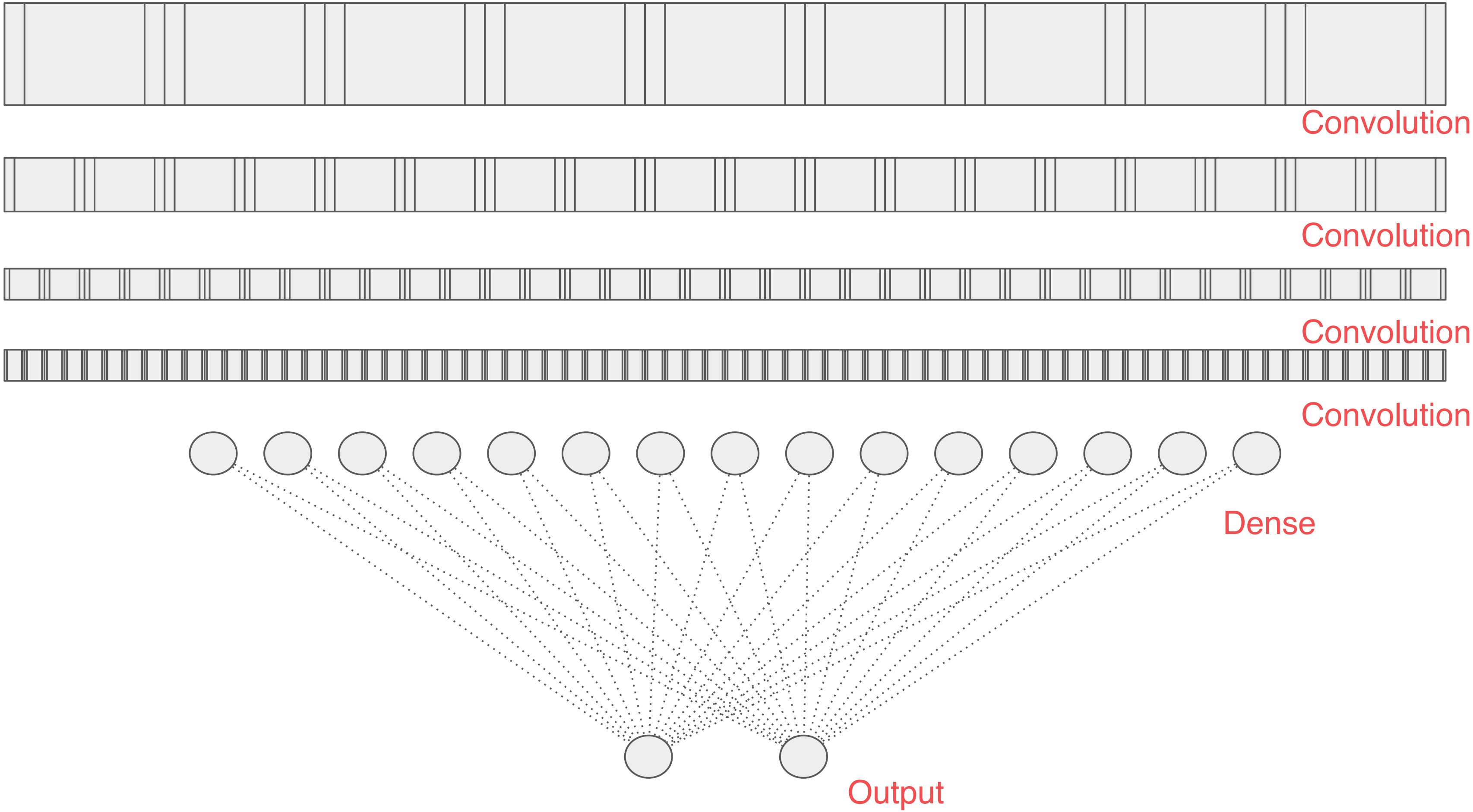
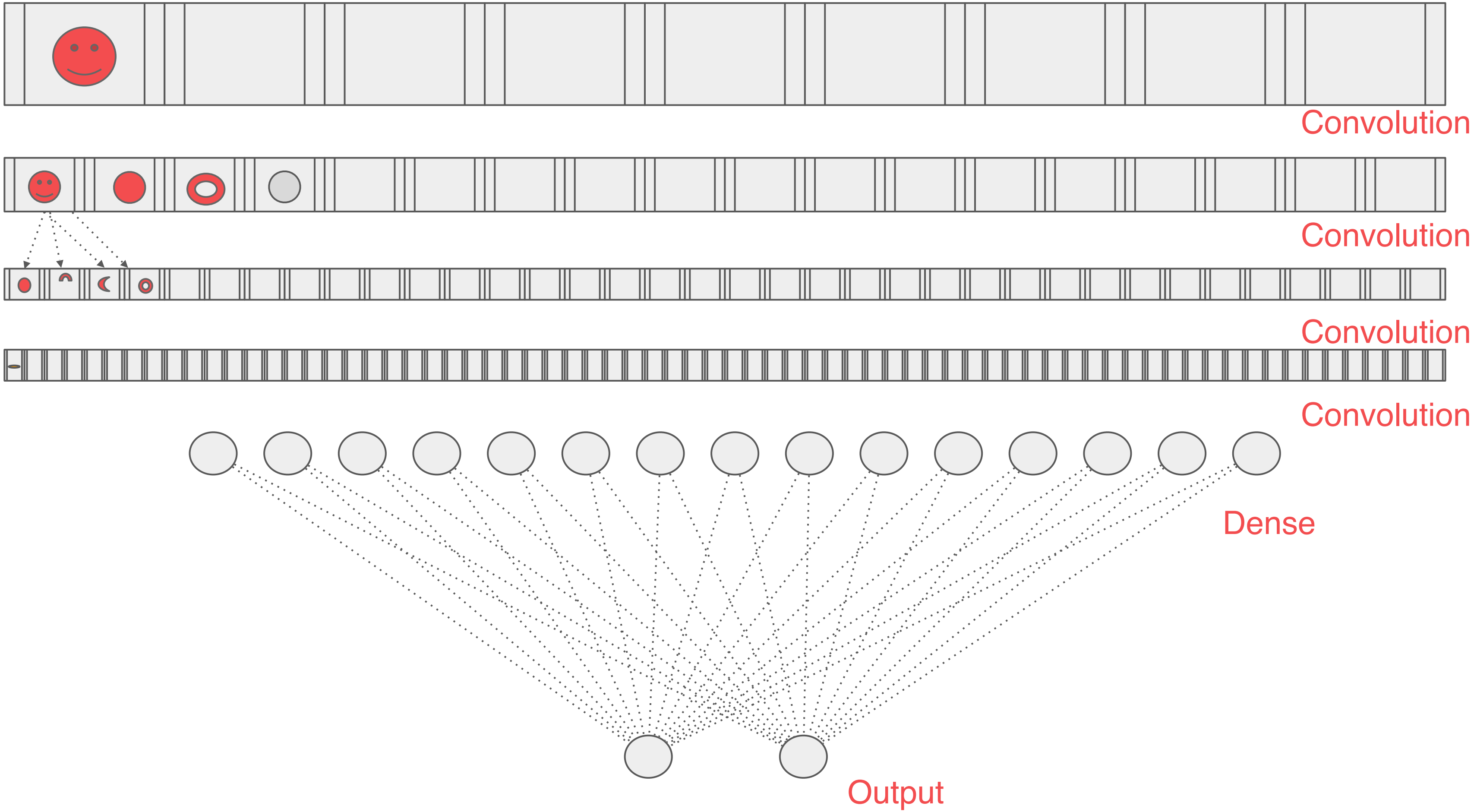
Copyright Notice

Convolution
Convolution
Convolution
Convolution
Dense
Output



Convolution

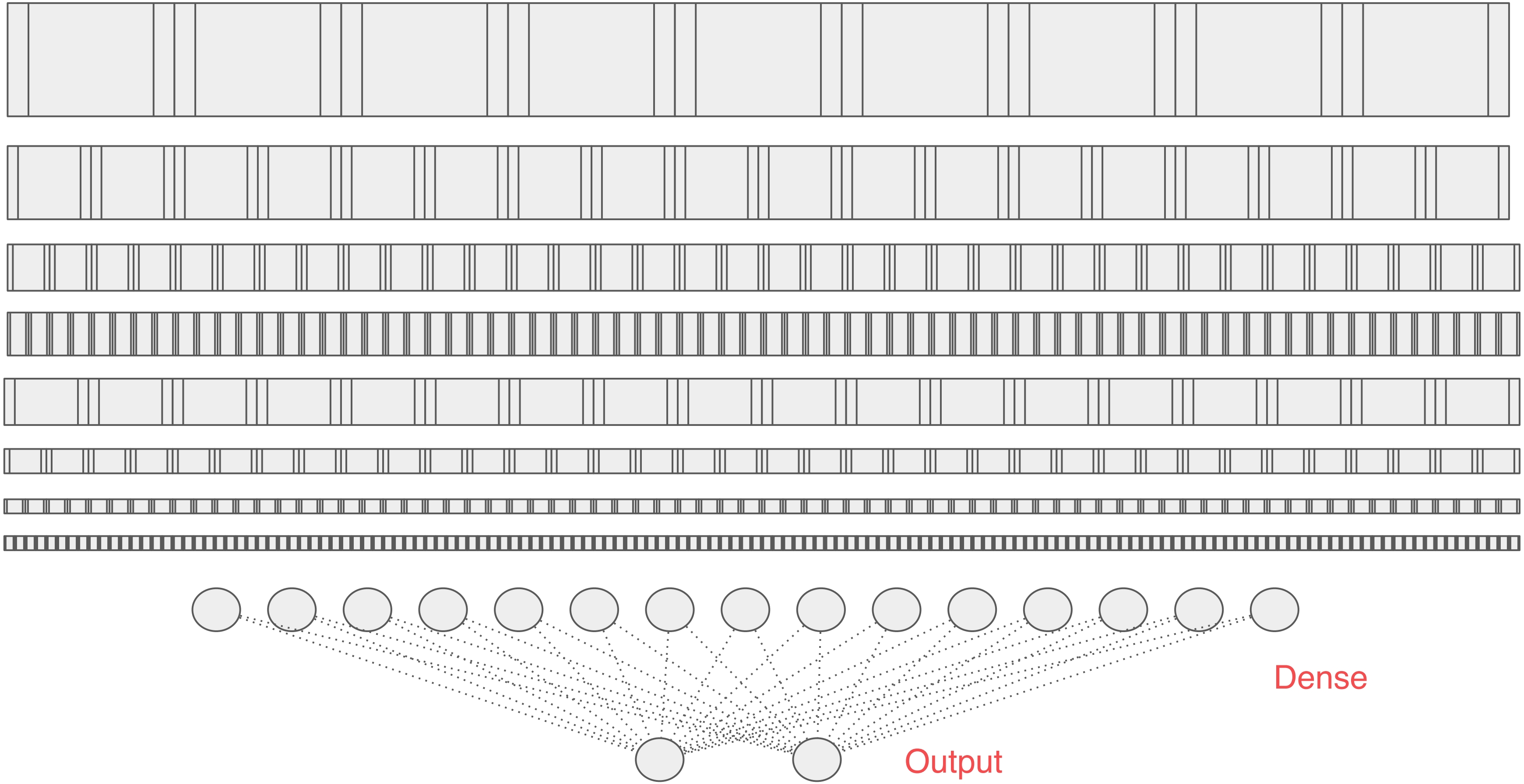
Convolution

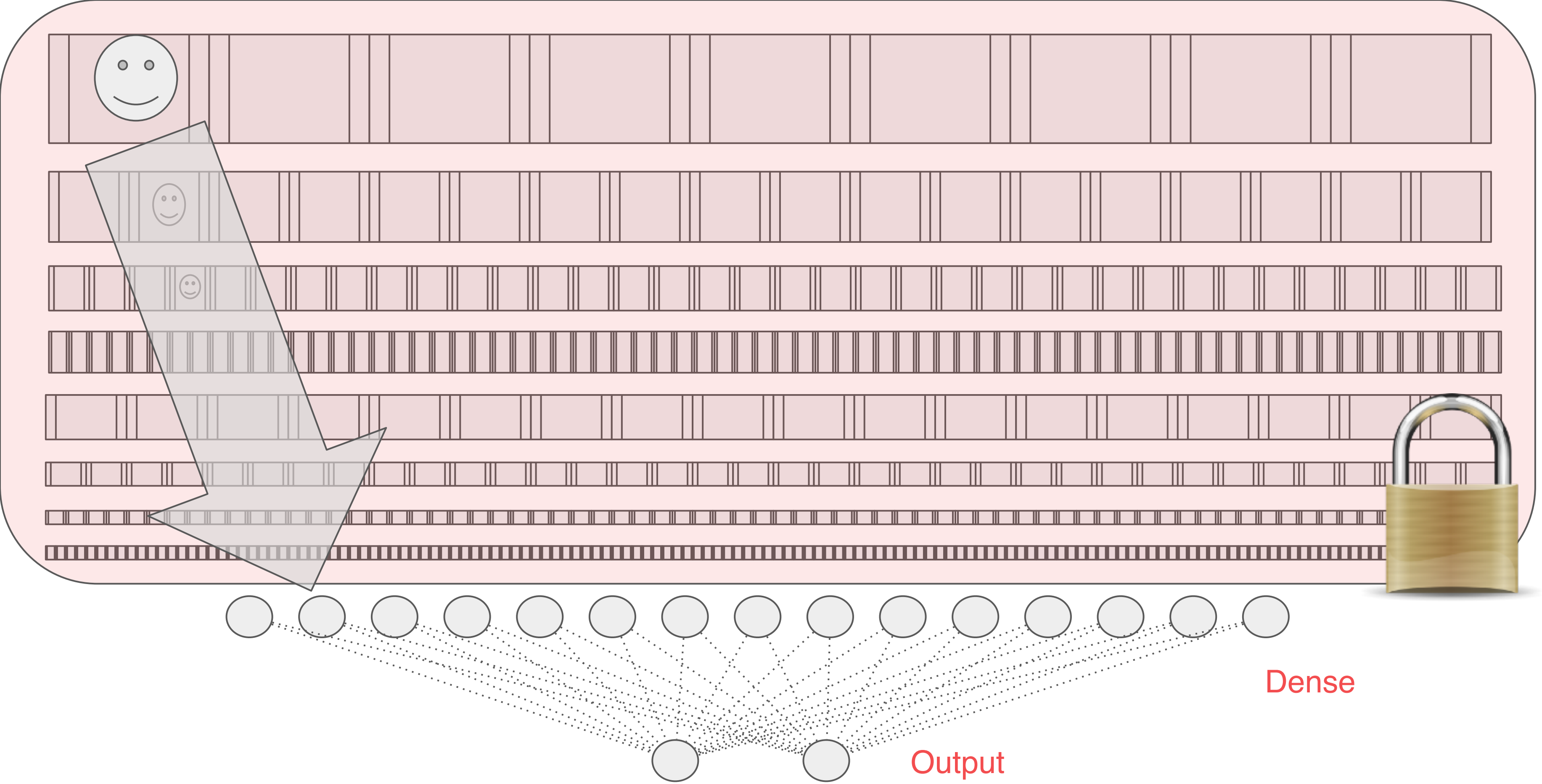
Convolution

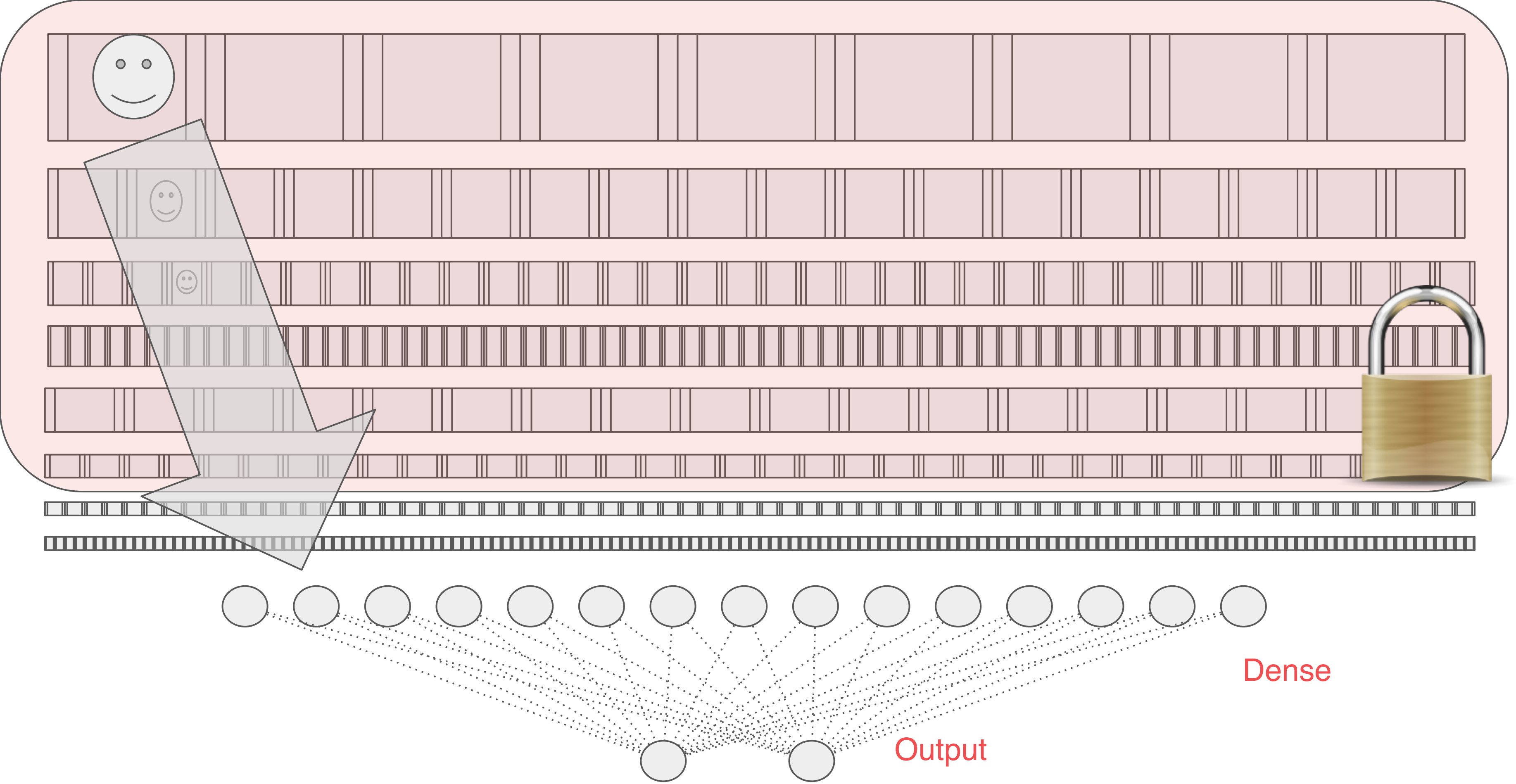
Convolution

Dense

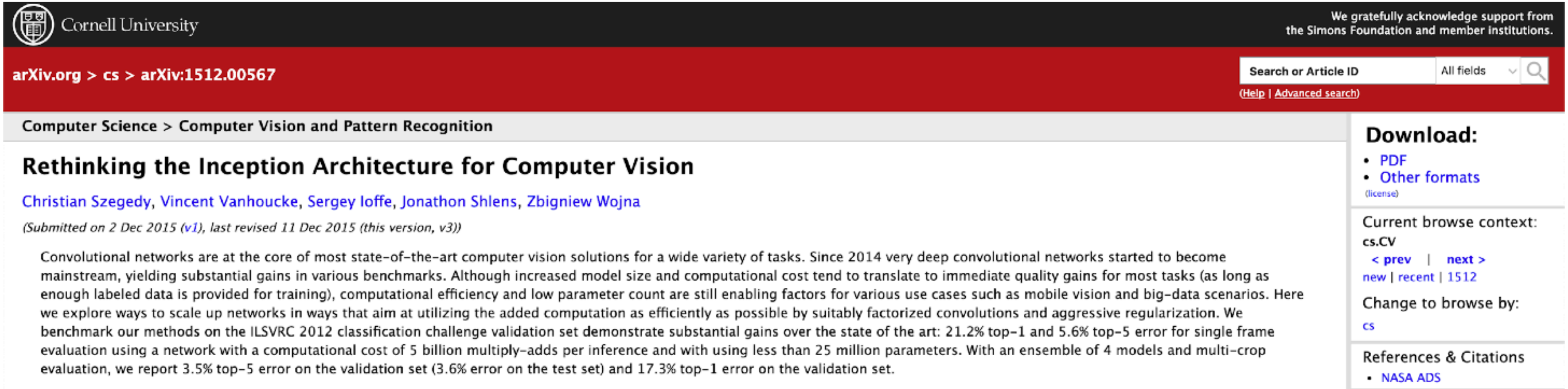
Output


Dense
Output


Dense
Output


Dense
Output

https://arxiv.org/abs/1512.00567



Computer Science > Computer Vision and Pattern Recognition

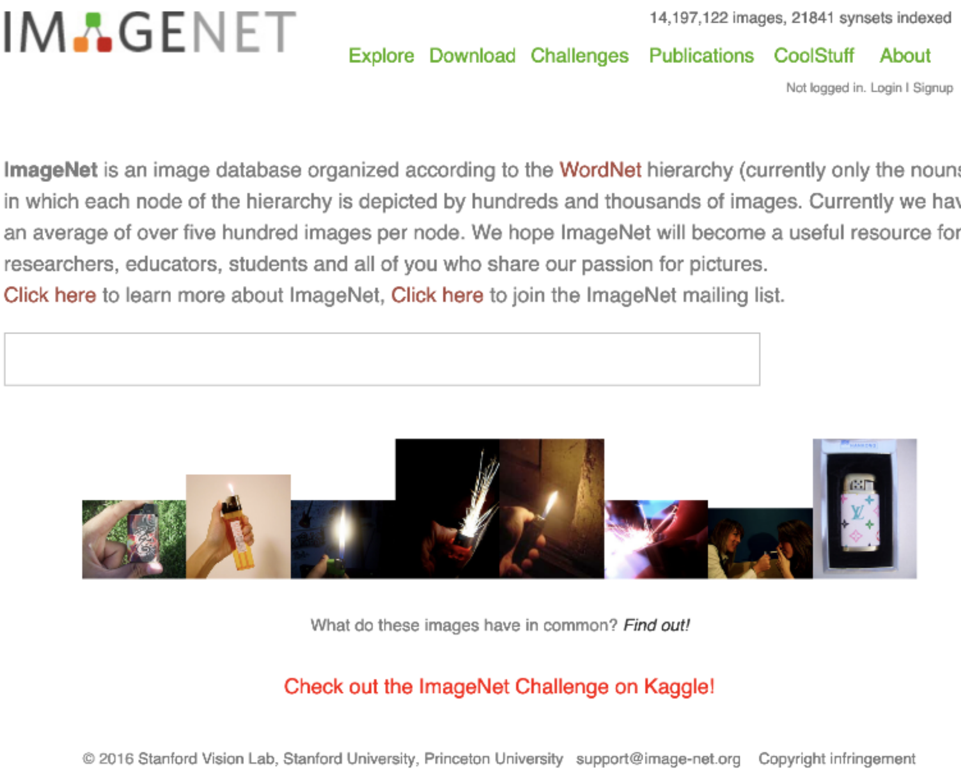
Rethinking the Inception Architecture for Computer Vision

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!

© 2016 Stanford Vision Lab, Stanford University, Princeton University support@image-net.org Copyright infringement

```
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

```
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
```

```python
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'])
```

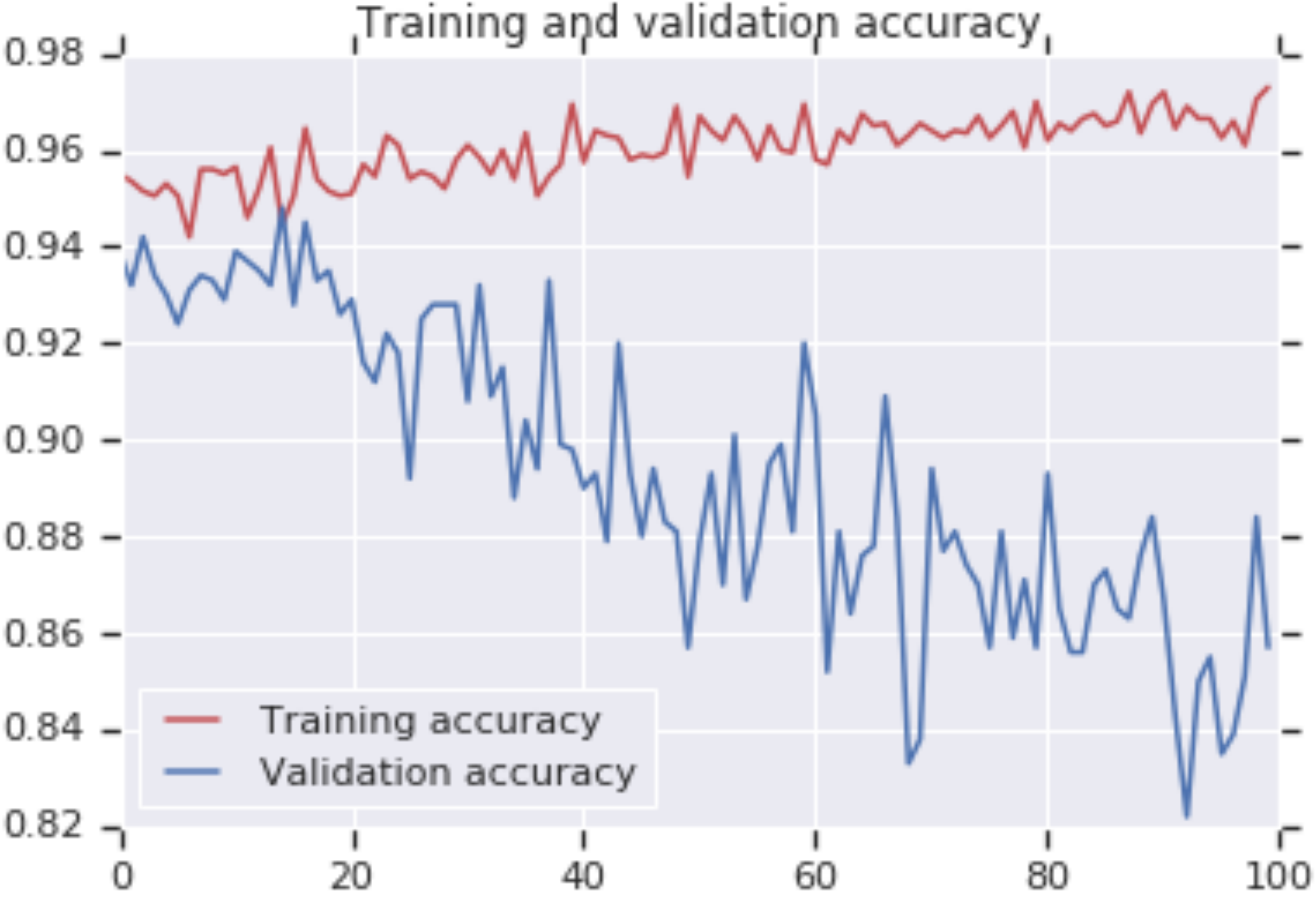
```
from tensorflow.keras.optimizers import RMSprop

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'])
```

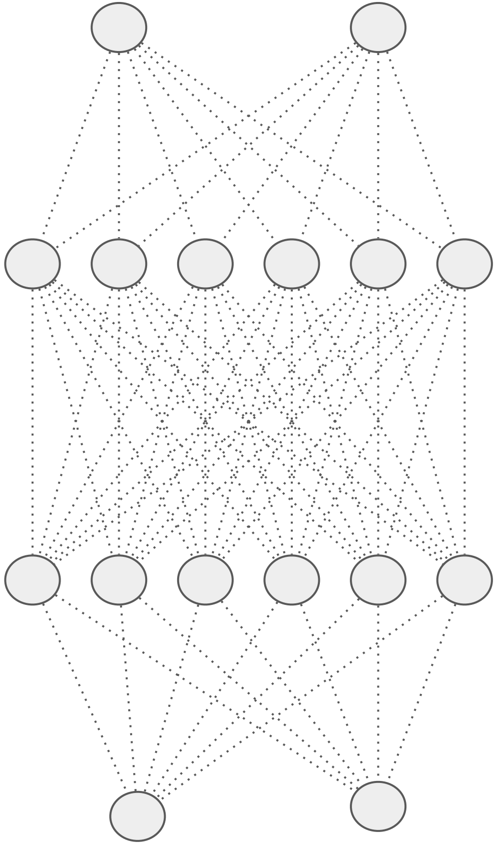
```
from tensorflow.keras.optimizers import RMSprop

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'])
```

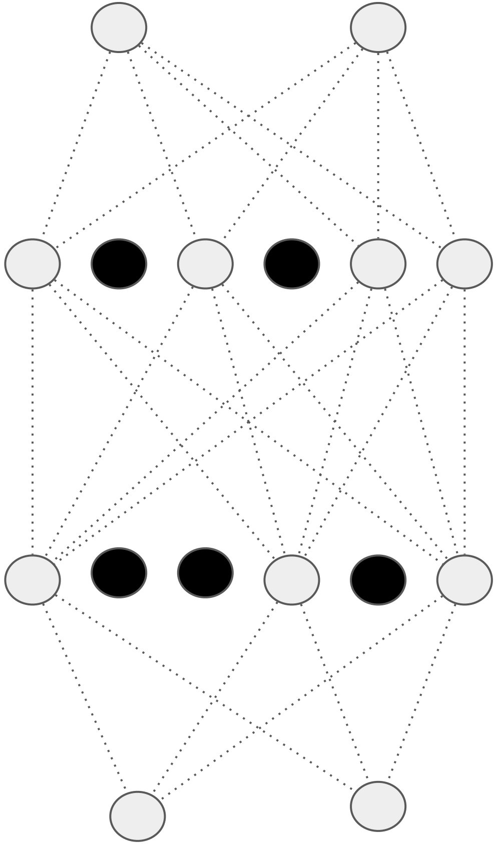
```
# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255.,
                    rotation_range = 40,
                    width_shift_range = 0.2,
                    height_shift_range = 0.2,
                    shear_range = 0.2,
                    zoom_range = 0.2,
                    horizontal_flip = True)
```

```
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)
```

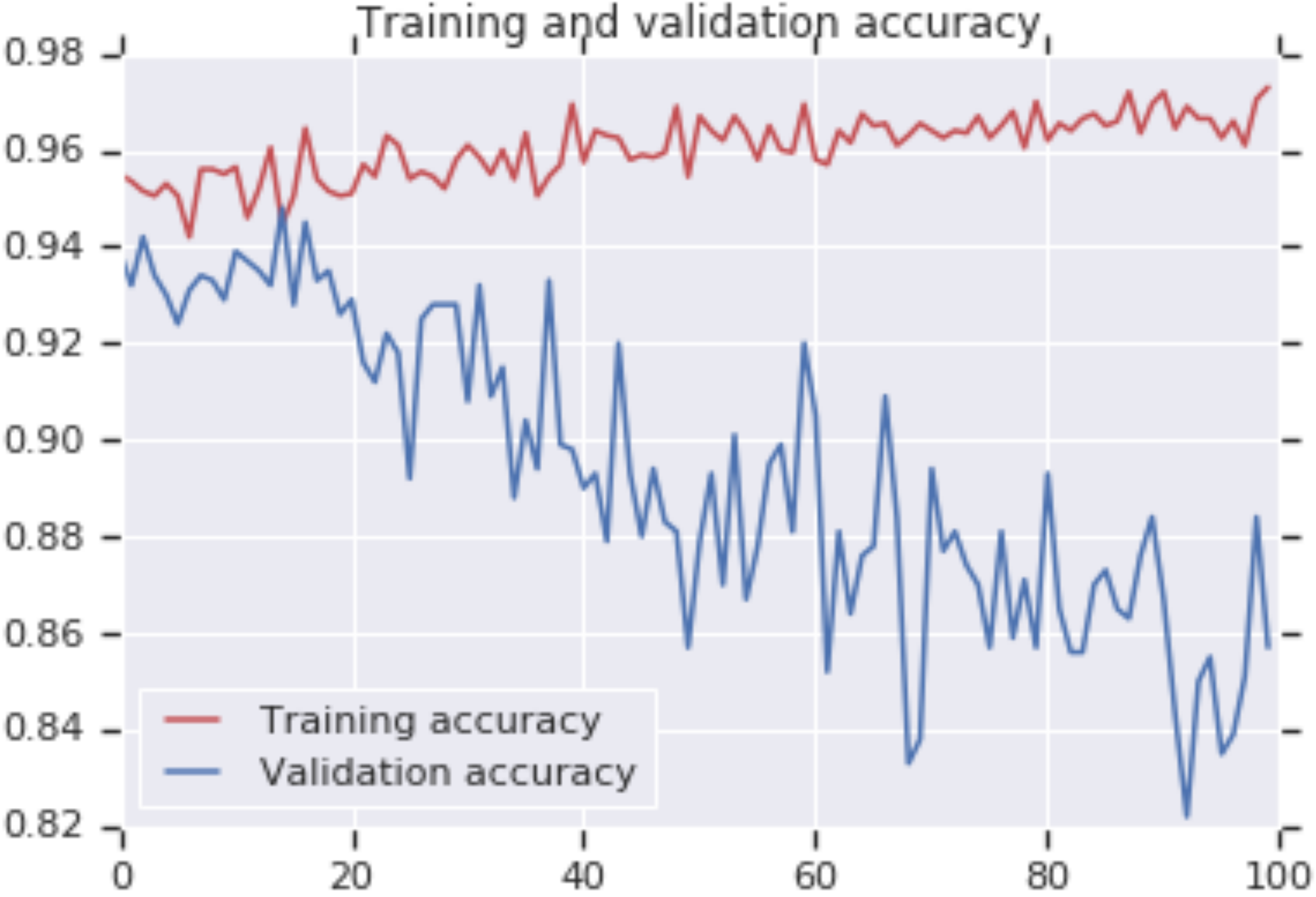

Training and validation accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0
20
40
60
80
100
Training accuracy
Validation accuracy

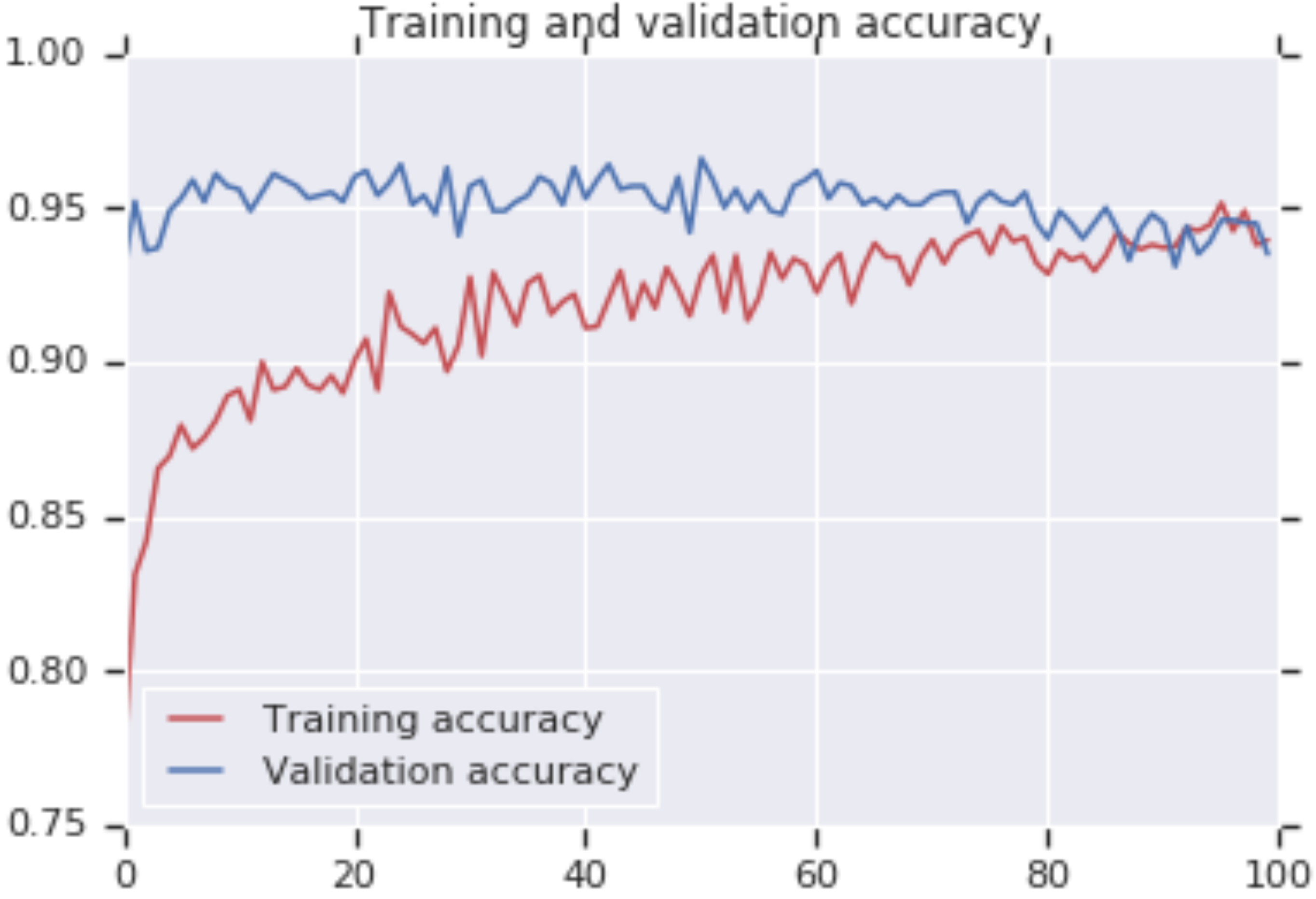




```python
from tensorflow.keras.optimizers import RMSprop

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
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