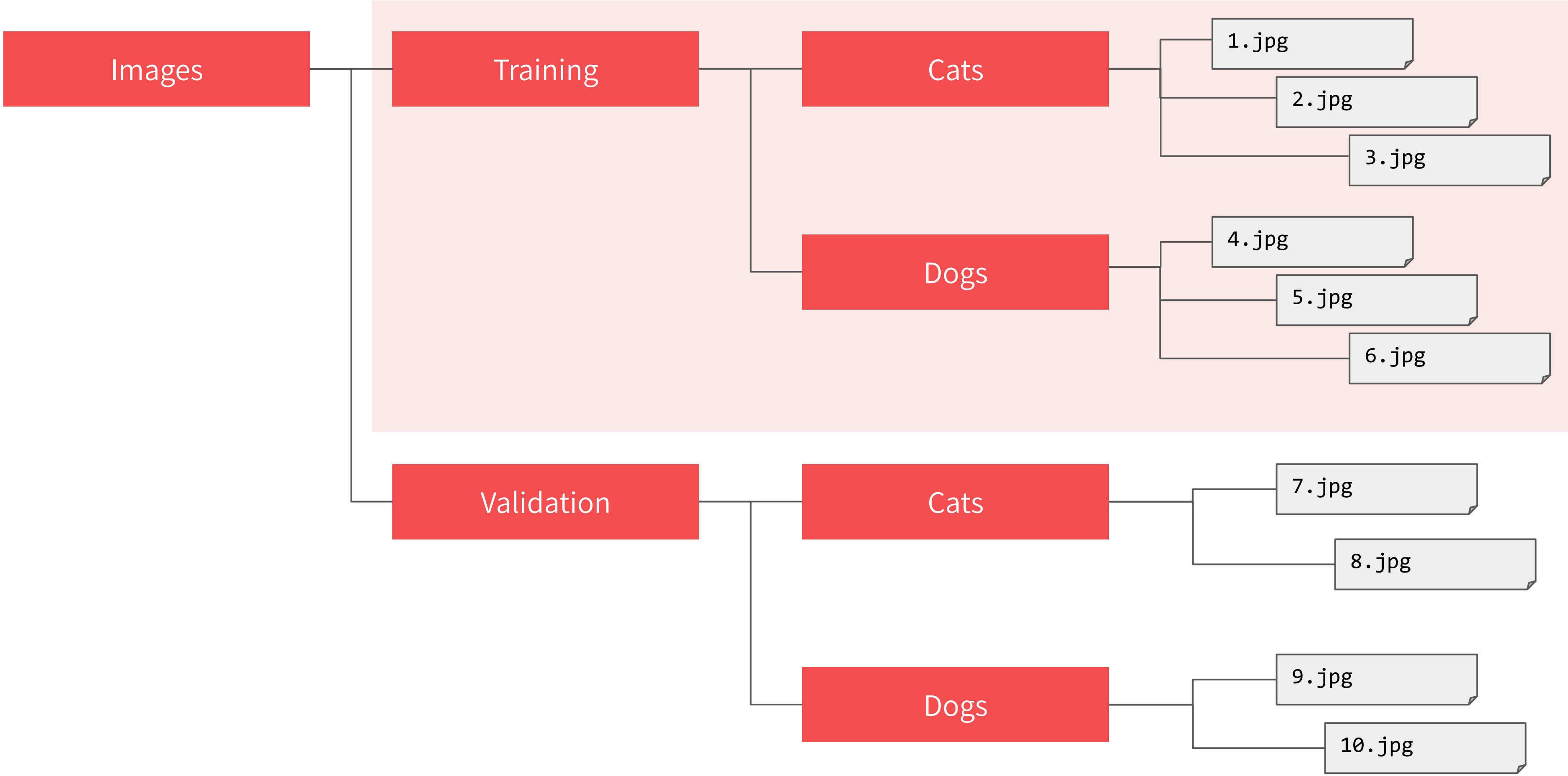
Data Description

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.


Images
Training
Cats
1.jpg
2.jpg
3.jpg
Dogs
4.jpg
5.jpg
6.jpg
Validation
Cats
7.jpg
8.jpg
Dogs
9.jpg
10.jpg



Images

Training

Cats

1.jpg

2.jpg

3.jpg

Dogs

4.jpg

5.jpg

6.jpg

Validation

Cats

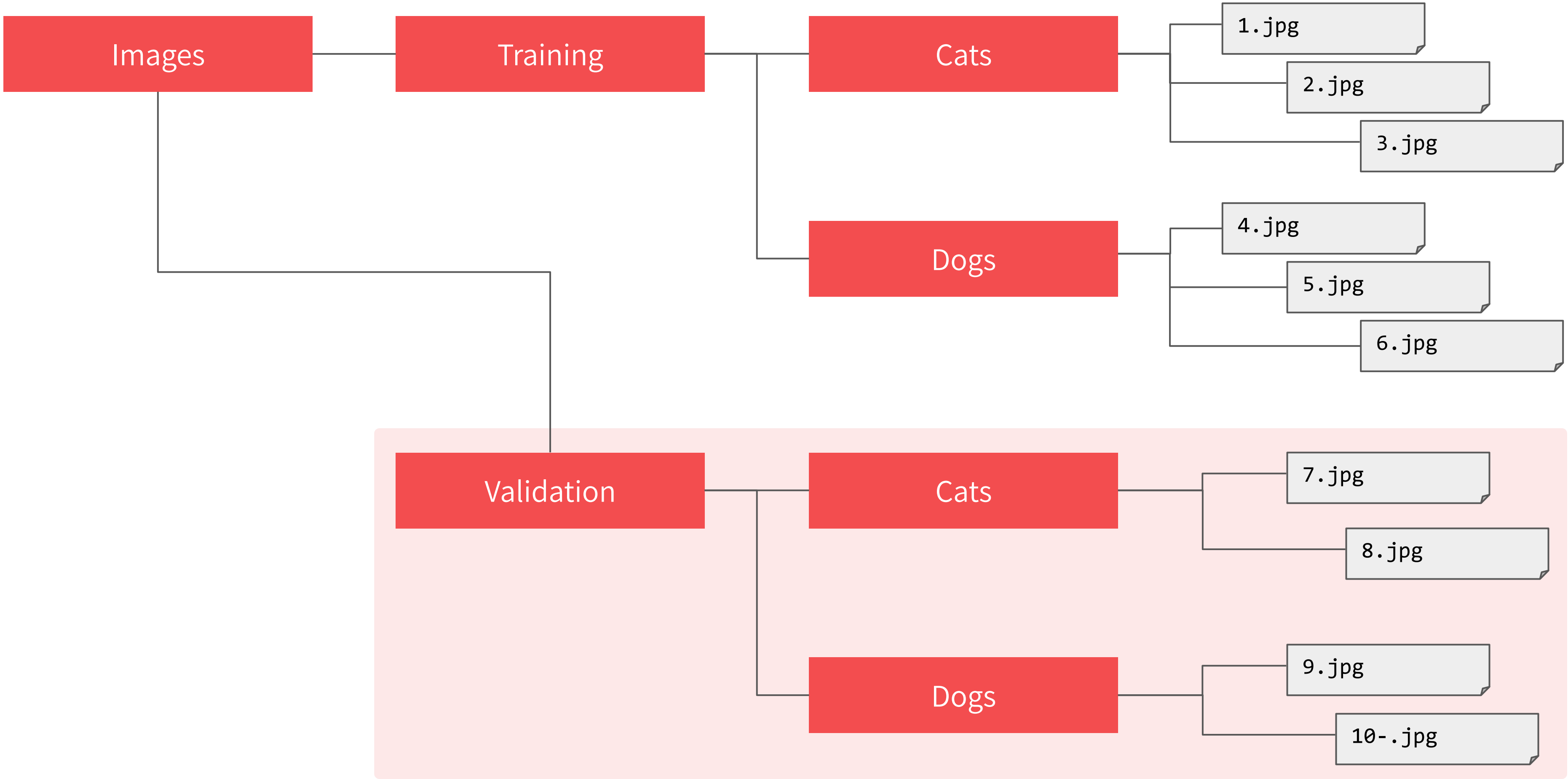
7.jpg

8.jpg

Dogs

9.jpg

10.jpg



Images

Training

Cats

1.jpg

2.jpg

3.jpg

Dogs

4.jpg

5.jpg

6.jpg

Validation

Cats

7.jpg

8.jpg

Dogs

9.jpg

10-.jpg

```python
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')
```

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')
```

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')
```

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')
```

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')
```

```
validaton_dataset = tf.keras.utils.image_dataset_from_directory(
    validation_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')
```

```
AUTOTUNE = tf.data.AUTOTUNE

train_dataset_final =
train_dataset_scaled.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)

validation_dataset_final =
validation_dataset_scaled.cache().prefetch(buffer_size=AUTOTUNE)
```

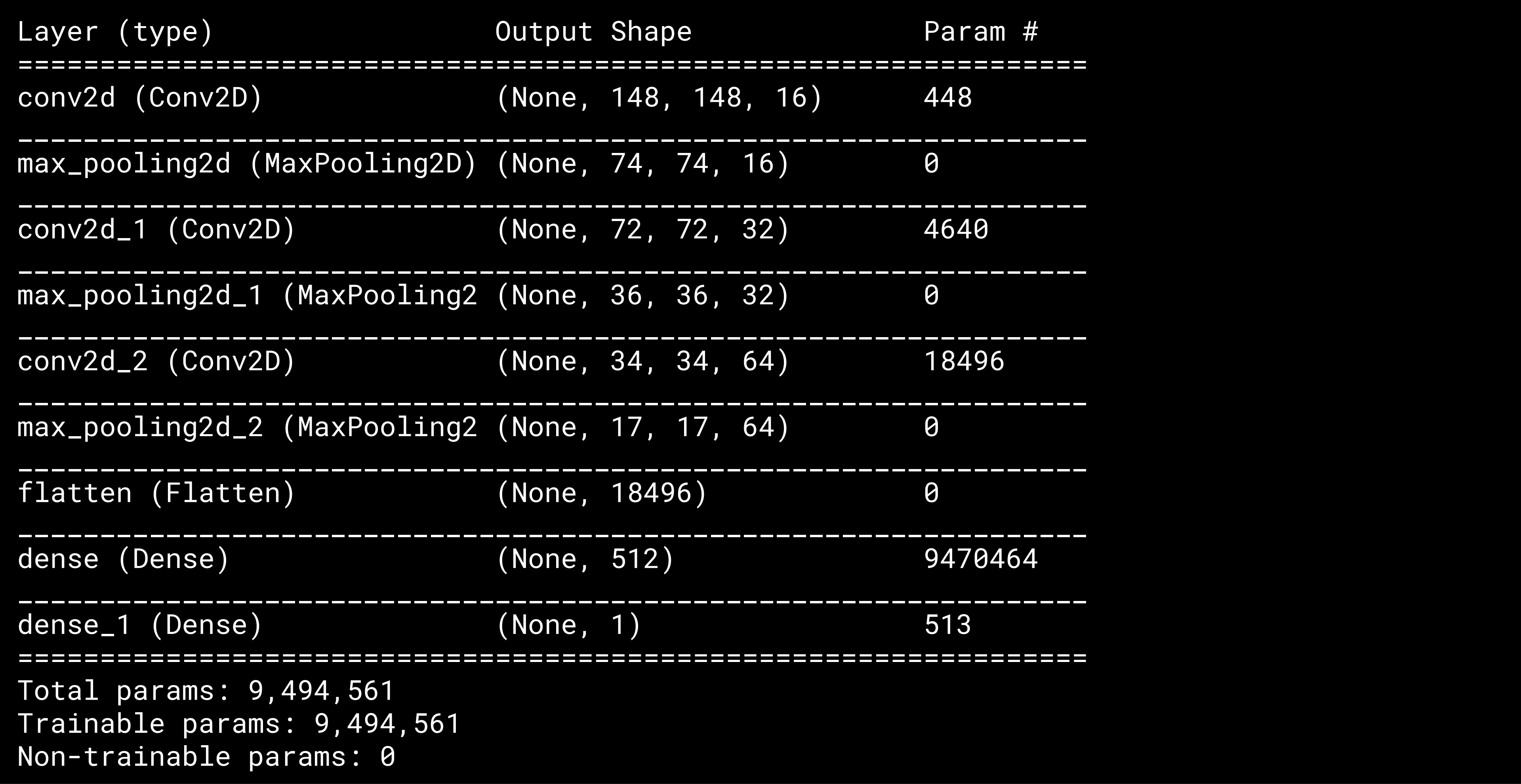
```
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 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')
])
```

```python
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    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.Input(shape=(150, 150, 3)),
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    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.Input(shape=(150, 150, 3)),
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    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.Input(shape=(150, 150, 3)),
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    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')
])
```

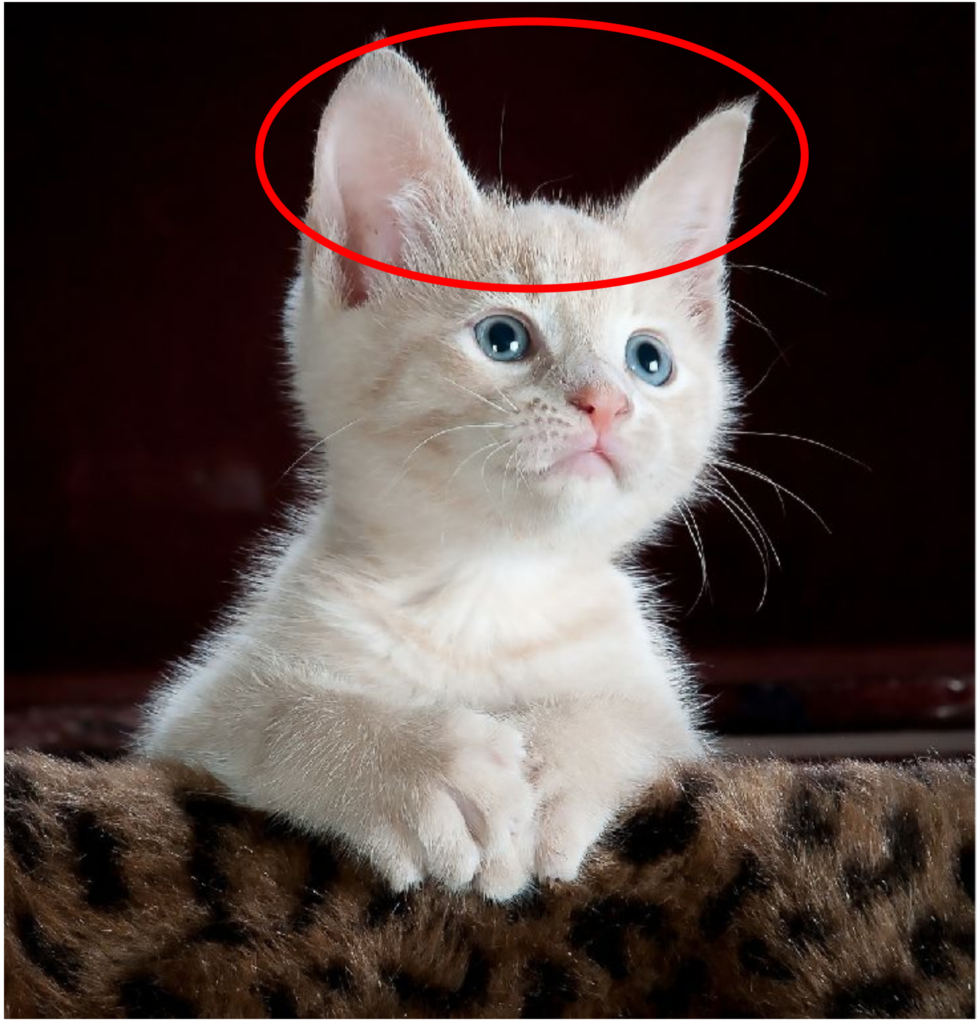


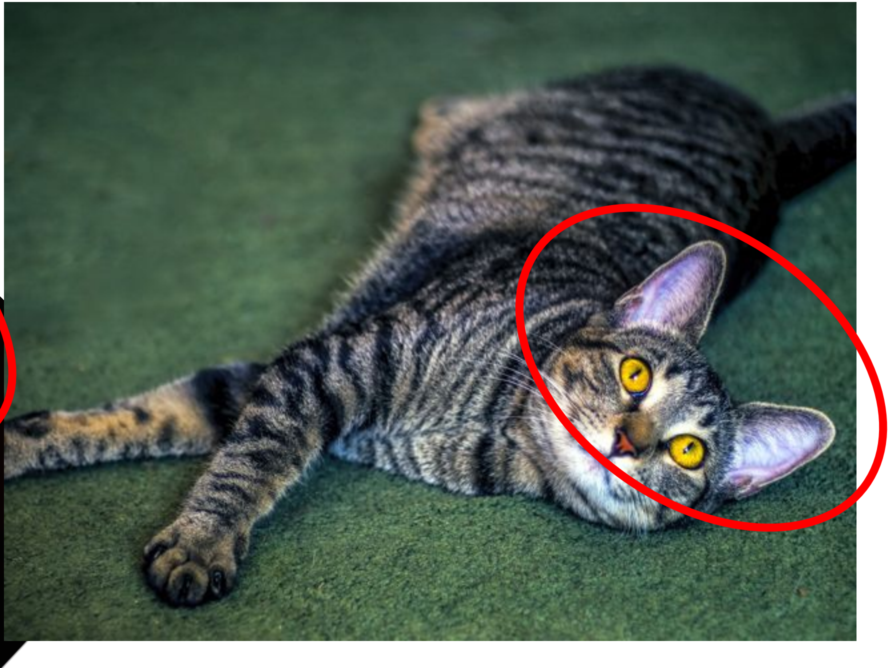
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 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
```

```
model.compile(loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001),
    metrics=['accuracy'])
```

```
history = model.fit(
      train_dataset_final,
      epochs=15,
      validation_data=validation_dataset_final,
      verbose=2)
```

Copyright Notice





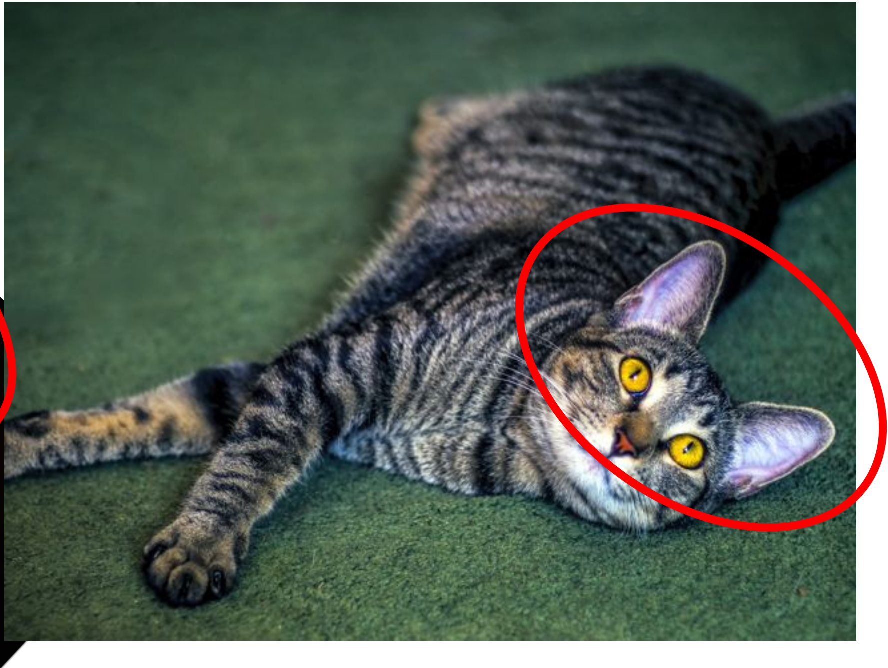
```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')

SHUFFLE_BUFFER_SIZE = 1000
PREFETCH_BUFFER_SIZE = tf.data.AUTOTUNE

train_dataset_final = (train_dataset
                       .cache()
                       .shuffle(SHUFFLE_BUFFER_SIZE)
                       .prefetch(buffer_size=AUTOTUNE))
```

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')

SHUFFLE_BUFFER_SIZE = 1000
PREFETCH_BUFFER_SIZE = tf.data.AUTOTUNE

train_dataset_final = (train_dataset
                       .cache()
                       .shuffle(SHUFFLE_BUFFER_SIZE)
                       .prefetch(buffer_size=AUTOTUNE))
```

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')

SHUFFLE_BUFFER_SIZE = 1000
PREFETCH_BUFFER_SIZE = tf.data.AUTOTUNE

train_dataset_final = (train_dataset
                       .cache()
                       .shuffle(SHUFFLE_BUFFER_SIZE)
                       .prefetch(buffer_size=AUTOTUNE))
```

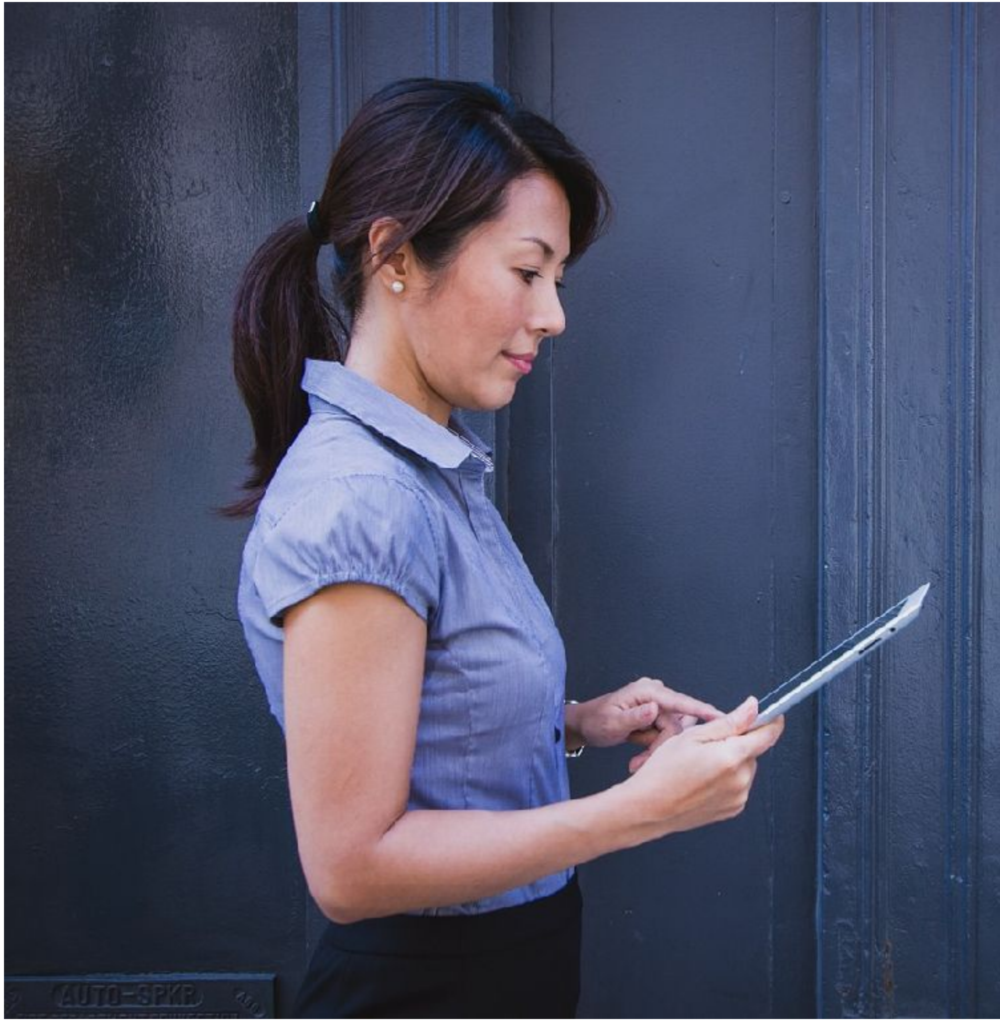
```python
data_augmentation = tf.keras.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2, fill_mode='nearest'),
    tf.keras.layers.RandomTranslation(0.2, 0.2, fill_mode='nearest'),
    tf.keras.layers.RandomZoom(0.2, fill_mode='nearest')
    ])
```

```
data_augmentation = tf.keras.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2, fill_mode='nearest'),
    tf.keras.layers.RandomTranslation(0.2, 0.2, fill_mode='nearest'),
    tf.keras.layers.RandomZoom(0.2, fill_mode='nearest')
    ])
```



```
data_augmentation = tf.keras.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2, fill_mode='nearest'),
    tf.keras.layers.RandomTranslation(0.2, 0.2, fill_mode='nearest'),
    tf.keras.layers.RandomZoom(0.2, fill_mode='nearest')
    ])
```

```
data_augmentation = tf.keras.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2, fill_mode='nearest'),
    tf.keras.layers.RandomTranslation(0.2, 0.2, fill_mode='nearest'),
    tf.keras.layers.RandomZoom(0.2, fill_mode='nearest')
    ])
```



AUTO-SPKR

UTO-SPKR

```
data_augmentation = tf.keras.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2, fill_mode='nearest'),
    tf.keras.layers.RandomTranslation(0.2, 0.2, fill_mode='nearest'),
    tf.keras.layers.RandomZoom(0.2, fill_mode='nearest')
    ])
```

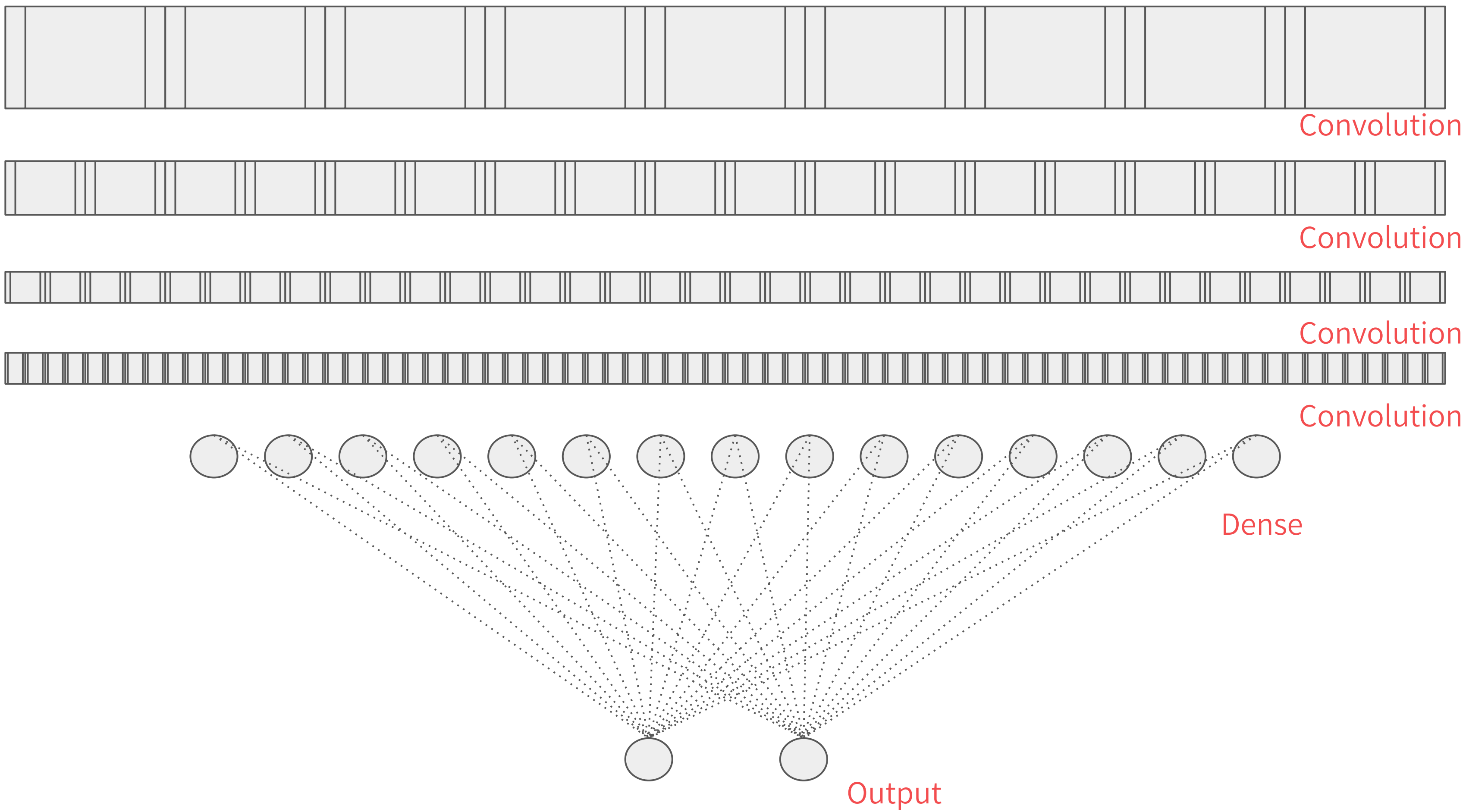
```
data_augmentation = tf.keras.Sequential([
    tf.keras.Input(shape=(150, 150, 3)),
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2, fill_mode='nearest'),
    tf.keras.layers.RandomTranslation(0.2, 0.2, fill_mode='nearest'),
    tf.keras.layers.RandomZoom(0.2, fill_mode='nearest')
    ])
```

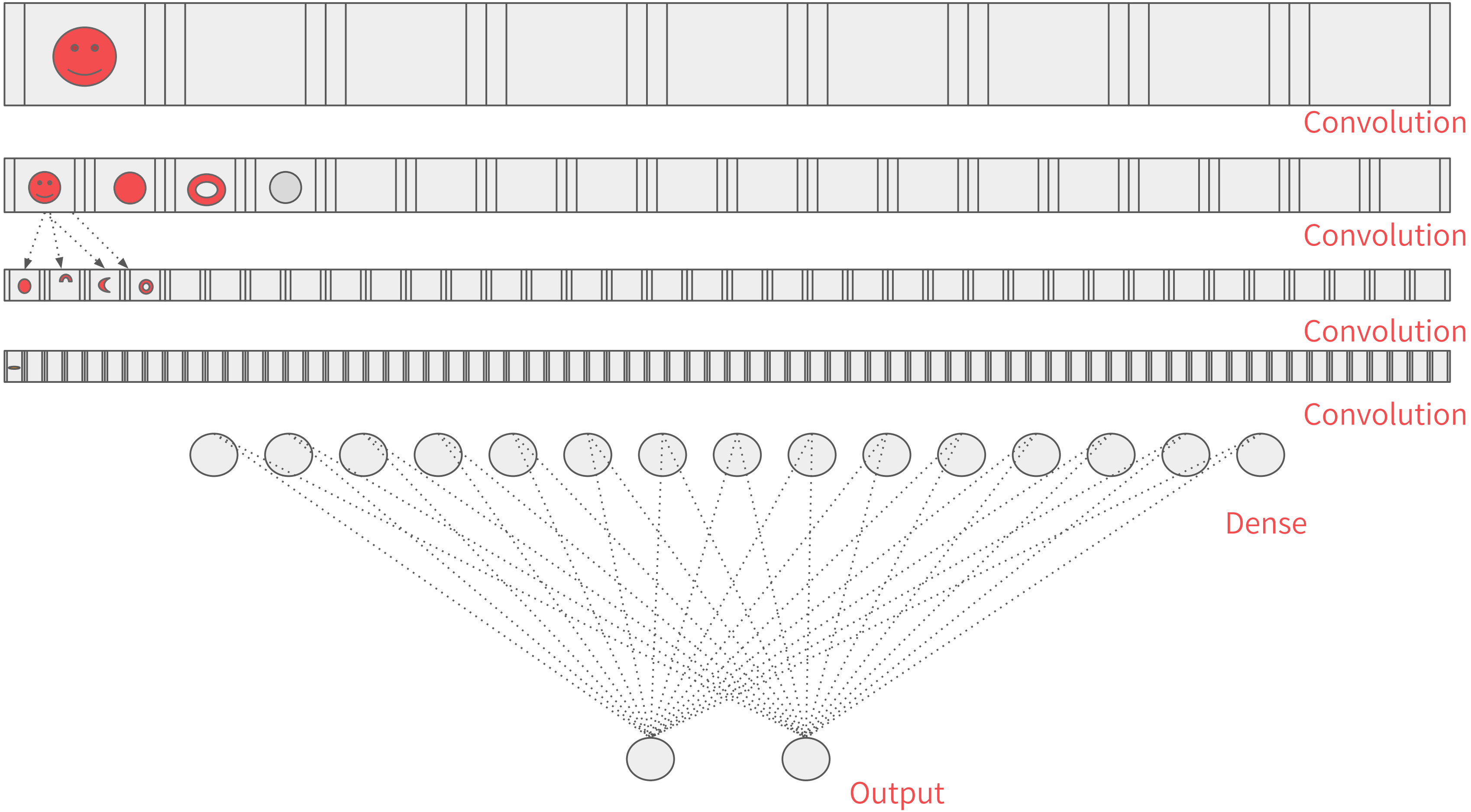
```
model_without_aug = create_model()

model_with_aug = tf.keras.models.Sequential([
    data_augmentation,
    model_without_aug
])

model_with_aug.compile(
    loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
    metrics=['accuracy']
```

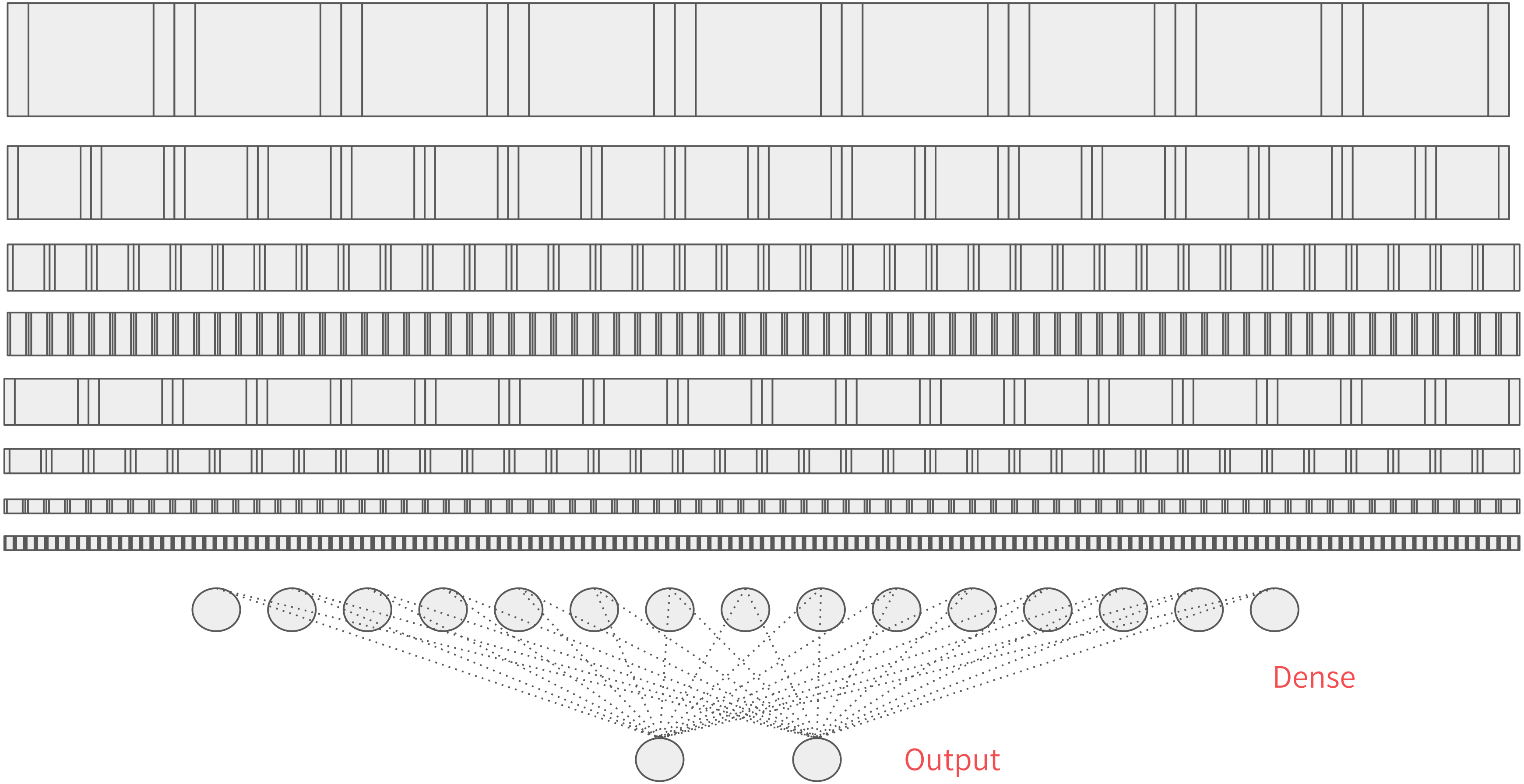
```
model_without_aug = create_model()

model_with_aug = tf.keras.models.Sequential([
    data_augmentation,
    model_without_aug
])

model_with_aug.compile(
    loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
    metrics=['accuracy']
```

```
model_without_aug = create_model()

model_with_aug = tf.keras.models.Sequential([
    data_augmentation,
    model_without_aug
])

model_with_aug.compile(
    loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
    metrics=['accuracy']
```

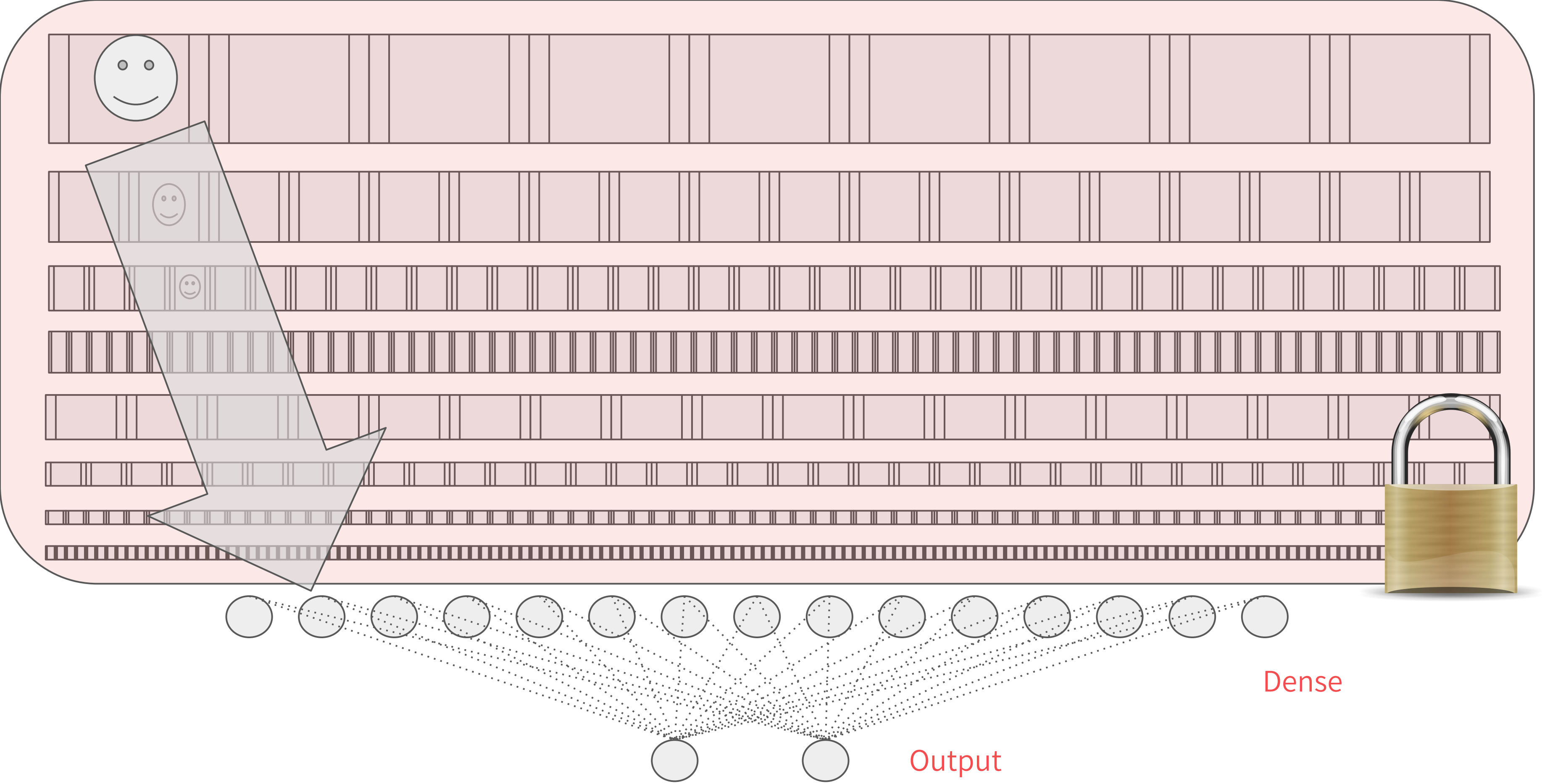
Copyright Notice

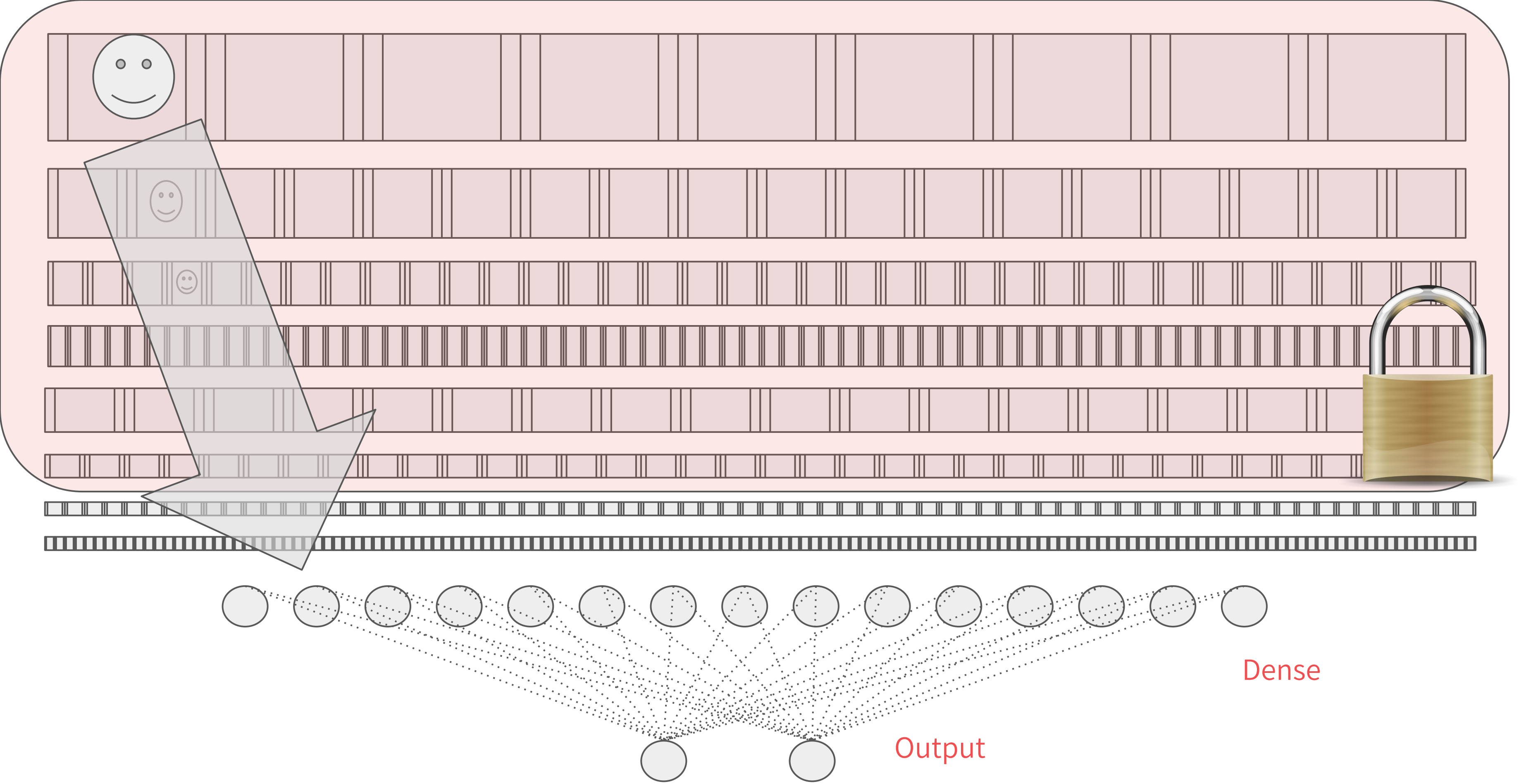

Convolution
Convolution
Convolution
Convolution
Dense
Output


Convolution
Convolution
Convolution
Convolution
Dense
Output

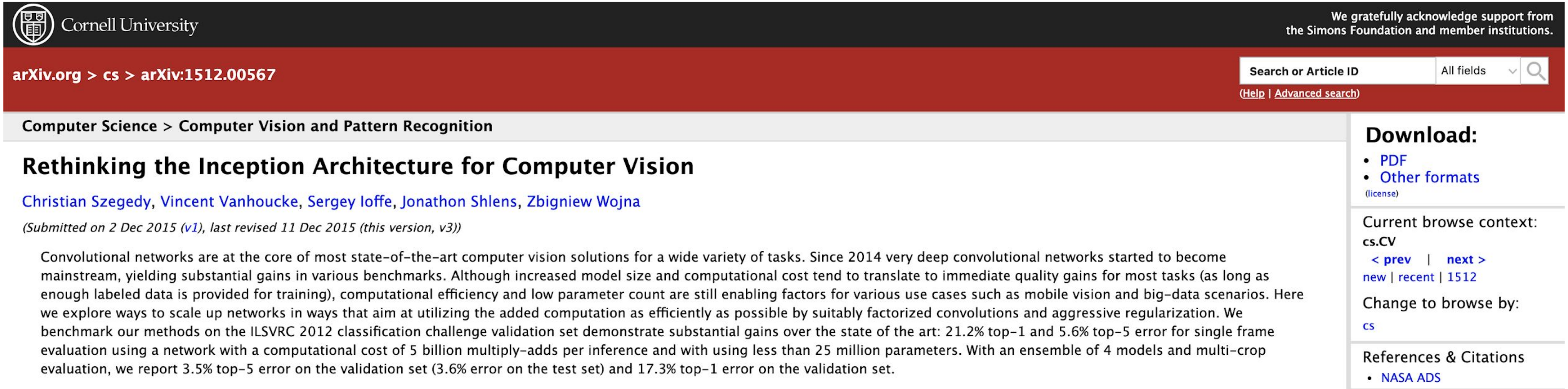


Dense
Output


Dense
Output


Dense
Output

https://arxiv.org/abs/1512.00567



Cornell University
We gratefully acknowledge support from the Simons Foundation and member institutions.
arXiv.org > cs > arXiv:1512.00567
Search or Article ID
All fields
(Help | Advanced search)

Computer Science > Computer Vision and Pattern Recognition

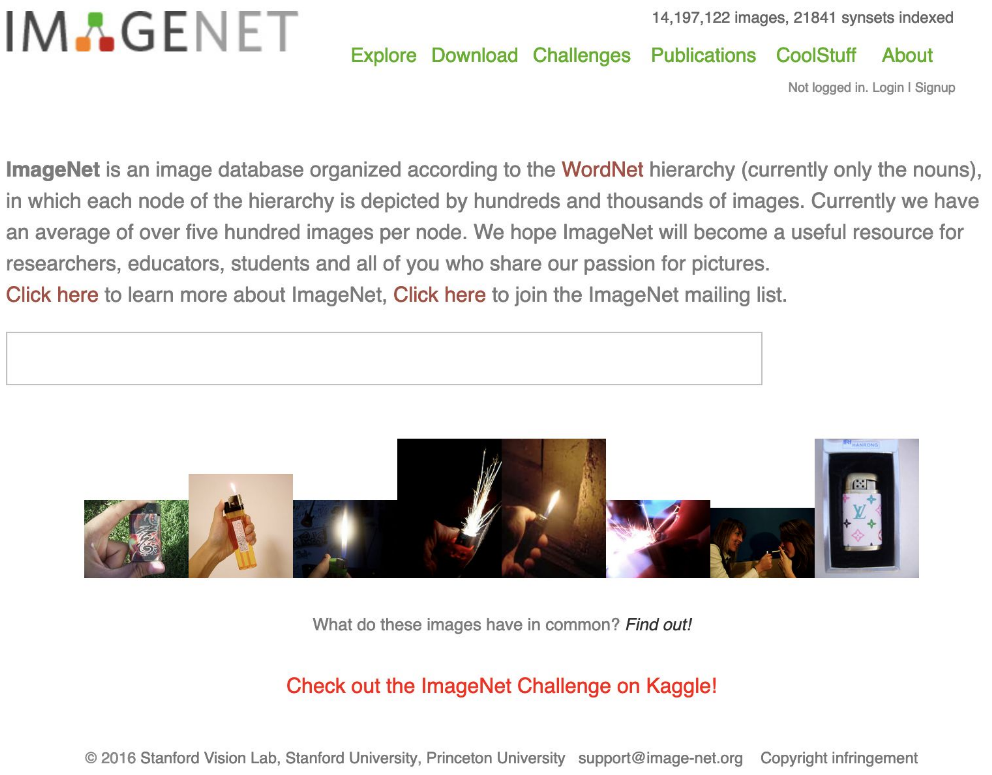
Rethinking the Inception Architecture for Computer Vision

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.

http://image-net.org/


IMAGENET
14,197,122 images, 21841 synsets indexed
Explore
Download
Challenges
Publications
CoolStuff
About
Not logged in. Login I 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!*

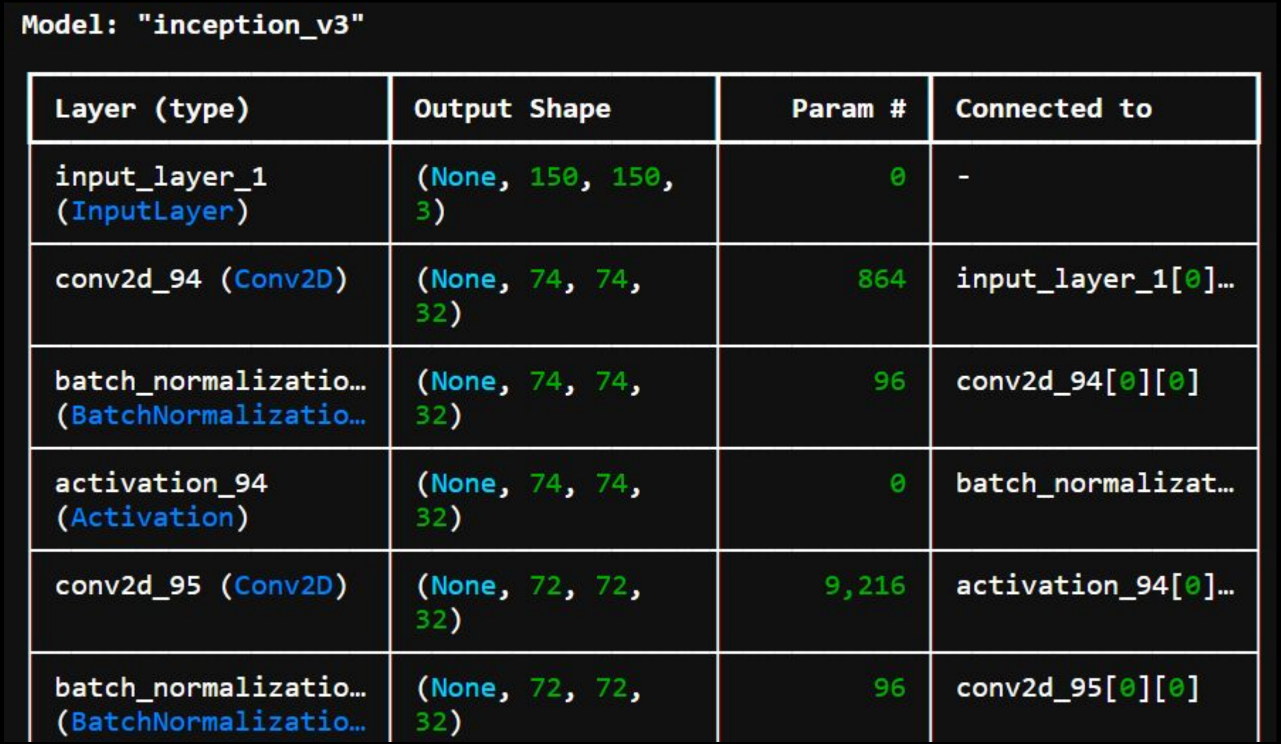
Check out the ImageNet Challenge on Kaggle!

https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5

```
local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

pre_trained_model = tf.keras.applications.inception_v3.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()
```



```
Model: "inception_v3"

┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape       ┃ Param # ┃ Connected to        ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━┩
│ input_layer_1       │ (None, 150, 150,   │       0 │ -                   │
│ (InputLayer)        │ 3)                 │         │                     │
├─────────────────────┼────────────────────┼─────────┼─────────────────────┤
│ conv2d_94 (Conv2D)  │ (None, 74, 74,     │     864 │ input_layer_1[0]…   │
│                     │ 32)                │         │                     │
├─────────────────────┼────────────────────┼─────────┼─────────────────────┤
│ batch_normalizatio… │ (None, 74, 74,     │      96 │ conv2d_94[0][0]     │
│ (BatchNormalizatio… │ 32)                │         │                     │
├─────────────────────┼────────────────────┼─────────┼─────────────────────┤
│ activation_94       │ (None, 74, 74,     │       0 │ batch_normalizat…   │
│ (Activation)        │ 32)                │         │                     │
├─────────────────────┼────────────────────┼─────────┼─────────────────────┤
│ conv2d_95 (Conv2D)  │ (None, 72, 72,     │   9,216 │ activation_94[0]…   │
│                     │ 32)                │         │                     │
├─────────────────────┼────────────────────┼─────────┼─────────────────────┤
│ batch_normalizatio… │ (None, 72, 72,     │      96 │ conv2d_95[0][0]     │
│ (BatchNormalizatio… │ 32)                │         │                     │
```

. . .

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

last_output = last_layer.output
```

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

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

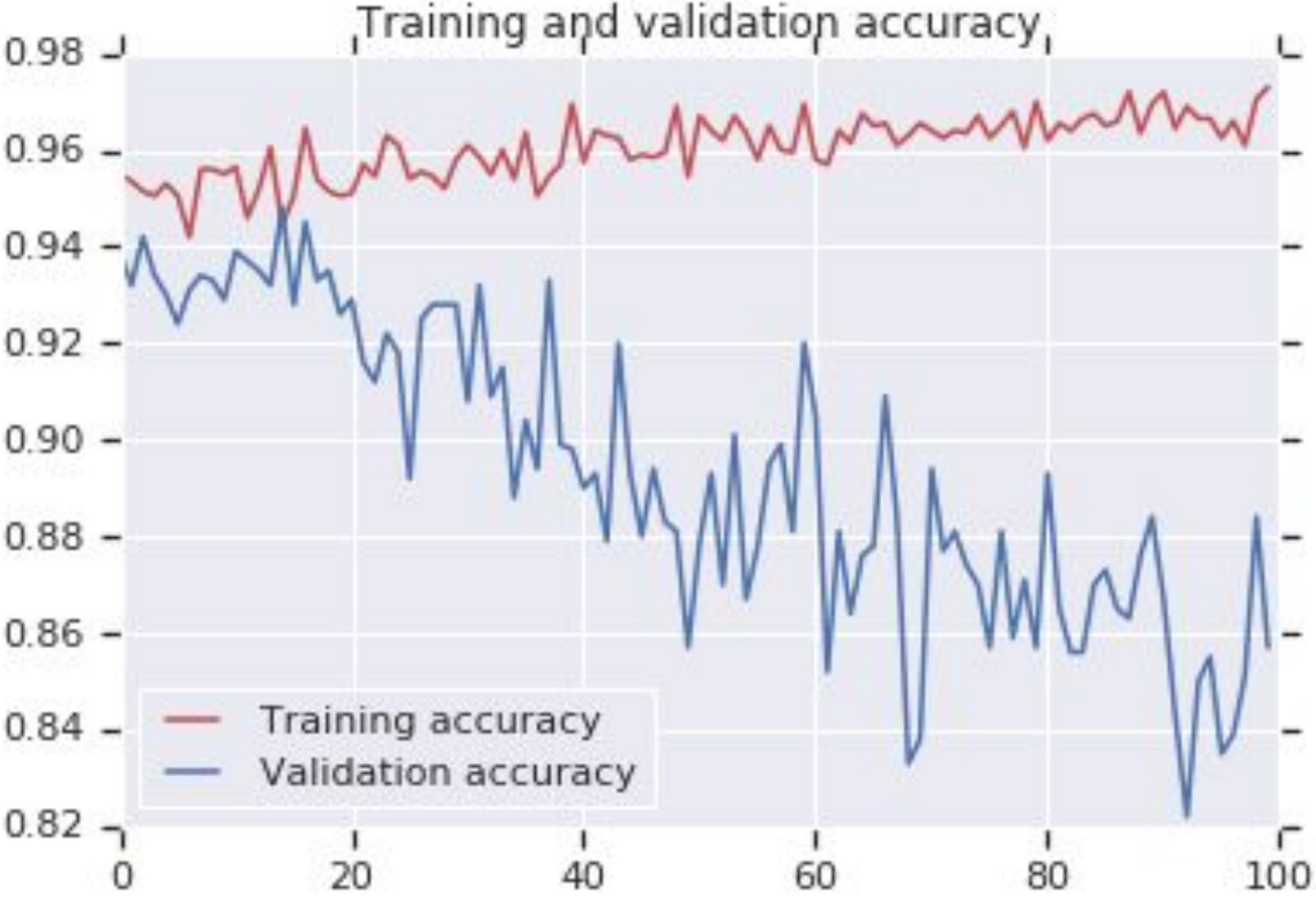
```
x = tf.keras.layers.Flatten()(last_output)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)

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

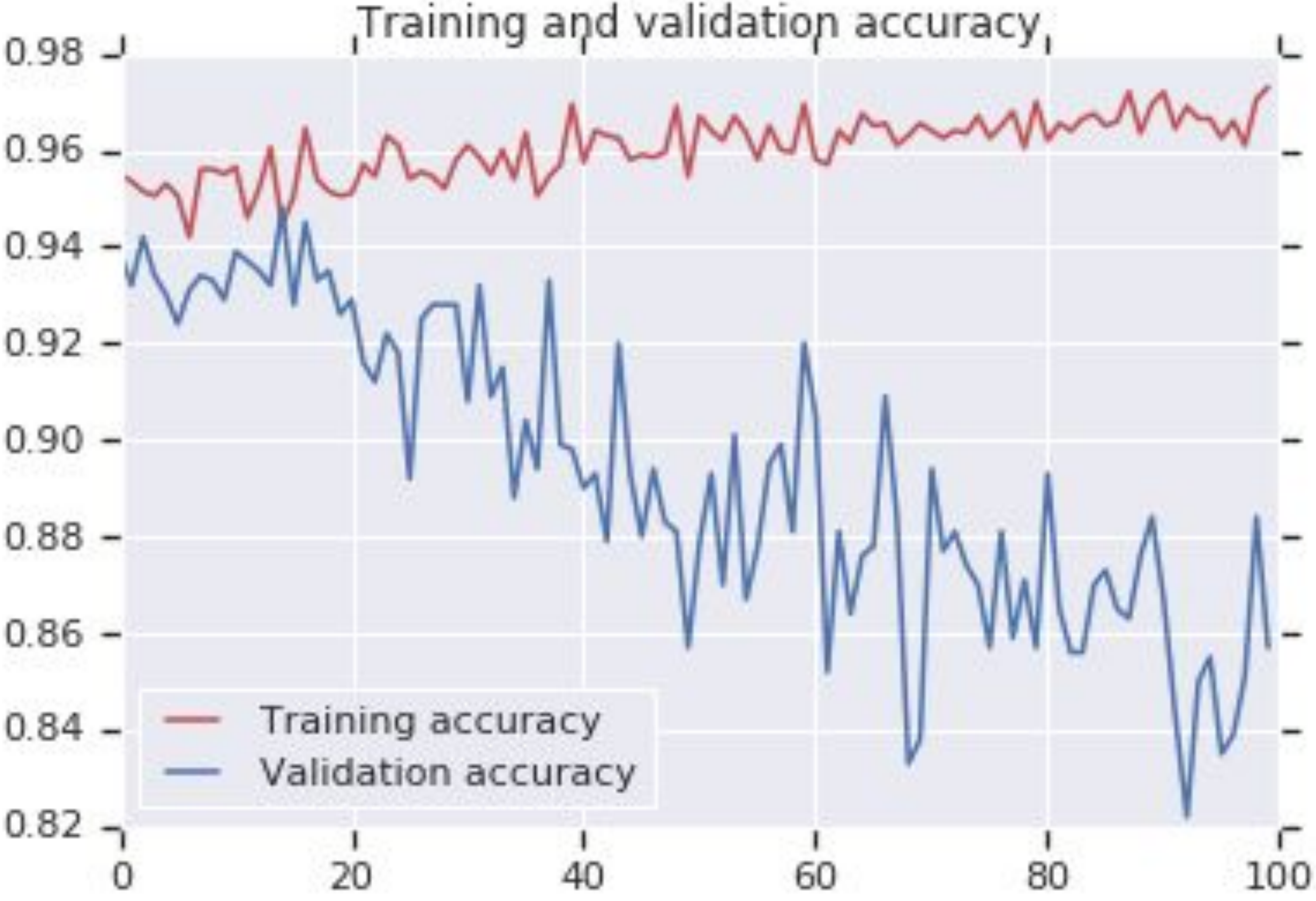
```
x = tf.keras.layers.Flatten()(last_output)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)

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

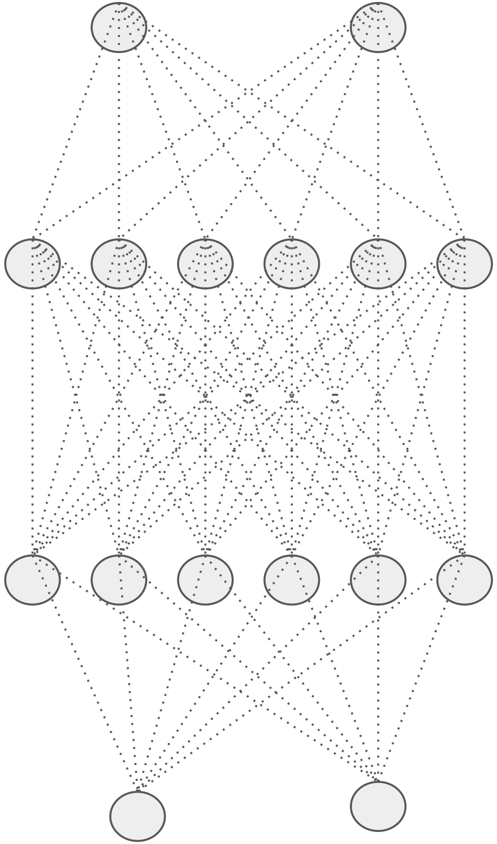
```
x = tf.keras.layers.Flatten()(last_output)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)

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

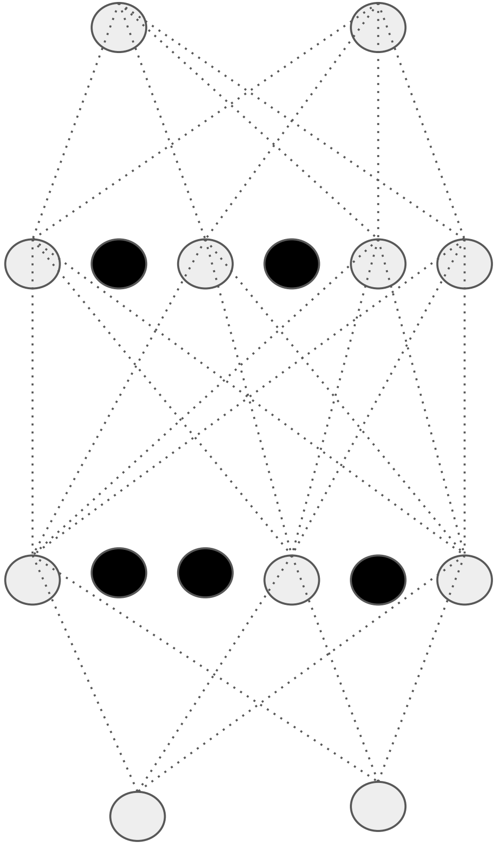
```
x = tf.keras.layers.Flatten()(last_output)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)

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

```
history = model_with_aug.fit(
    train_dataset_final,
    validation_data=validation_dataset_final,
    epochs=20,
    verbose=2)
```


Training and validation accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0
20
40
60
80
100
Training accuracy
Validation accuracy

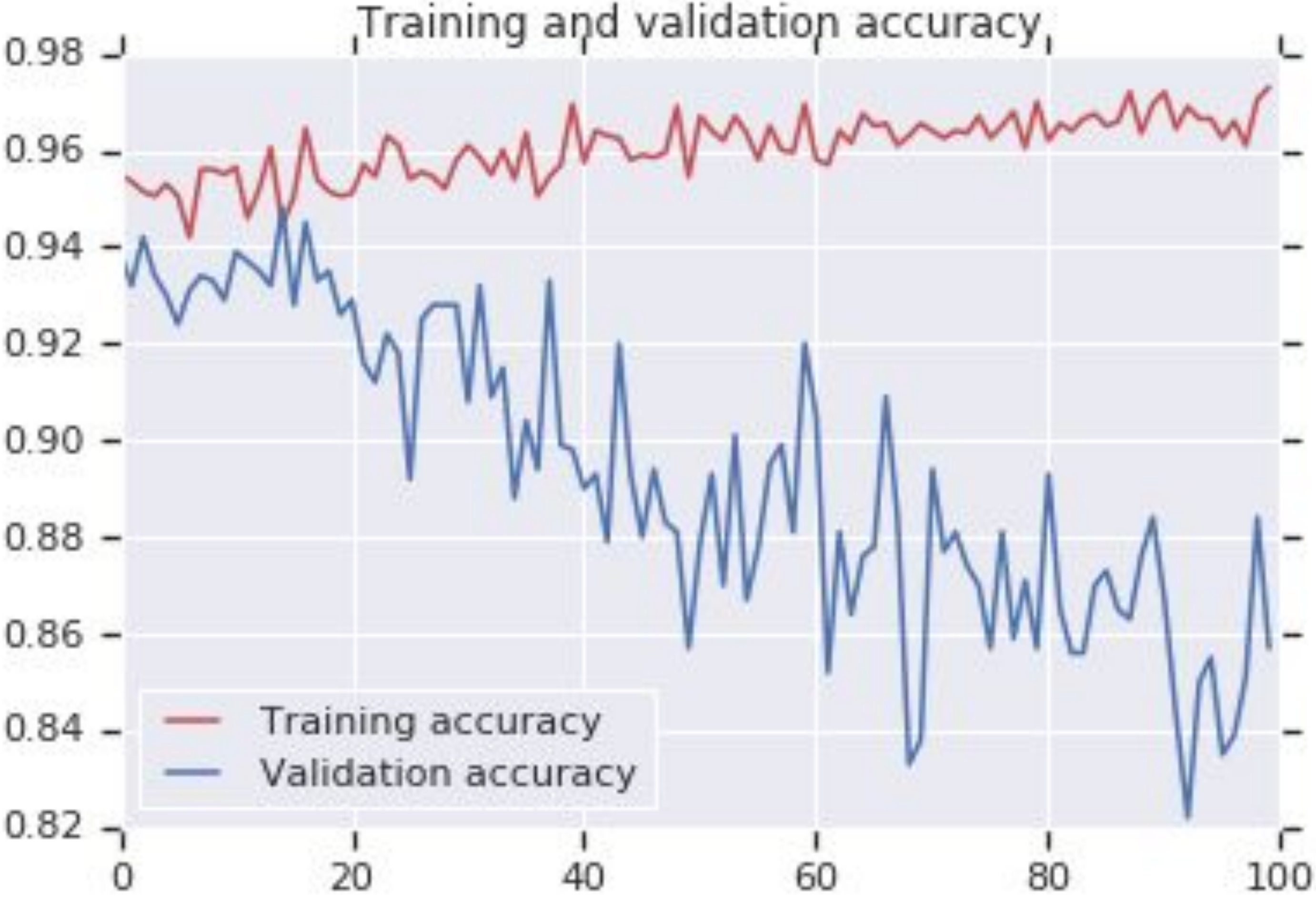

Training and validation accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0
20
40
60
80
100
Training accuracy
Validation accuracy

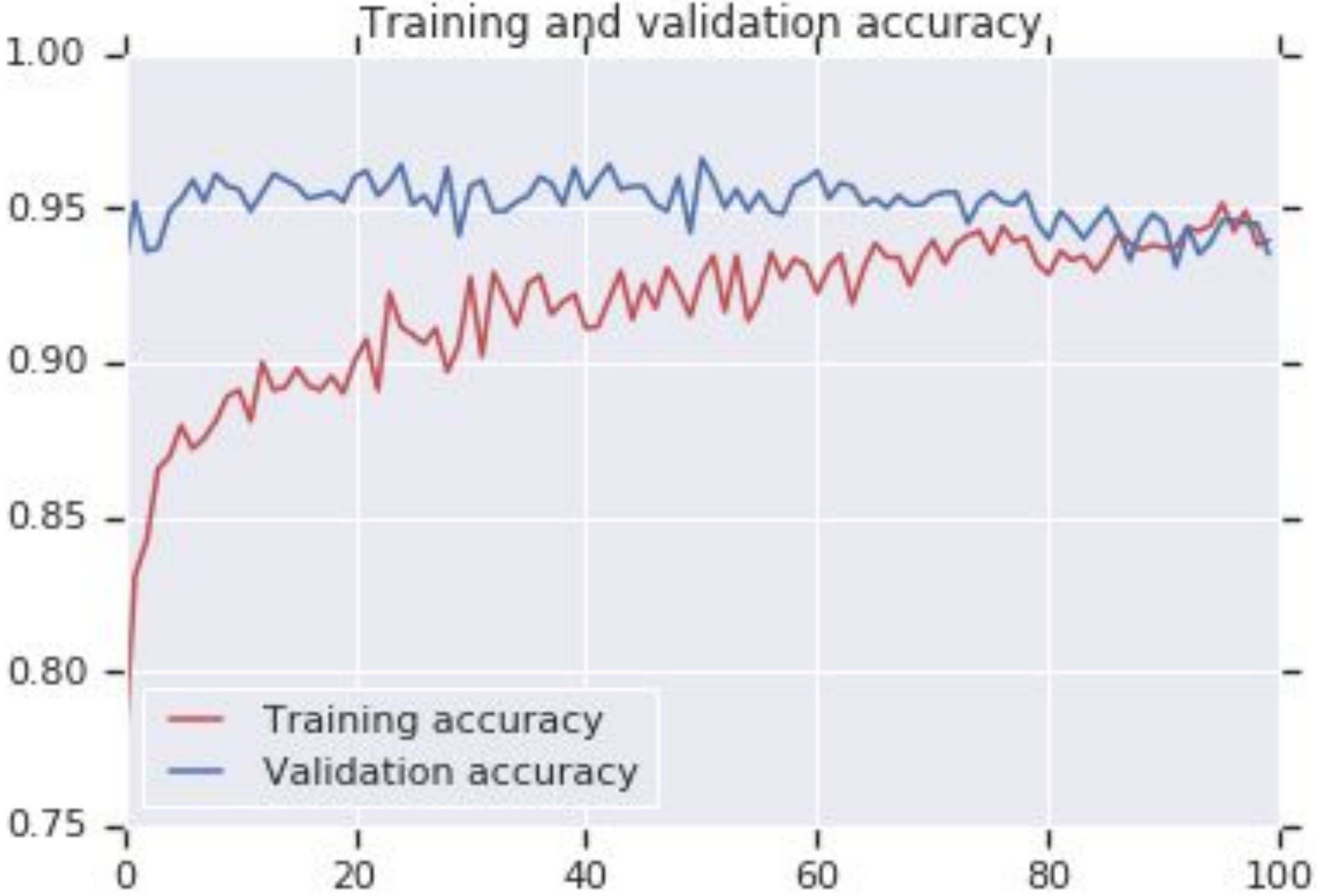




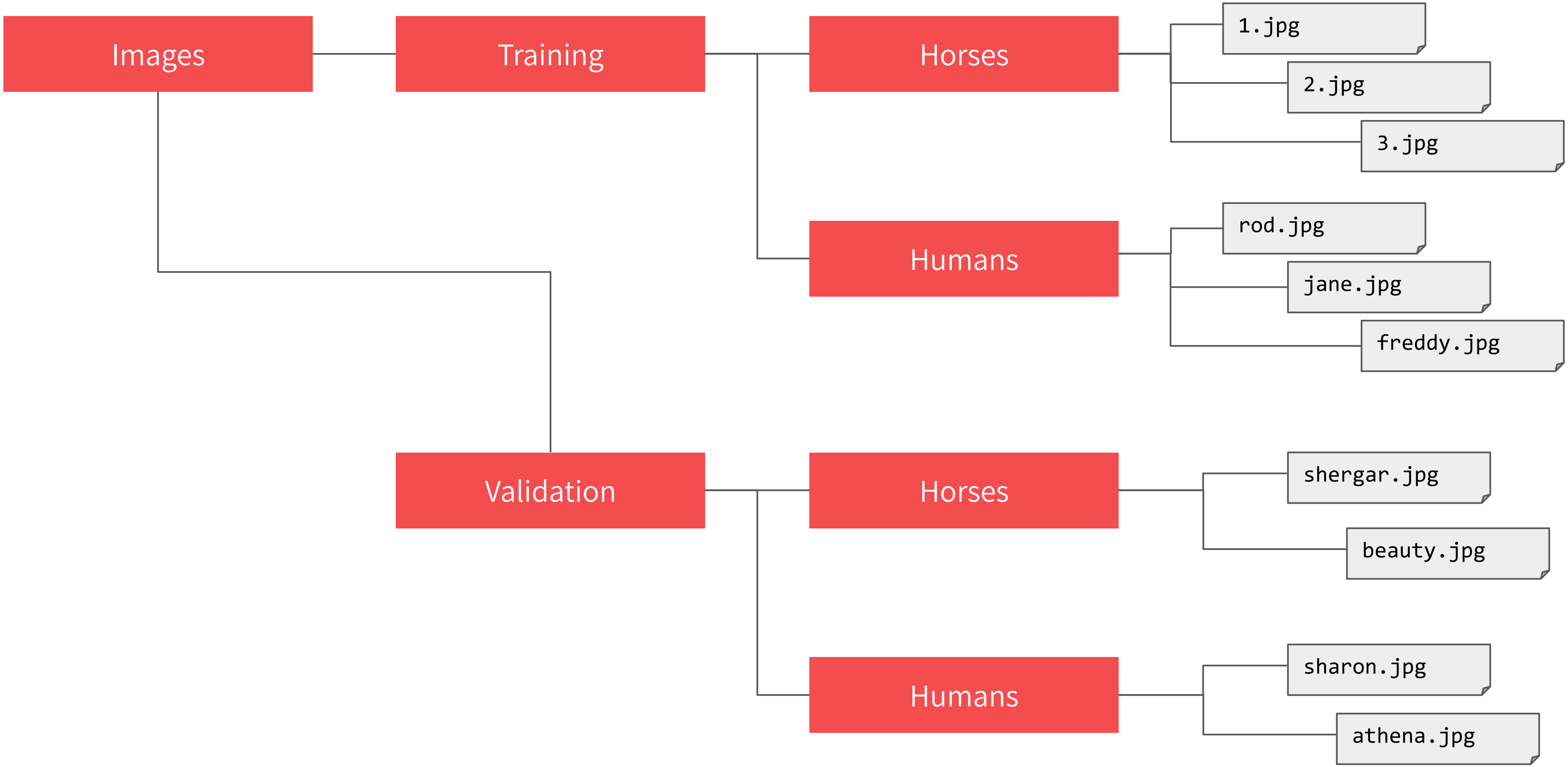
```
x = tf.keras.layers.Flatten()(last_output)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)

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

```
x = tf.keras.layers.Flatten()(last_output)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)

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


Training and validation accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0
20
40
60
80
100
Training accuracy
Validation accuracy


Training and validation accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0
20
40
60
80
100
Training accuracy
Validation accuracy

Copyright Notice



Images

Training

Horses

1.jpg

2.jpg

3.jpg

Humans

rod.jpg

jane.jpg

freddy.jpg

Validation

Horses

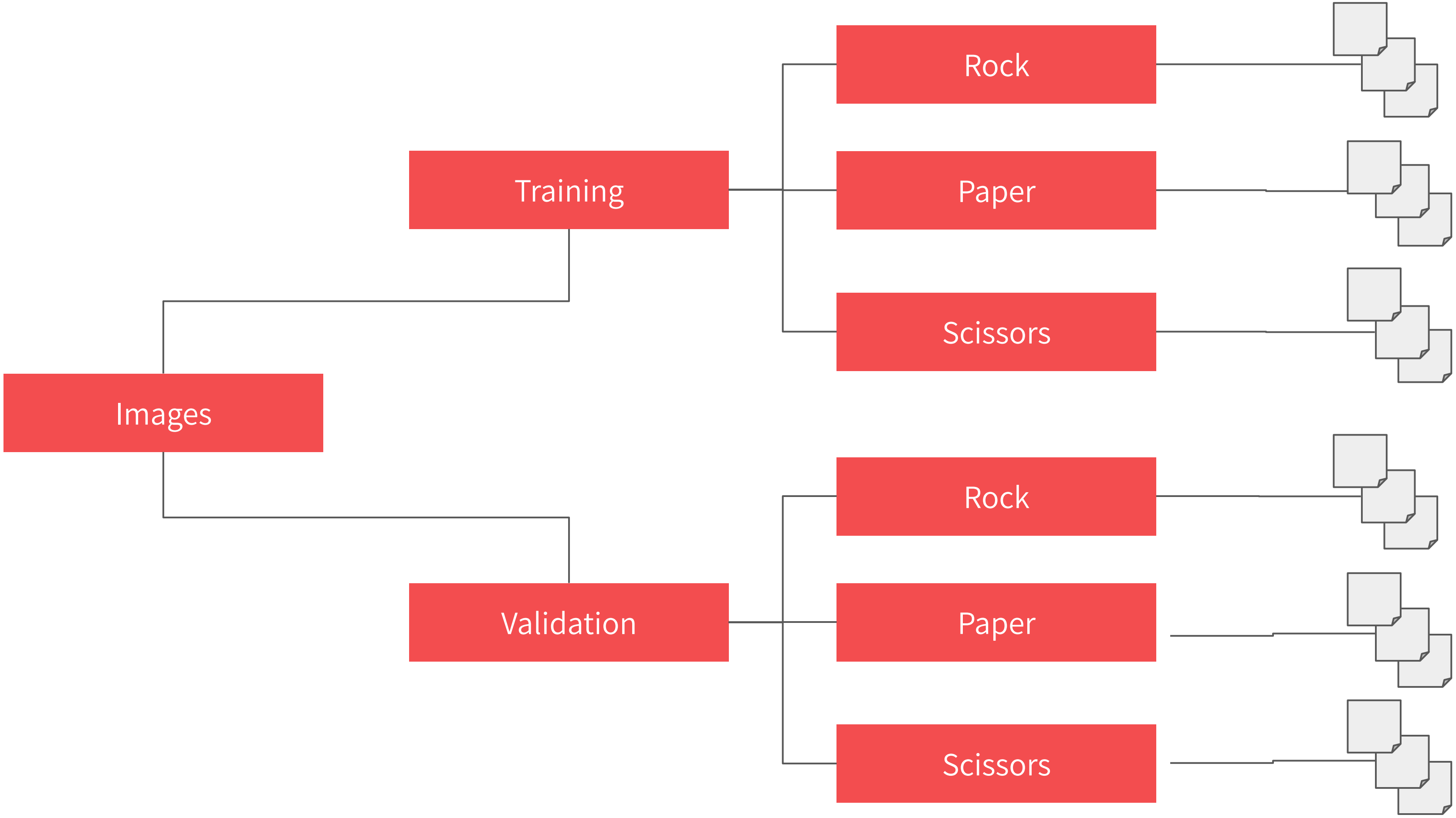
shergar.jpg

beauty.jpg

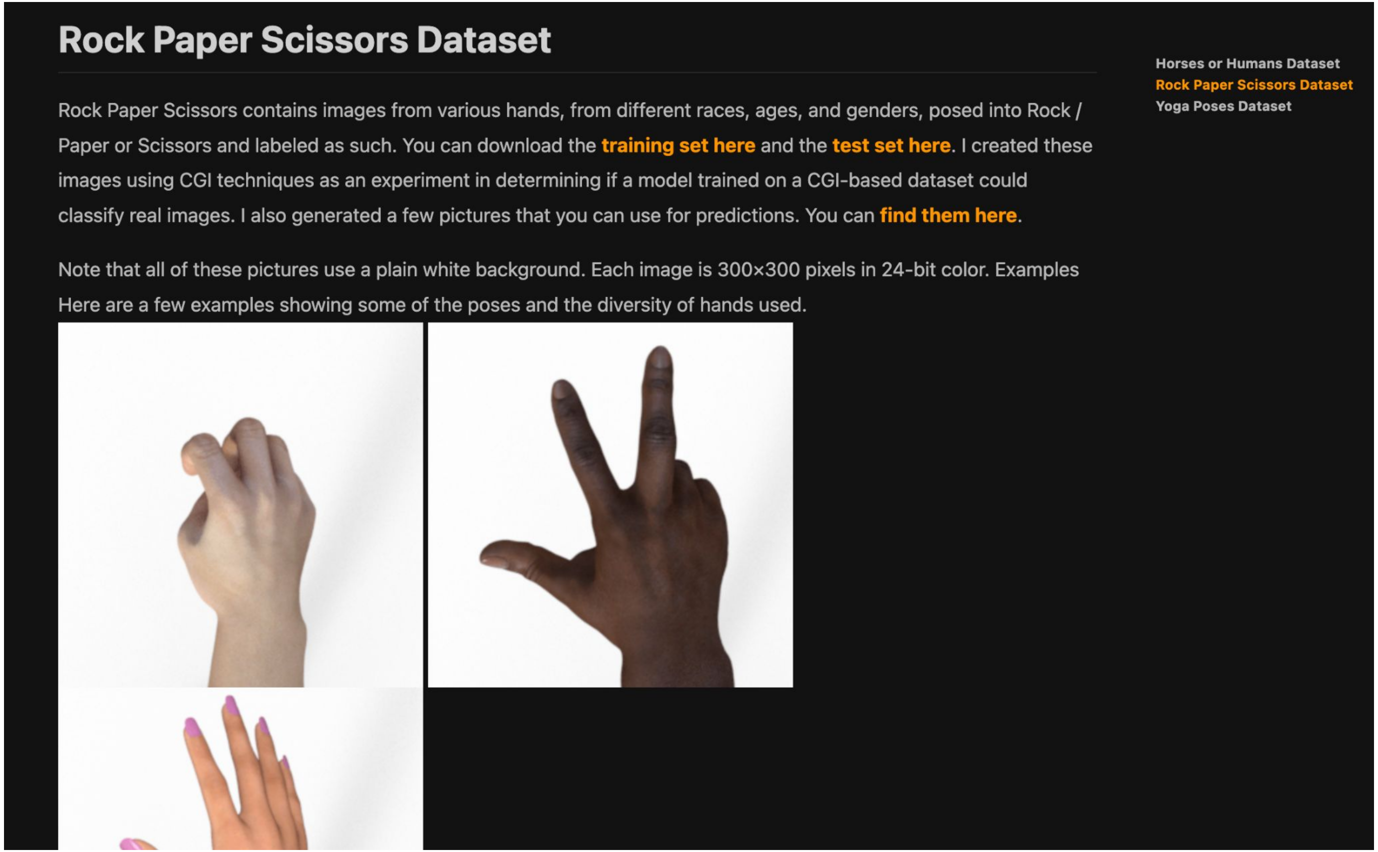
Humans

sharon.jpg

athena.jpg


Images
Training
Rock
Paper
Scissors
Validation
Rock
Paper
Scissors

https://laurencemoroney.com/datasets.html#rock-paper-scissors-dataset



Rock Paper Scissors Dataset

Rock Paper Scissors contains images from various hands, from different races, ages, and genders, posed into Rock / Paper or Scissors and labeled as such. You can download the **training set here** and the **test set here**. I created these images using CGI techniques as an experiment in determining if a model trained on a CGI-based dataset could classify real images. I also generated a few pictures that you can use for predictions. You can **find them here**.

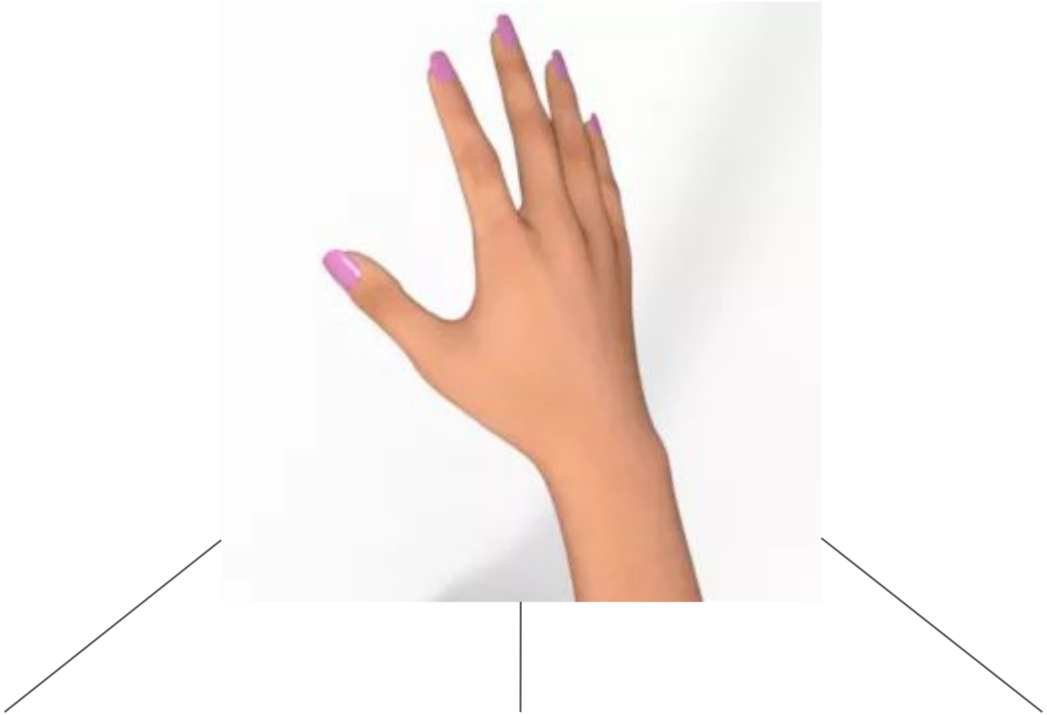
Note that all of these pictures use a plain white background. Each image is 300×300 pixels in 24-bit color. Examples Here are a few examples showing some of the poses and the diversity of hands used.

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='binary')
```

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=(150, 150),
    batch_size=20,
    label_mode='categorical')
```

```
model = tf.keras.models.Sequential([
    tf.keras.Input(input_shape=(150, 150, 3)),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    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')
])
```

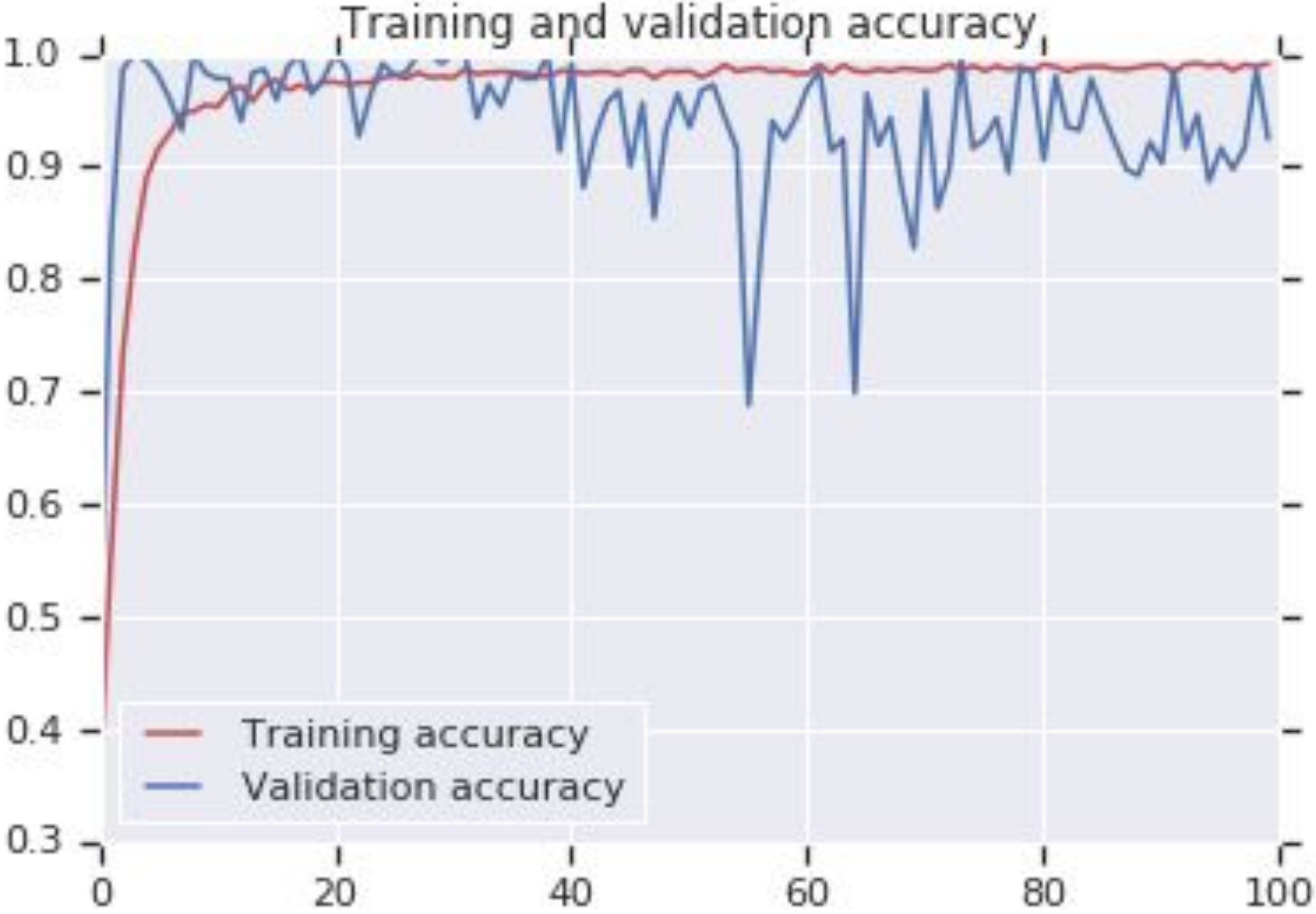
```
model = tf.keras.models.Sequential([
    tf.keras.Input(input_shape=(150, 150, 3)),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (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

```
model.compile(loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001),
    metrics=['accuracy'])
```

```
model.compile(loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001),
    metrics=['accuracy'])
```


Training and validation accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100
Training accuracy
Validation accuracy