Convolutional Networks (ConvNets): Part I

October 27, 2022

ddebarr@uw.edu

http://cross-entropy.net/ml530/Deep_Learning_2.pdf

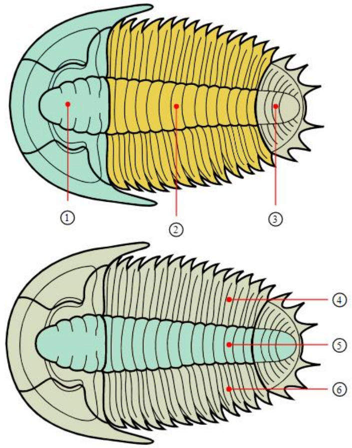
Agenda for Tonight

- Homework Review
- [DLI] Chapter 1: Biological and Machine Vision
- [DLI] Chapter 10: Machine Vision

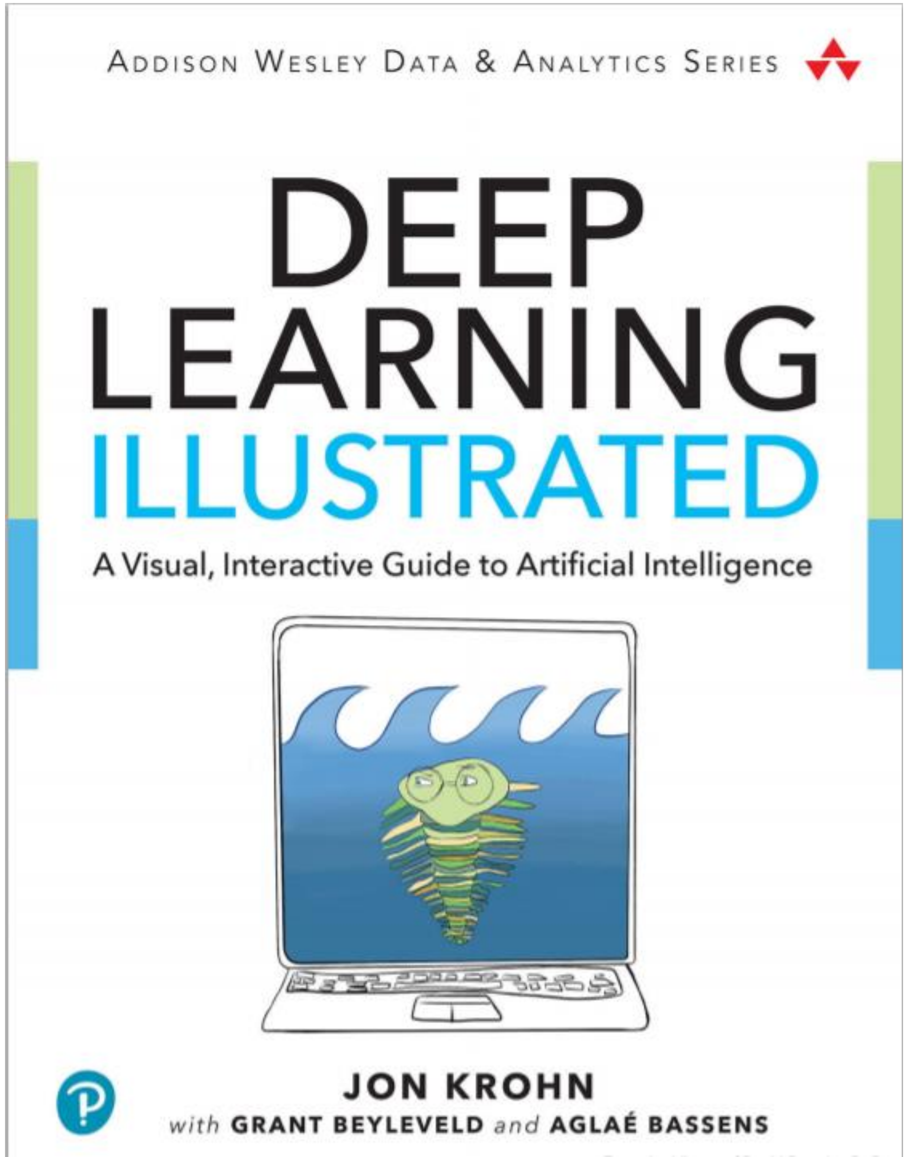
Textbook #1

The cover of our textbook contains the unofficial mascot of the book, a trilobite in deep waters ☺

The trilobite is a 3-section, 3-lobe marine arthropod that went extinct 250 million years ago



https://en.wikipedia.org/wiki/Trilobite



Addison Wesley Data & Analytics Series

DEEP LEARNING ILLUSTRATED

A Visual, Interactive Guide to Artificial Intelligence

JON KROHN

with GRANT BEYLEVELD and AGLAÉ BASSENS

[DLI] Chapter 1: Biological and Machine Vision

- Biological Vision
- Machine Vision
  - The Neocognitron
  - LeNet-5
  - The Traditional Machine Learning Approach
  - ImageNet and the ILSVRC
  - AlexNet
- TensorFlow Playground
- Quick, Draw!
- Summary

Authors

- Jon Krohn: chief data scientist at untapt.com

- Grant Beyleveld: data scientist at untapt.com

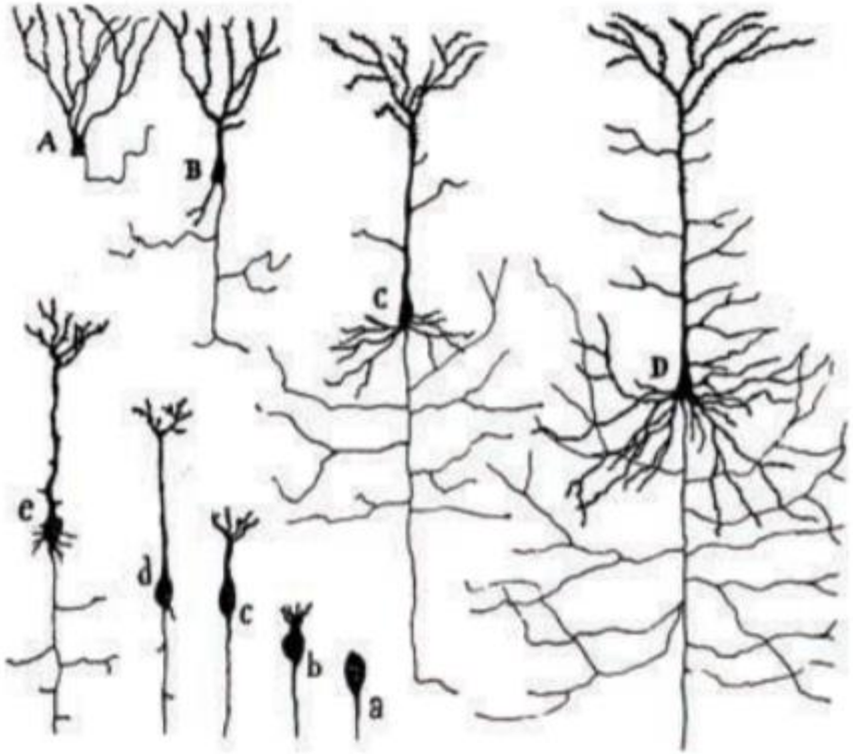
- Aglae Bassens: Belgian artist based in Paris

Santiago Cajal: 1852 - 1934

Spanish physician who was the first to identify neurons, by examining thin slices of brain tissue

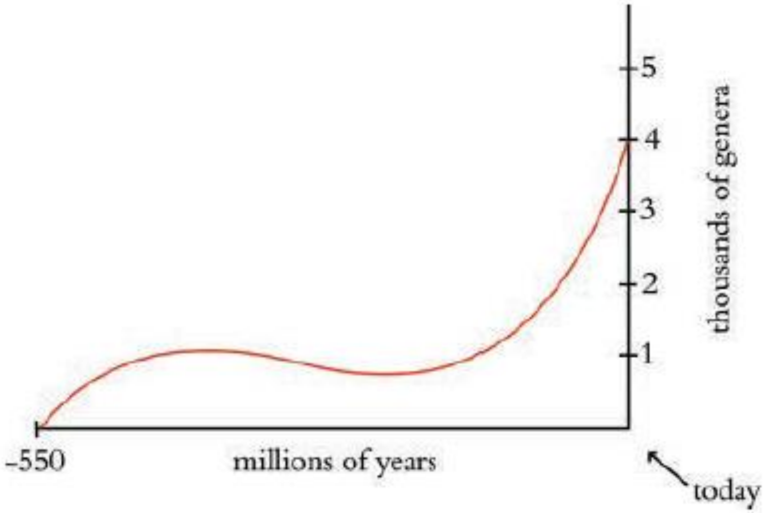
Hand-Drawn Neurons from Cajal (1894)

- (a)-(e) illustrates the growth of a neuron
- (A)-(D) contrasts the neurons of a frog, lizard, rat, and human respectively



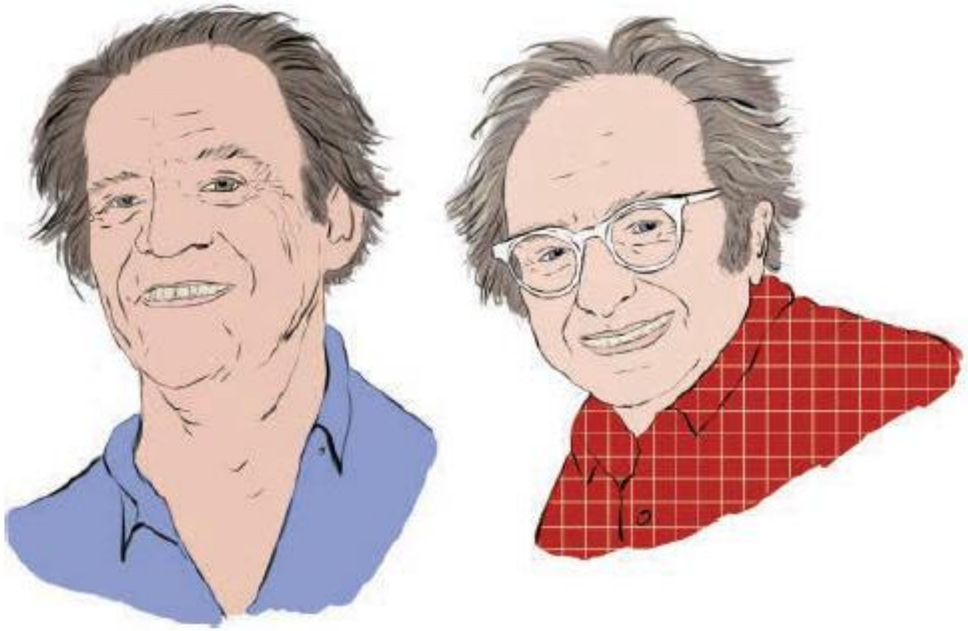
Surge in the Number of Species

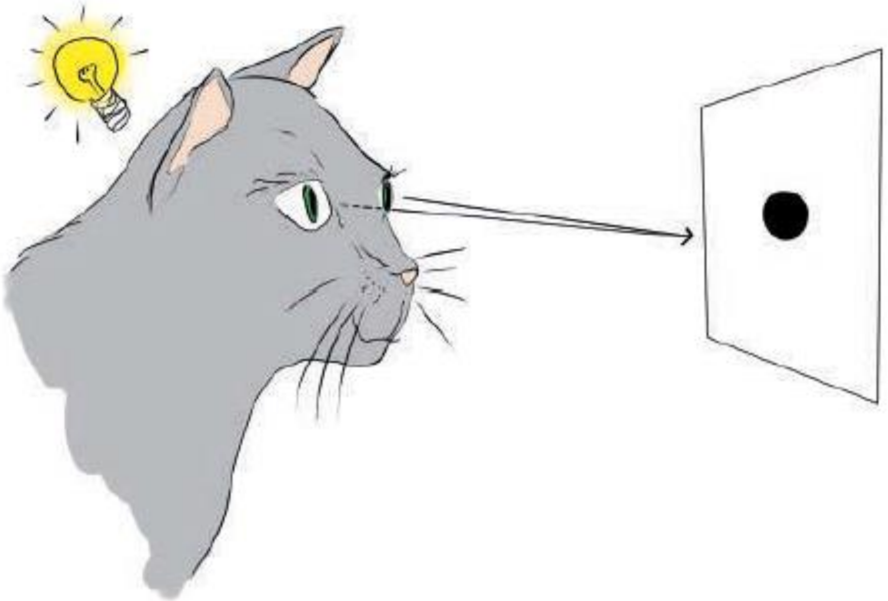
Evidence suggests this surge was driven by the development of light detectors in the trilobyte



Torsten Wiesel and David Hubel

- Nobel prize-winning neurophysiologists
- Implanted electrical recording equipment within cat skulls, to measure activity in the primary visual cortex when projected presenting slides to anesthetized cats (slide edges elicited response)



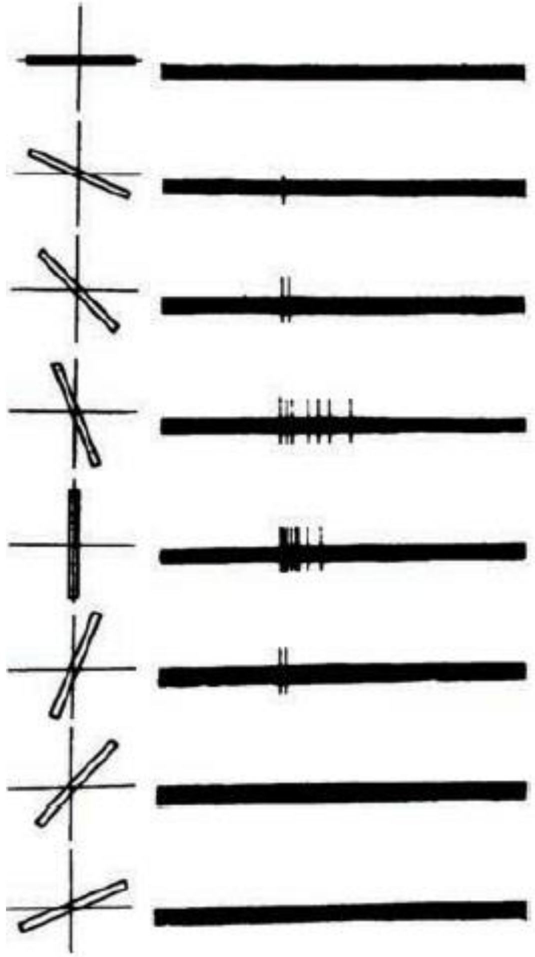


Cell Responding to Line Orientation

- Orientation of the line on the left

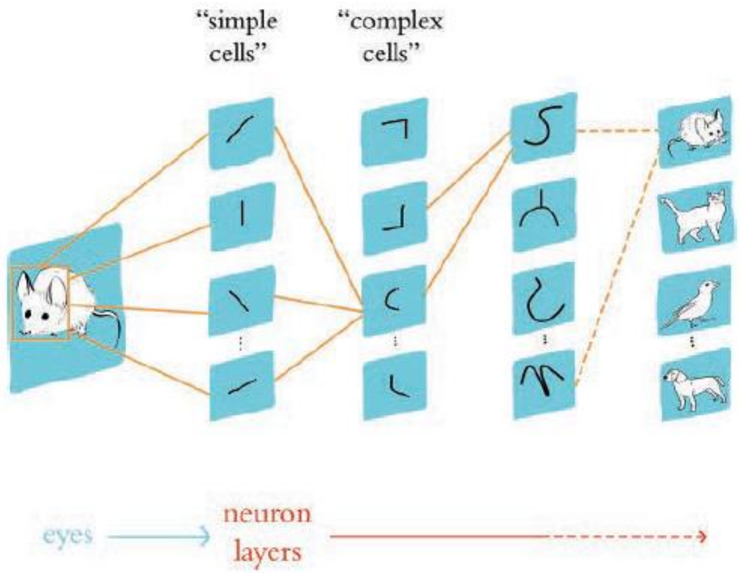
- Electrical activity over a second on the right

- Vertical line (5th row) generates the largest response



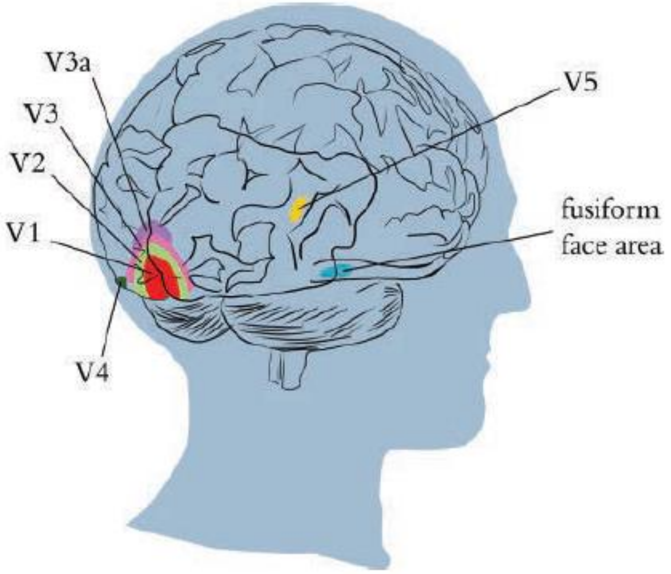
Consecutive Layers of Biological Neurons

- Cartoon has 5 layers: an input layer, 3 hidden layers, and an output layer
- First hidden layer neurons “fire” in response to simple concepts; e.g. edges
- Second and third layer neurons fire in response to successively more complex concepts; e.g. shapes and textures



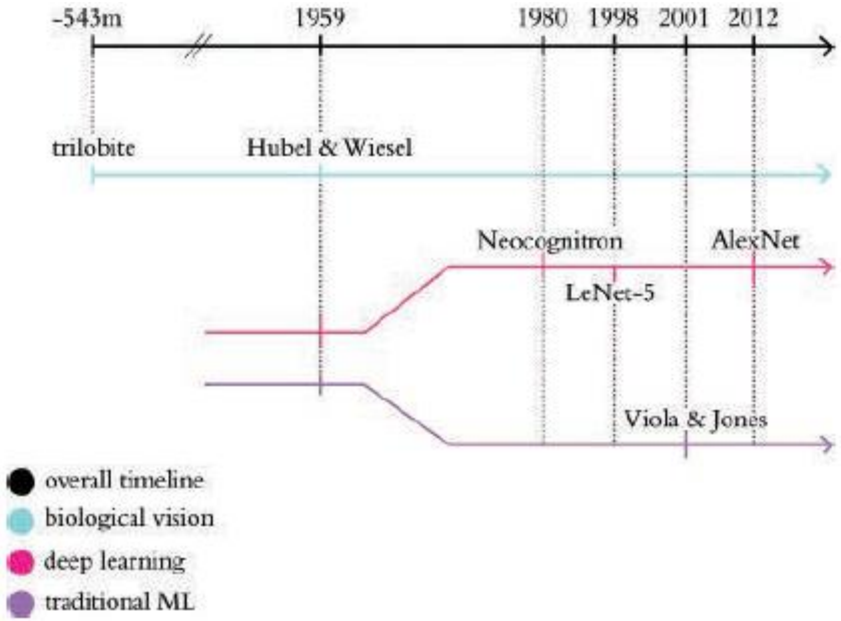
Regions of the Visual Cortex

- V1: contains simple cells that receive input from the eyes and detect edge orientations
- V2, V3, and V3a: increasingly complex, abstract concepts detected
- V4 specializes in detection of color
- V5 specializes in detection of motion
- Fusiform (spindle shaped) face area specializes in detection of faces



Computer Vision Timeline

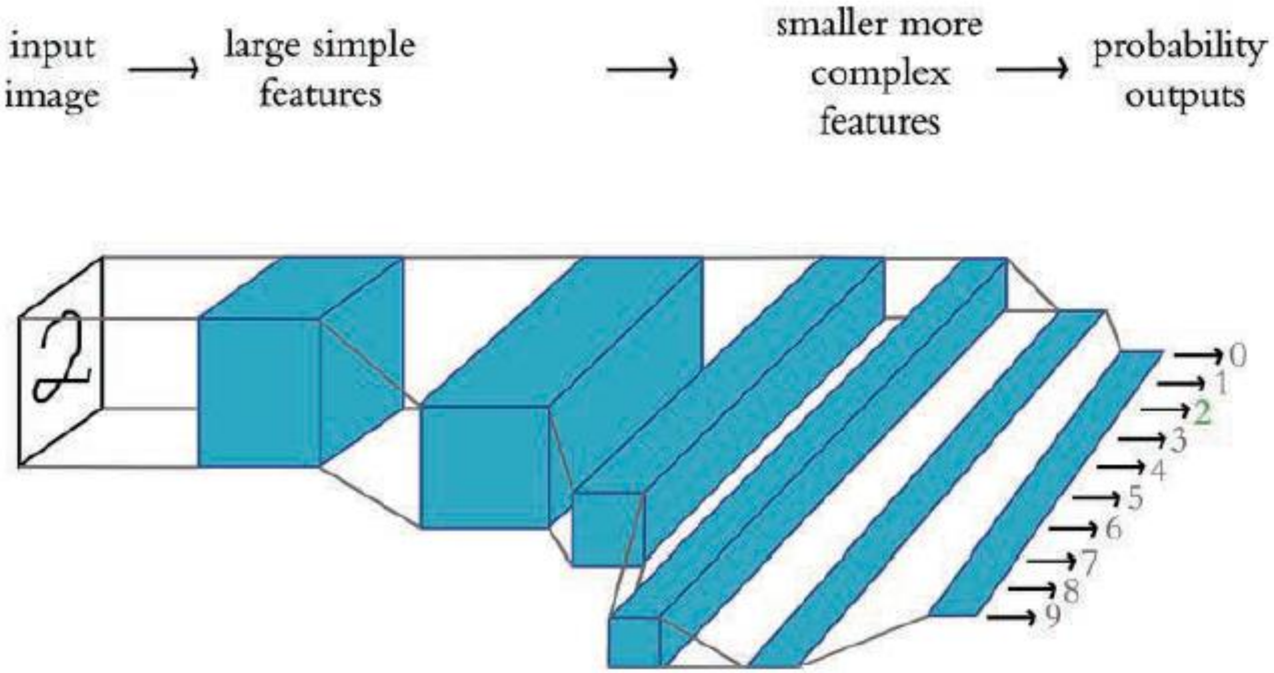
Light detectors; experiments on cats; handwritten digit detection (simple and complex cells; convolution and pooling); face detection (Haar features and adaptive boosting); image classification … something for everyone



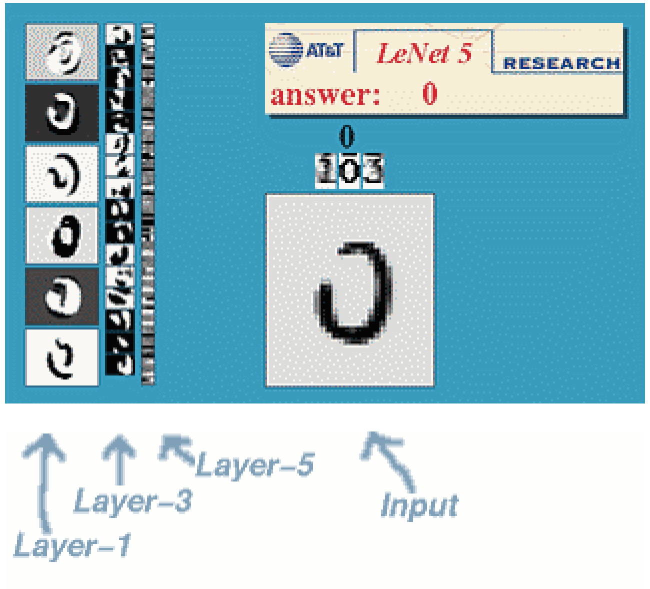
Yann LeCun and Yoshua Bengio

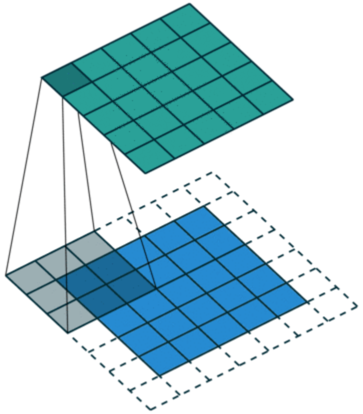
- Developed LeNet-5: first convolutional neural network; more accurate and more efficient for handwritten digit detection, compared to the Neocognitron
- Yann is Chief AI Scientist at Facebook
- Yoshua is a professor at the University of Montreal [Theano]

LeNet-5



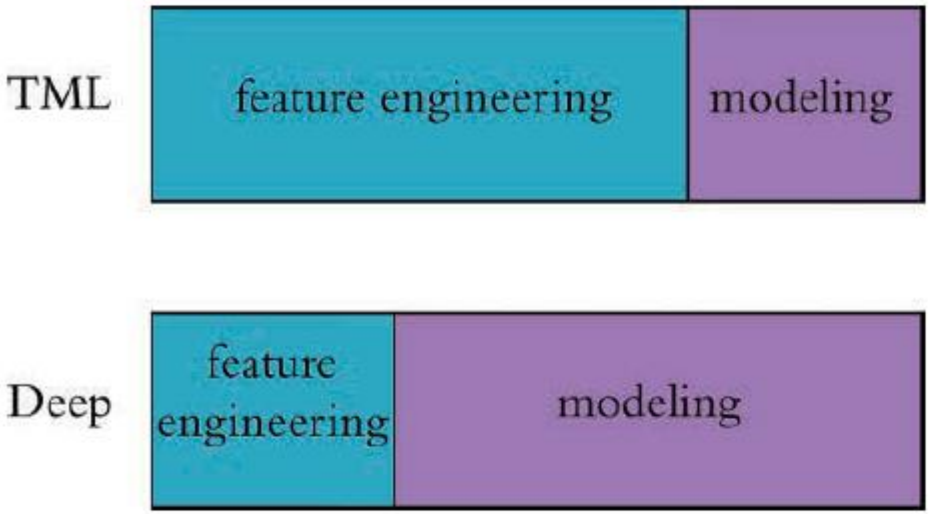
http://yann.lecun.com/exdb/lenet/





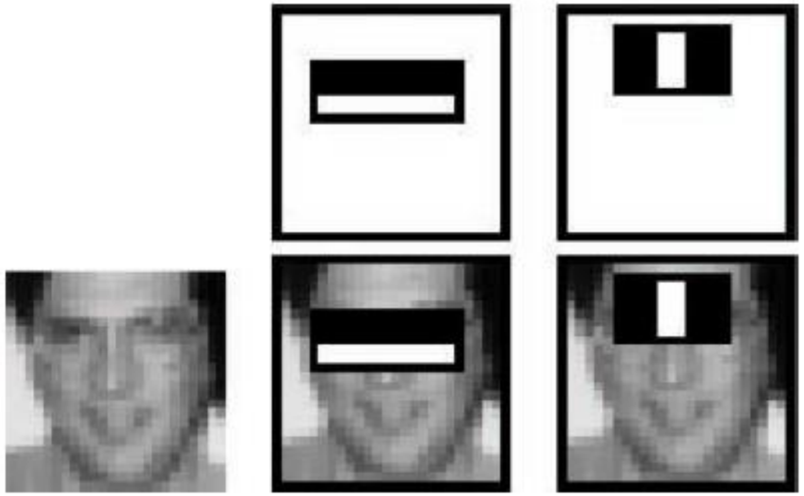
http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

Traditional Machine Learning versus Deep Learning



Viola Jones Face Detection

- Haar features used to detect regions, such as the eyes or the bridge of the nose
  - Eye region darker than the upper cheeks
  - Nose bridge region is brighter than the eyes
- Adaptive Boosting (AdaBoost) used for classification

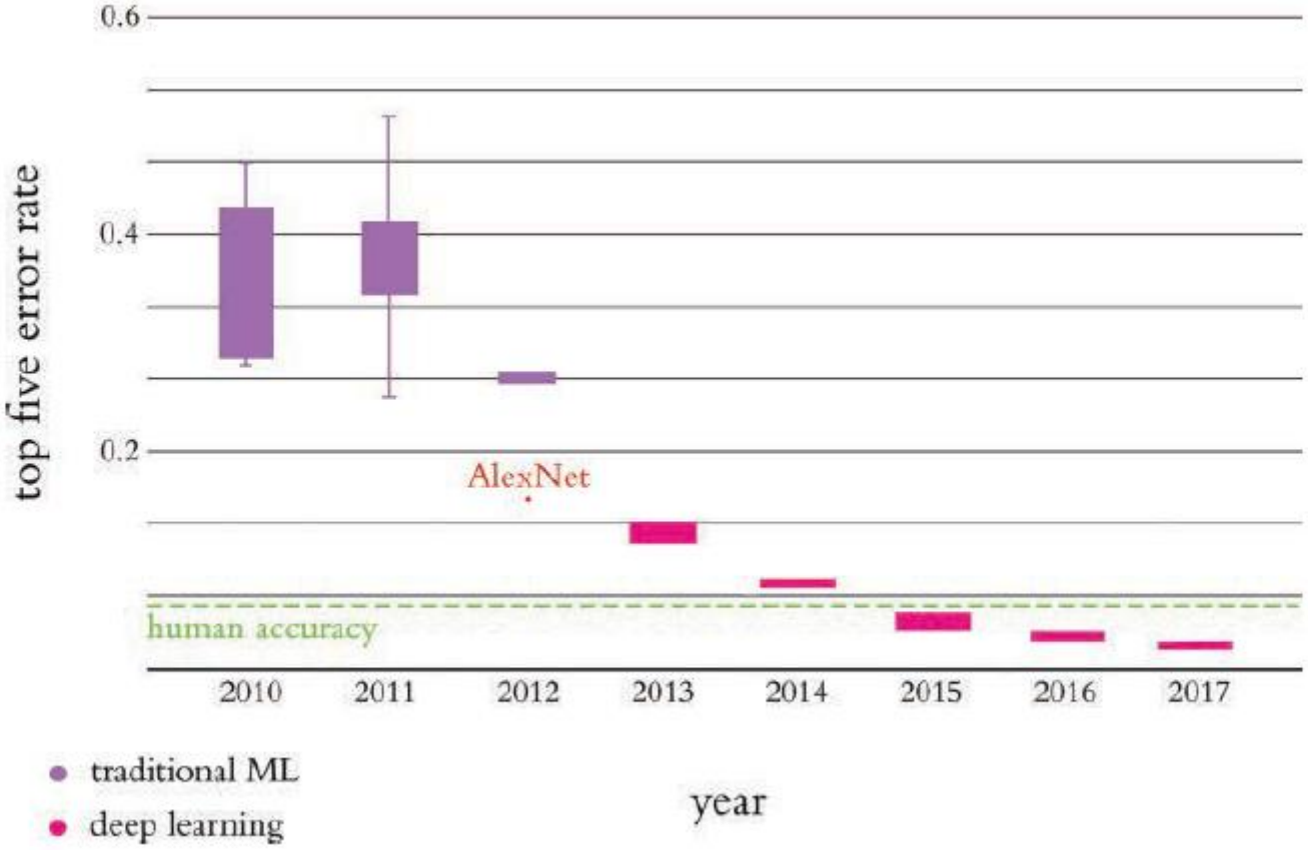


ILSVRC

- ImageNet dataset
  - Collected by Fei Fei Li and her colleagues at Princeton
  - 14 million images; 22,000 classes
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset
  - 1.4 million images
  - 1,000 classes
  - Used for both image classification (image contains a person) and object detection (bounding box contains a person)

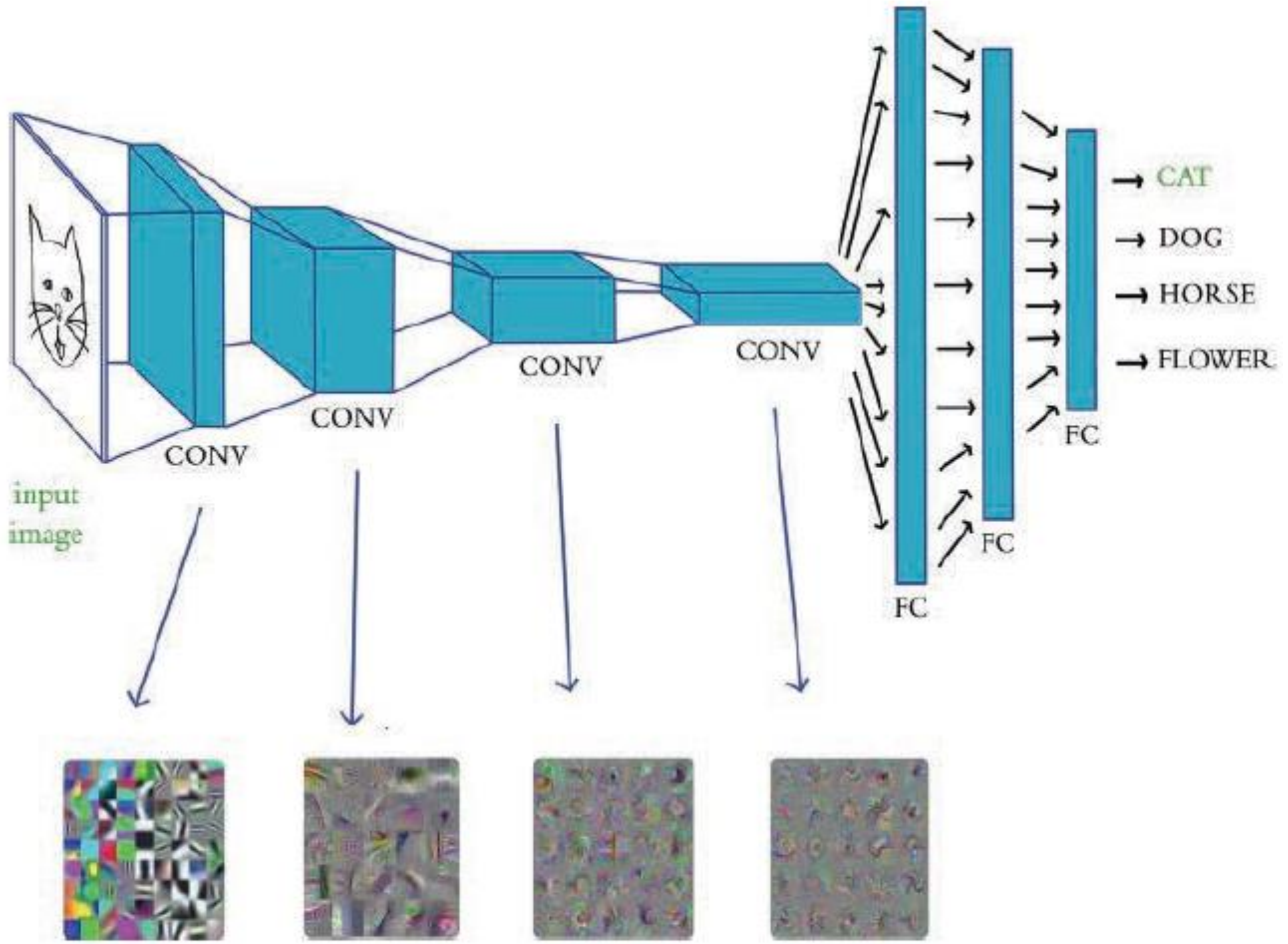
ILSVRC Performance

AlexNet (a convolutional neural network trained by Alex Krizhevsky, a student of Geoff Hinton) achieved a top-5 error rate of 15.3%



AlexNet Architecture

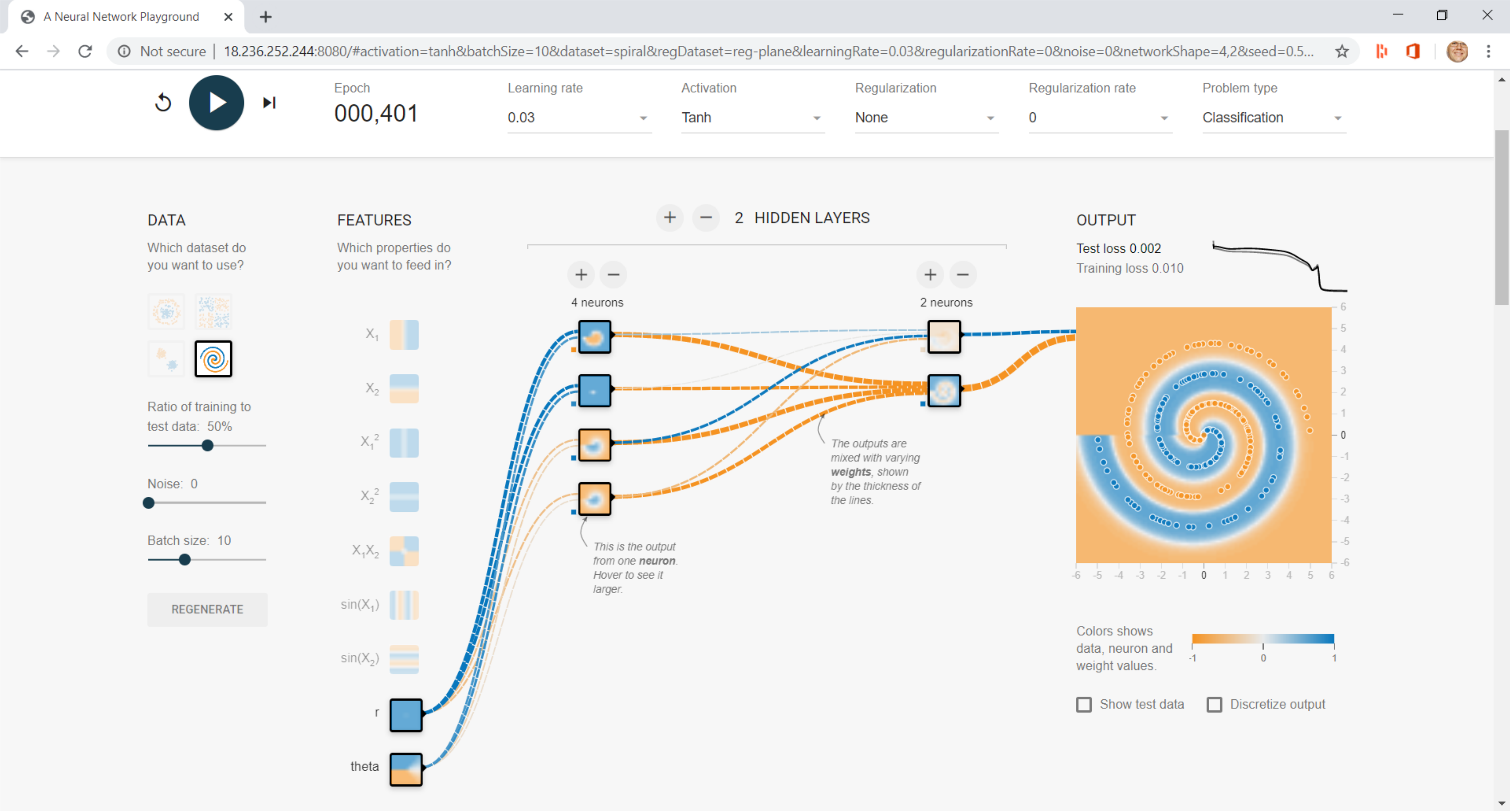
Convolution blocks followed by fully connected (dense) layers



Customizing Tensorflow Playground

```
git clone https://github.com/tensorflow/playground
cd playground/src
nano plaground.ts
# add below sinY
  "r": {f: (x, y) => Math.sqrt(x * x + y * y), label: "r"},
  "theta": {f: (x, y) => Math.atan2(y, x), label: "theta"},
nano state.ts
# add below first occurrence of sinY
    {name: "r", type: Type.BOOLEAN},
    {name: "theta", type: Type.BOOLEAN},
# add below second occurrence of sinY
  r = false;
  theta=false;
cd ..
sudo apt install npm
npm i
npm run build
npm run serve
```

Spiral Classification with Polar Coordinates



A Neural Network Playground

Not secure | 18.236.252.244:8080/#activation=tanh&batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.5...

Epoch 000,401

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

This is the output from one ***neuron****. Hover to see it larger.*

The outputs are mixed with varying ***weights****, shown by the thickness of the lines.*

OUTPUT

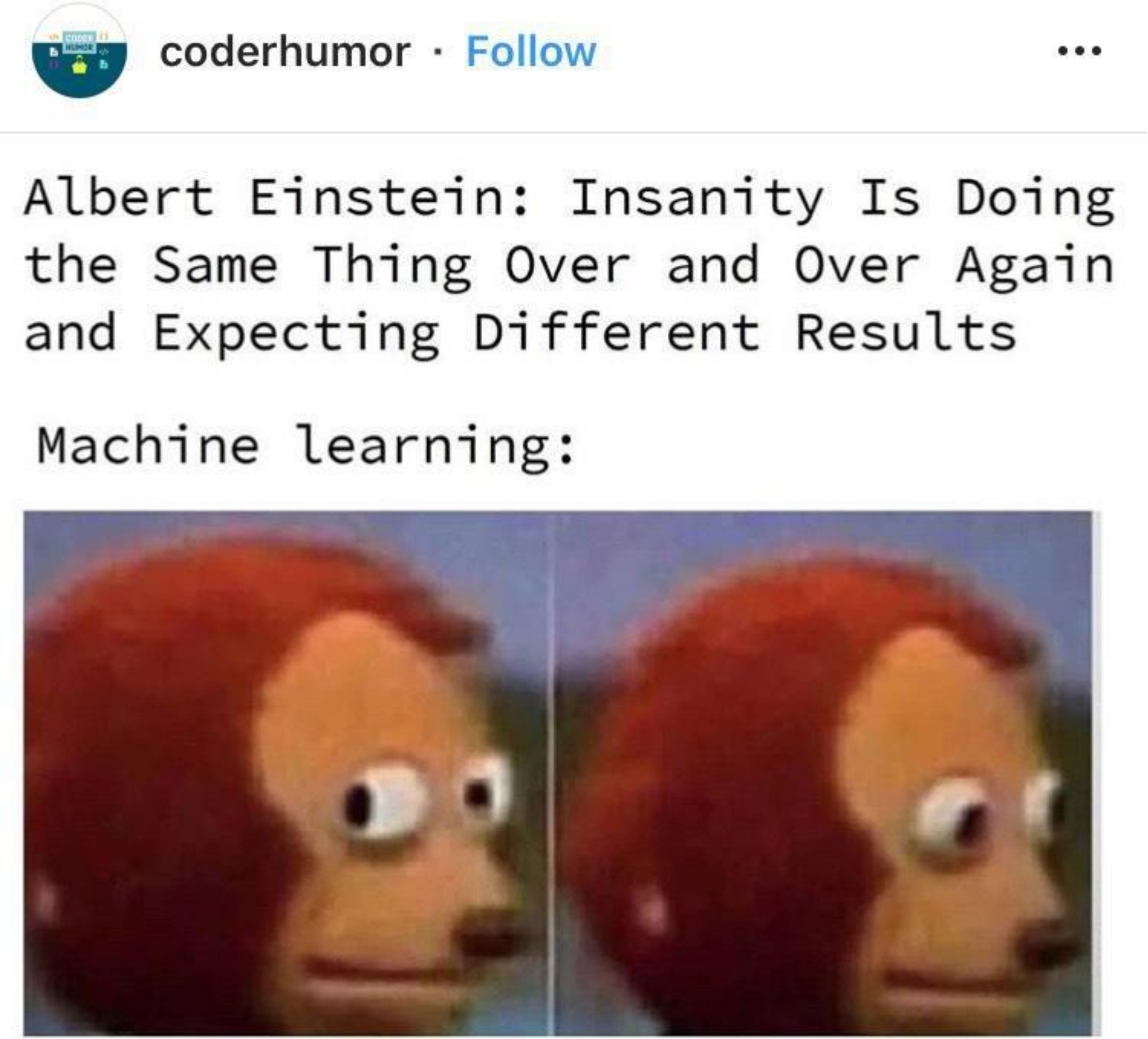
Test loss 0.002

Training loss 0.010

Colors shows data, neuron and weight values.

□ Show test data □ Discretize output

Reminder About Repeatability

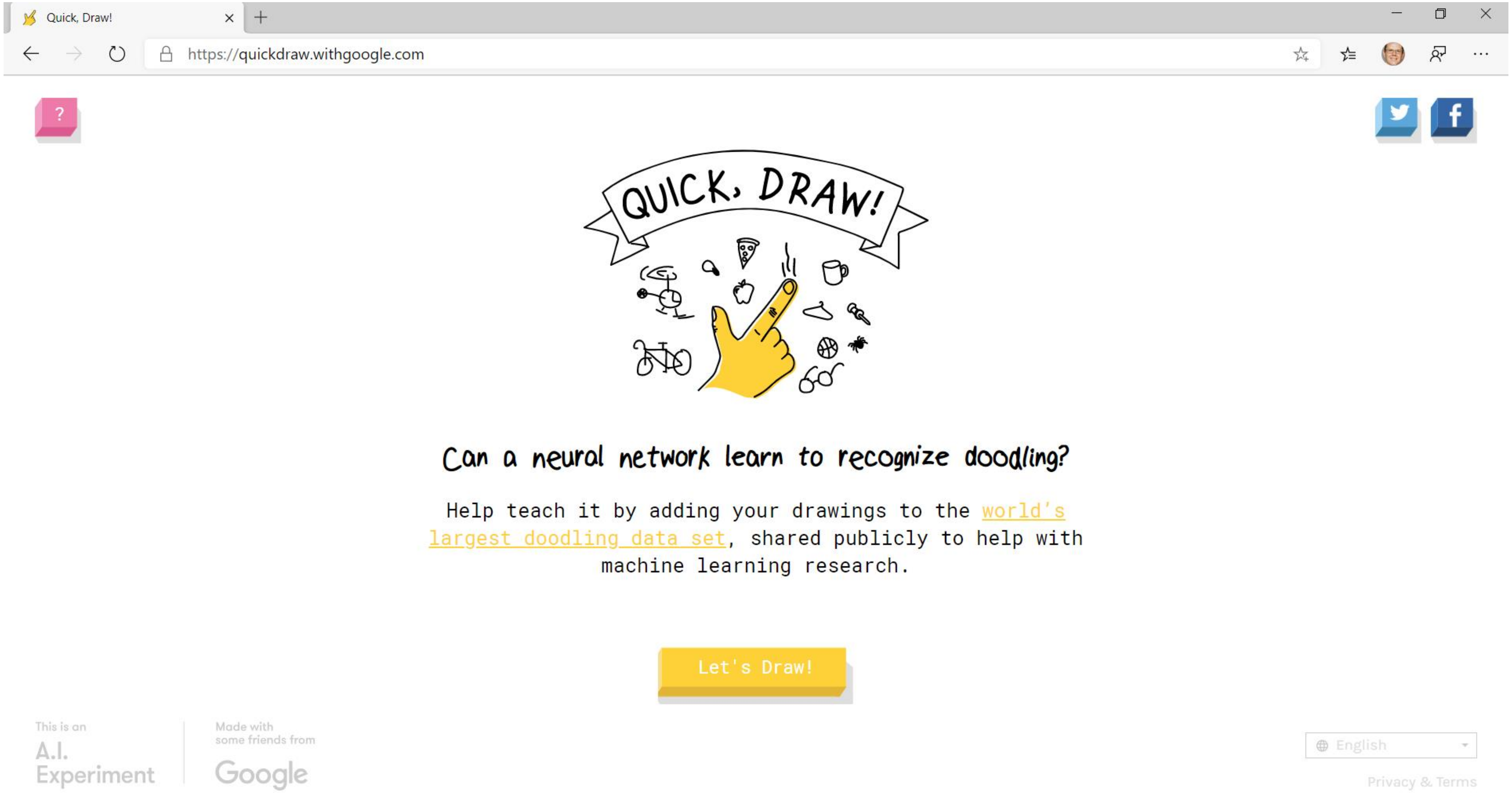


coderhumor · Follow

Albert Einstein: Insanity Is Doing the Same Thing Over and Over Again and Expecting Different Results

Machine learning:

Fun with https://quickdraw.withgoogle.com/



Quick, Draw!
https://quickdraw.withgoogle.com
?
QUICK, DRAW!

Can a neural network learn to recognize doodling?

Help teach it by adding your drawings to the world's largest doodling data set, shared publicly to help with machine learning research.

Let's Draw!
This is an A.I. Experiment
Made with some friends from Google
English
Privacy & Terms

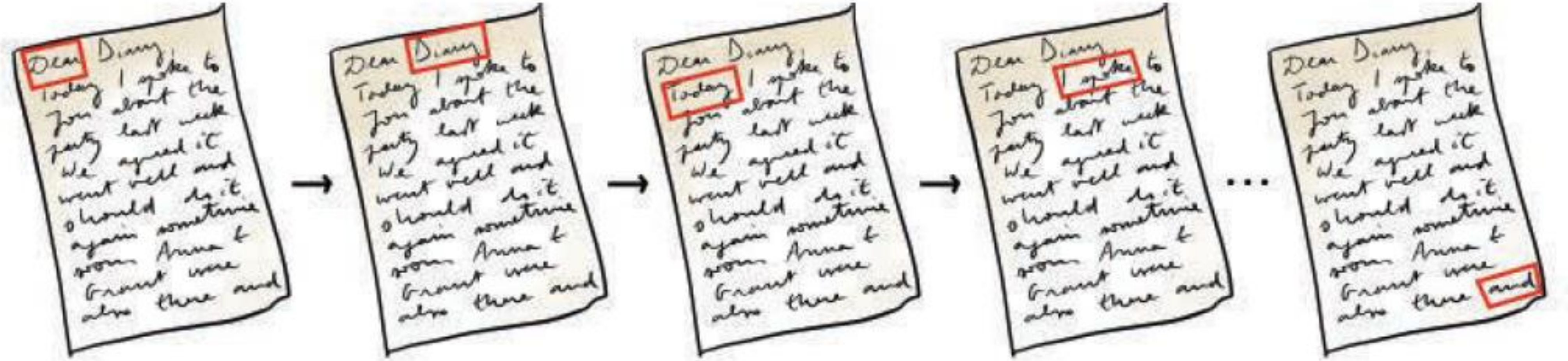
Summary

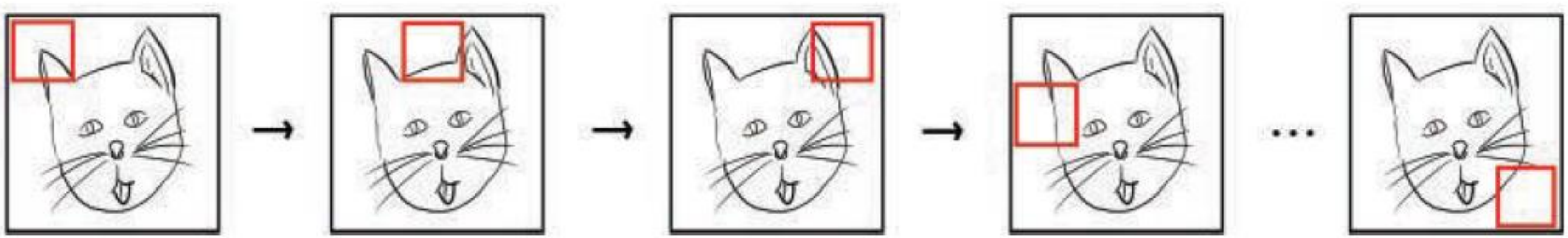
In this chapter, we traced the history of deep learning from its biological inspiration through to the AlexNet triumph in 2012 that brought the technique to the fore. All the while, we reiterated that the hierarchical architecture of deep learning models enables them to encode increasingly complex representations. To concretize this concept, we concluded with an interactive demonstration of hierarchical representations in action by training an artificial neural network in the TensorFlow Playground. In Chapter 2, we will expand on the ideas introduced in this chapter by moving from vision applications to language applications.

[DLI] Machine Vision

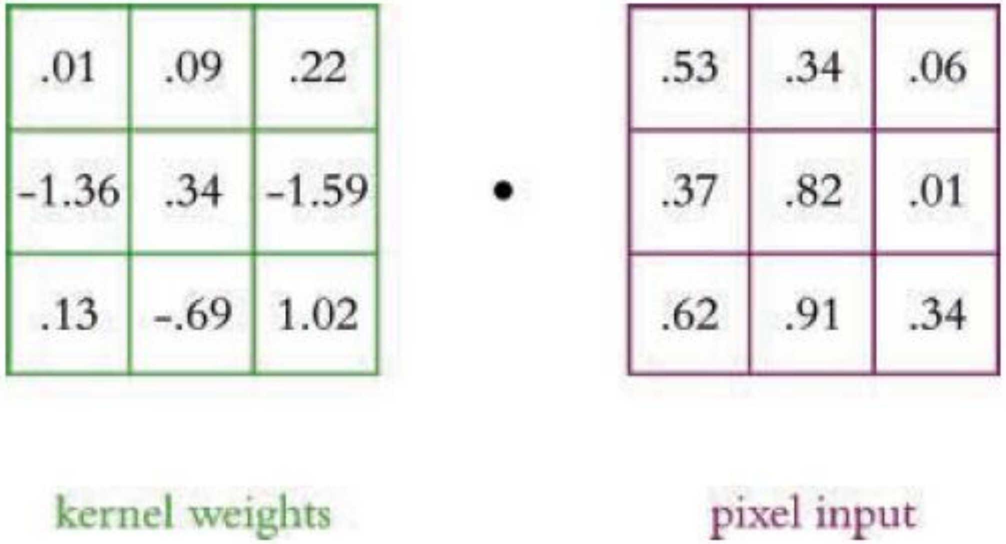
- Convolutional Neural Networks
- Pooling Layers
- LeNet-5 in Keras
- AlexNet and VGGNet in Keras
- Residual Networks
- Applications of Machine Vision

Motion of Convolution Filter





Convolution Filter Output Example



.01	.09	.22
-1.36	.34	-1.59
.13	-.69	1.02

•

.53	.34	.06
.37	.82	.01
.62	.91	.34

kernel weights

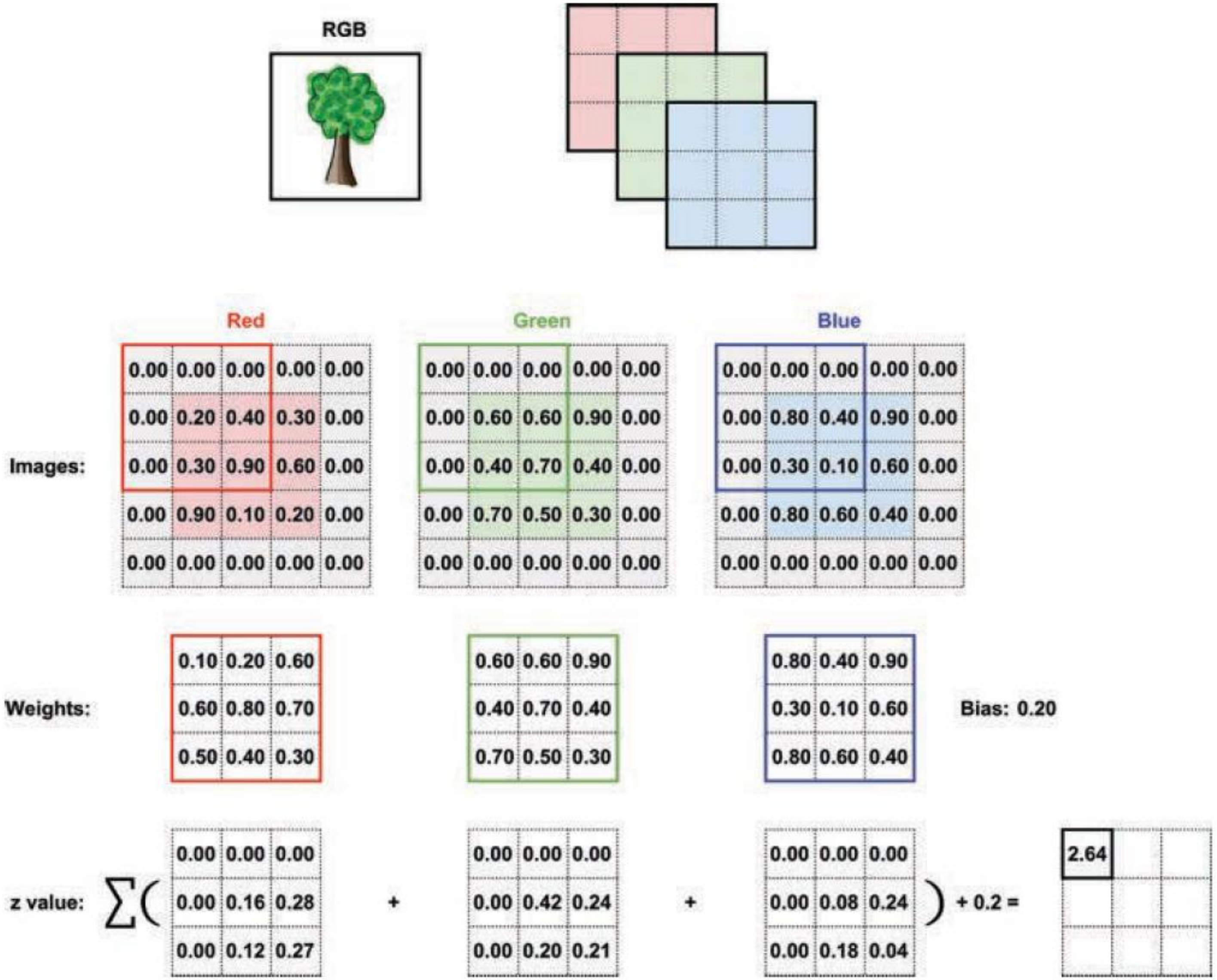
pixel input

$$
\begin{aligned}
w \cdot x &= .01 \times .53 + .09 \times .34 + .22 \times .06 \\
&\quad + -1.36 \times .37 + .34 \times .82 + -1.59 \times .01 \\
&\quad + .13 \times .62 + -.69 \times .91 + 1.02 \times .34 \\
&= -0.3917
\end{aligned}
$$

$$
\begin{aligned}
z &= w \cdot x + b \\
&= -0.39 + b \\
&= -0.39 + 0.20 \\
&= -0.19
\end{aligned}
$$

Convolution Filter Output Example: Same Padding

A filter will have weights for each input channel …



Images:

Red

0.00	0.00	0.00	0.00	0.00
0.00	0.20	0.40	0.30	0.00
0.00	0.30	0.90	0.60	0.00
0.00	0.90	0.10	0.20	0.00
0.00	0.00	0.00	0.00	0.00

Green

0.00	0.00	0.00	0.00	0.00
0.00	0.60	0.60	0.90	0.00
0.00	0.40	0.70	0.40	0.00
0.00	0.70	0.50	0.30	0.00
0.00	0.00	0.00	0.00	0.00

Blue

0.00	0.00	0.00	0.00	0.00
0.00	0.80	0.40	0.90	0.00
0.00	0.30	0.10	0.60	0.00
0.00	0.80	0.60	0.40	0.00
0.00	0.00	0.00	0.00	0.00

Weights:

0.10	0.20	0.60
0.60	0.80	0.70
0.50	0.40	0.30

0.60	0.60	0.90
0.40	0.70	0.40
0.70	0.50	0.30

0.80	0.40	0.90
0.30	0.10	0.60
0.80	0.60	0.40

Bias: 0.20

z value: ∑(

0.00	0.00	0.00
0.00	0.16	0.28
0.00	0.12	0.27

+

0.00	0.00	0.00
0.00	0.42	0.24
0.00	0.20	0.21

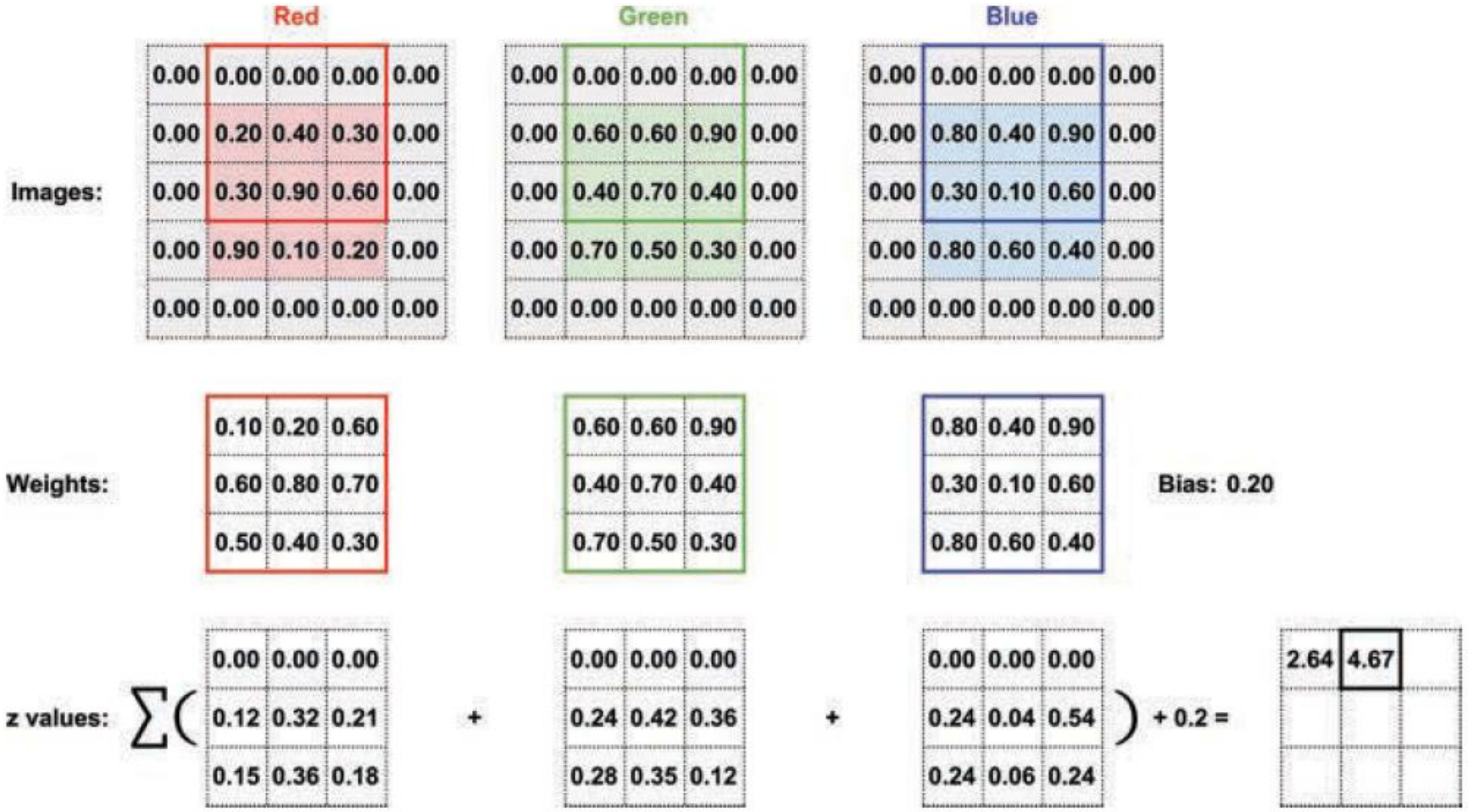
+

0.00	0.00	0.00
0.00	0.08	0.24
0.00	0.18	0.04

) + 0.2 =

2.64		

Same Filter, Different Position



Images:

Red

0.00	0.00	0.00	0.00	0.00
0.00	0.20	0.40	0.30	0.00
0.00	0.30	0.90	0.60	0.00
0.00	0.90	0.10	0.20	0.00
0.00	0.00	0.00	0.00	0.00

Green

0.00	0.00	0.00	0.00	0.00
0.00	0.60	0.60	0.90	0.00
0.00	0.40	0.70	0.40	0.00
0.00	0.70	0.50	0.30	0.00
0.00	0.00	0.00	0.00	0.00

Blue

0.00	0.00	0.00	0.00	0.00
0.00	0.80	0.40	0.90	0.00
0.00	0.30	0.10	0.60	0.00
0.00	0.80	0.60	0.40	0.00
0.00	0.00	0.00	0.00	0.00

Weights:

0.10	0.20	0.60
0.60	0.80	0.70
0.50	0.40	0.30

0.60	0.60	0.90
0.40	0.70	0.40
0.70	0.50	0.30

0.80	0.40	0.90
0.30	0.10	0.60
0.80	0.60	0.40

Bias: 0.20

z values: ∑(

0.00	0.00	0.00
0.12	0.32	0.21
0.15	0.36	0.18

+

0.00	0.00	0.00
0.24	0.42	0.36
0.28	0.35	0.12

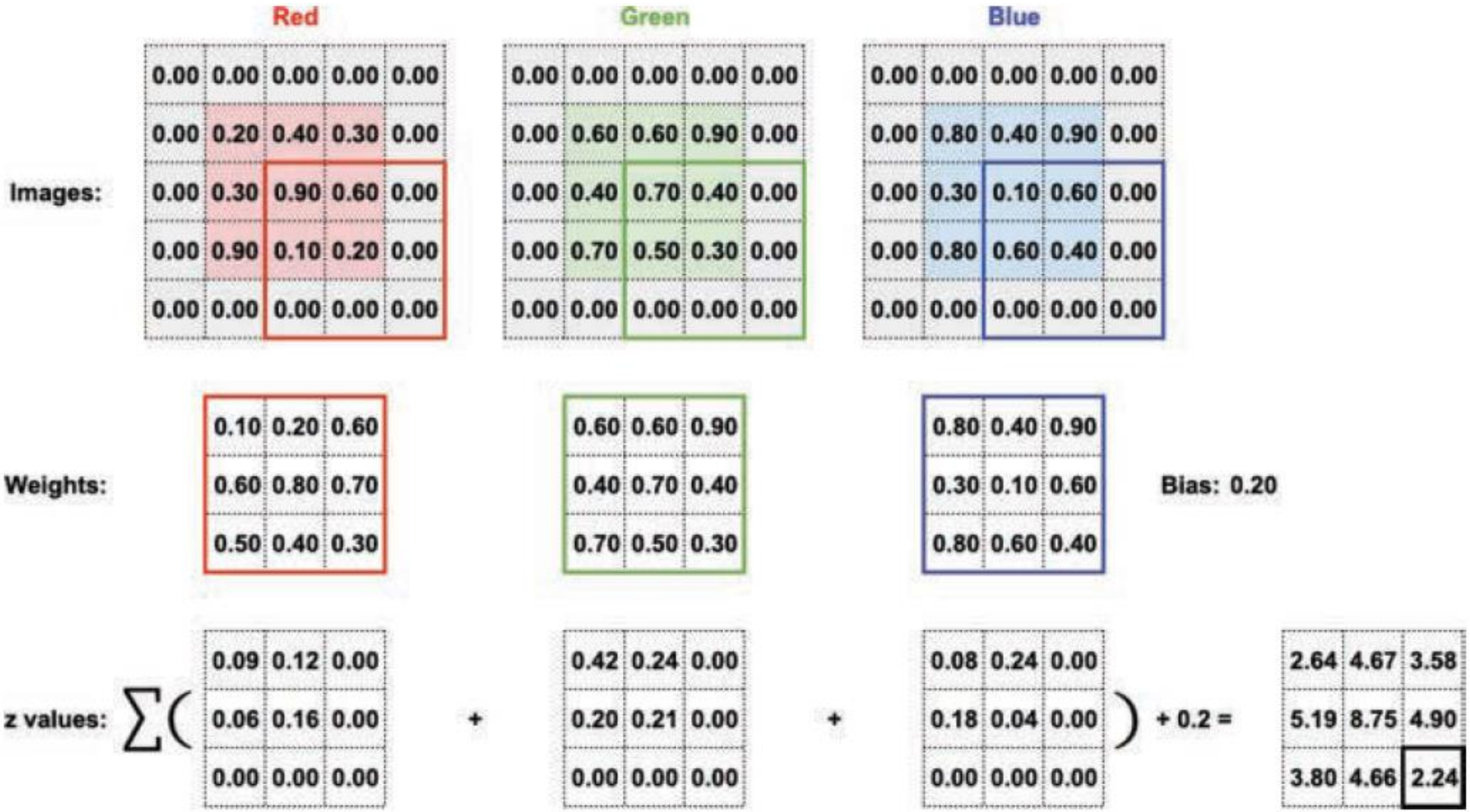
+

0.00	0.00	0.00
0.24	0.04	0.54
0.24	0.06	0.24

) + 0.2 =

2.64	4.67	

Same Filter, Last Position



Images:

Red

0.00	0.00	0.00	0.00	0.00
0.00	0.20	0.40	0.30	0.00
0.00	0.30	0.90	0.60	0.00
0.00	0.90	0.10	0.20	0.00
0.00	0.00	0.00	0.00	0.00

Green

0.00	0.00	0.00	0.00	0.00
0.00	0.60	0.60	0.90	0.00
0.00	0.40	0.70	0.40	0.00
0.00	0.70	0.50	0.30	0.00
0.00	0.00	0.00	0.00	0.00

Blue

0.00	0.00	0.00	0.00	0.00
0.00	0.80	0.40	0.90	0.00
0.00	0.30	0.10	0.60	0.00
0.00	0.80	0.60	0.40	0.00
0.00	0.00	0.00	0.00	0.00

Weights:

0.10	0.20	0.60
0.60	0.80	0.70
0.50	0.40	0.30

0.60	0.60	0.90
0.40	0.70	0.40
0.70	0.50	0.30

0.80	0.40	0.90
0.30	0.10	0.60
0.80	0.60	0.40

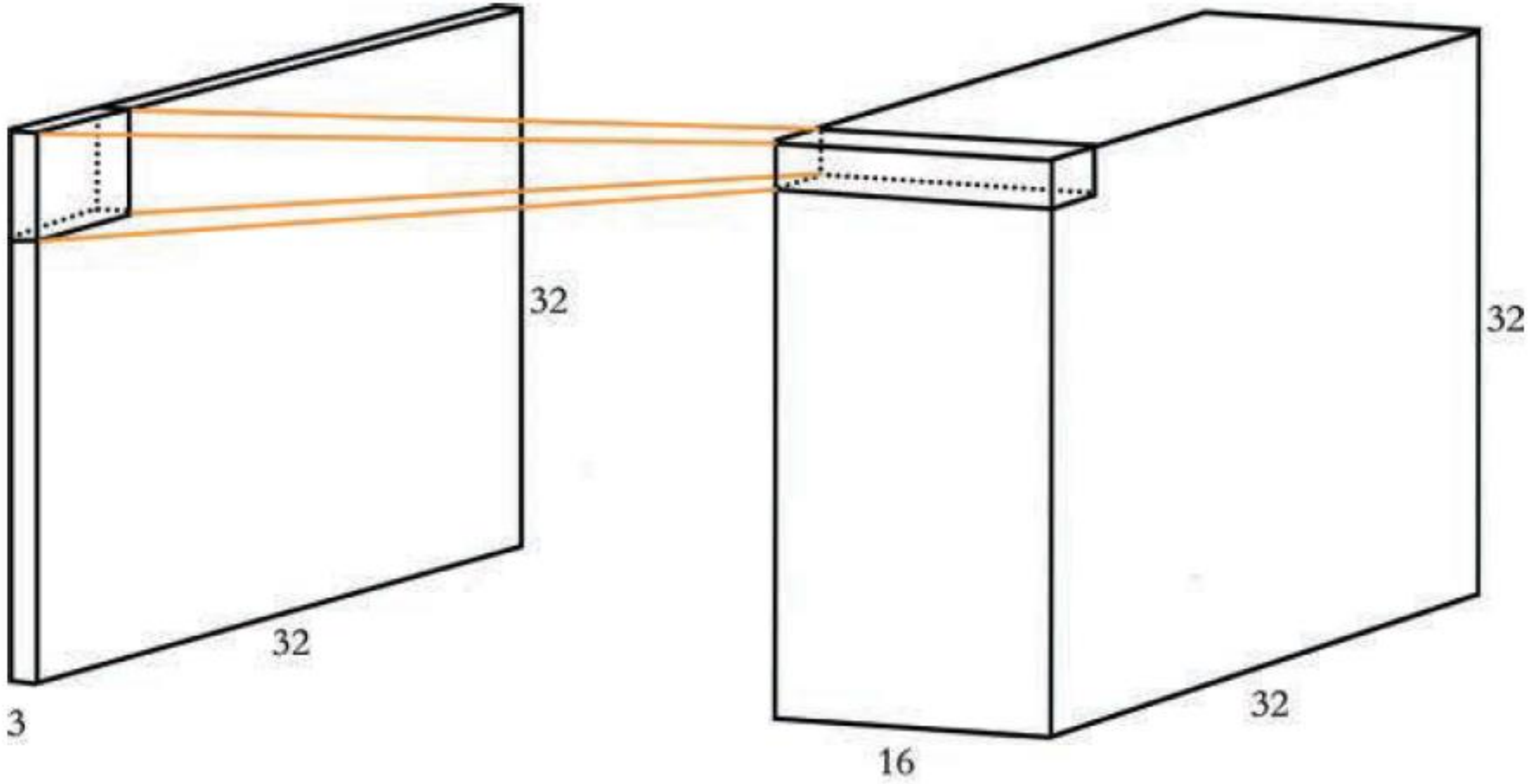
Bias: 0.20

z values:

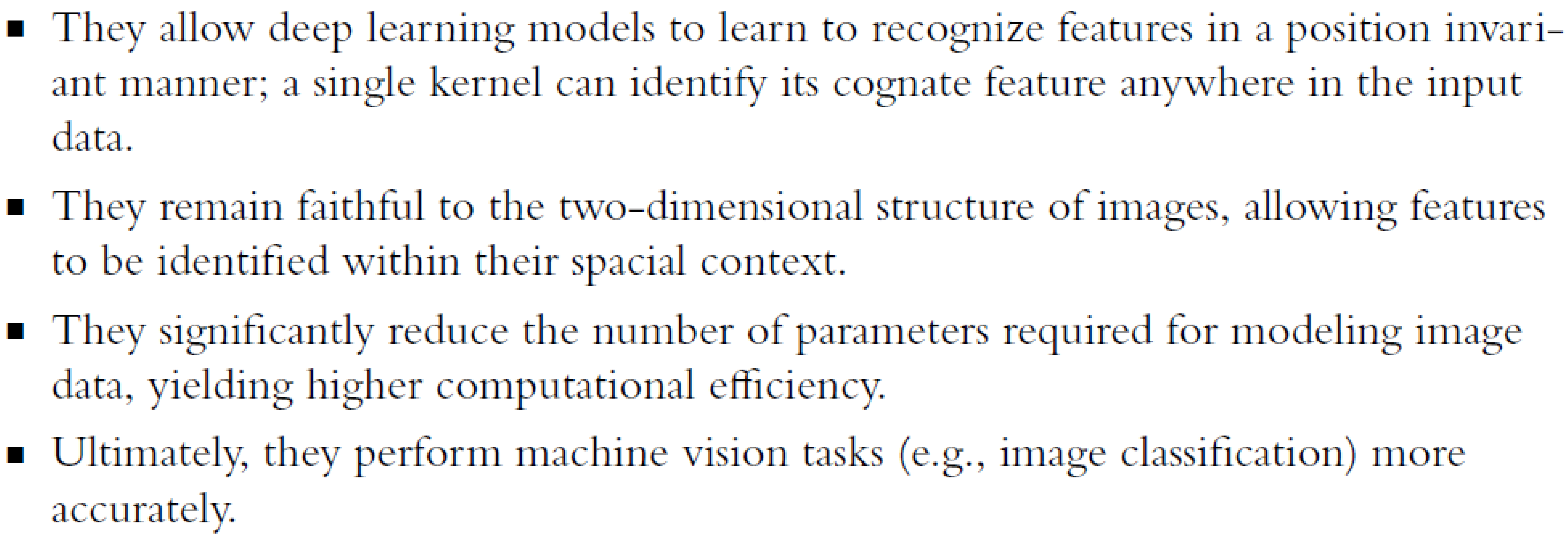
$$\sum\left(\begin{array}{ccc} 0.09 & 0.12 & 0.00 \\ 0.06 & 0.16 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{array} + \begin{array}{ccc} 0.42 & 0.24 & 0.00 \\ 0.20 & 0.21 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{array} + \begin{array}{ccc} 0.08 & 0.24 & 0.00 \\ 0.18 & 0.04 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{array}\right) + 0.2 = \begin{array}{ccc} 2.64 & 4.67 & 3.58 \\ 5.19 & 8.75 & 4.90 \\ 3.80 & 4.66 & \boxed{2.24} \end{array}$$

3 Channel Input to 16 Channel Output

- 16 convolution filters: same padding



Basic Convolution Filters



- They allow deep learning models to learn to recognize features in a position invariant manner; a single kernel can identify its cognate feature anywhere in the input data.
- They remain faithful to the two-dimensional structure of images, allowing features to be identified within their spacial context.
- They significantly reduce the number of parameters required for modeling image data, yielding higher computational efficiency.
- Ultimately, they perform machine vision tasks (e.g., image classification) more accurately.

Careful with the “higher computational efficiency” talk: for example, for “same” padding a single convolution filter will be applied once for every output position

Nice review of convolutional features: http://bit.ly/DeepViz

Activation Map Dimensions

This corresponds to the number of activations per filter

$$\text{Activation map} = \frac{D - F + 2P}{S} + 1$$

- D is the size of the image (either width or height, depending on whether you're calculating the width or height of the activation map).
- F is the size of the filter.
- P is the amount of padding.
- S is the stride length.

$$\text{Activation map} = \frac{D - F + 2P}{S} + 1$$

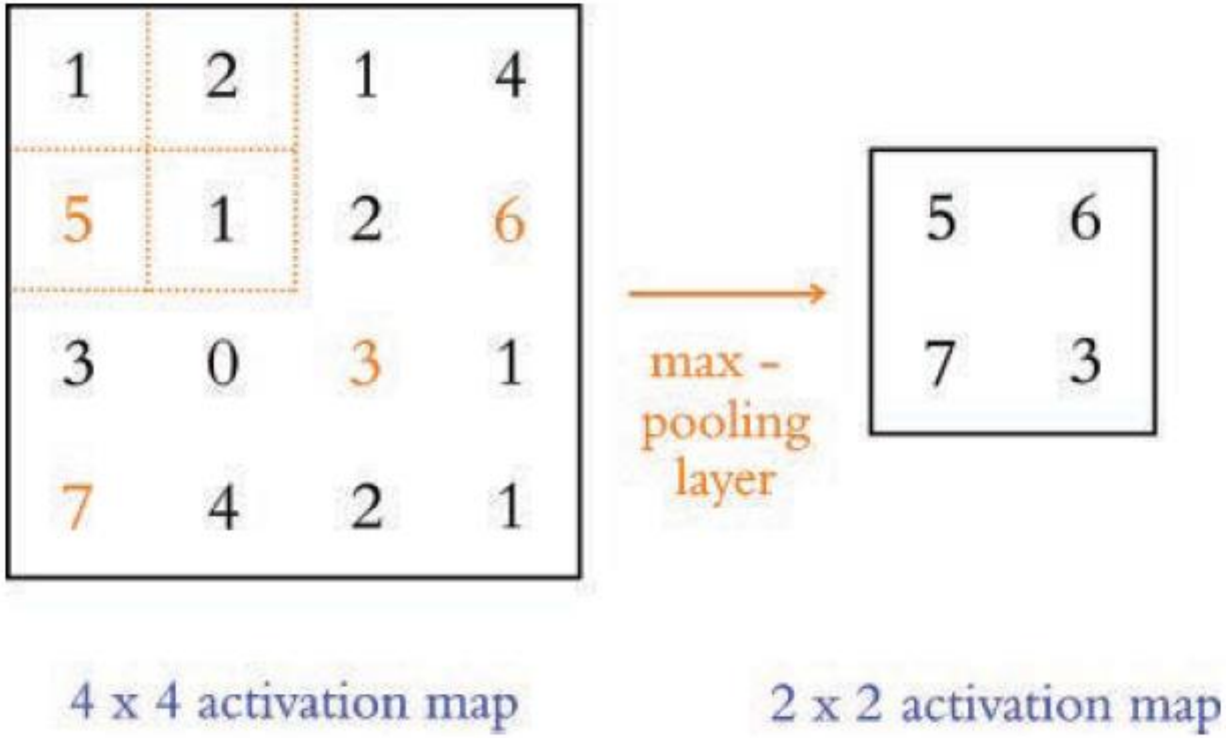
$$\text{Activation map} = \frac{28 - 5 + 2 \times 2}{1} + 1$$

$$\text{Activation map} = 28$$

… for MNIST, with a 5x5 filter, "same" padding, and stride 1

Max Pooling Example

pool_size = (2, 2) with stride = (2, 2)



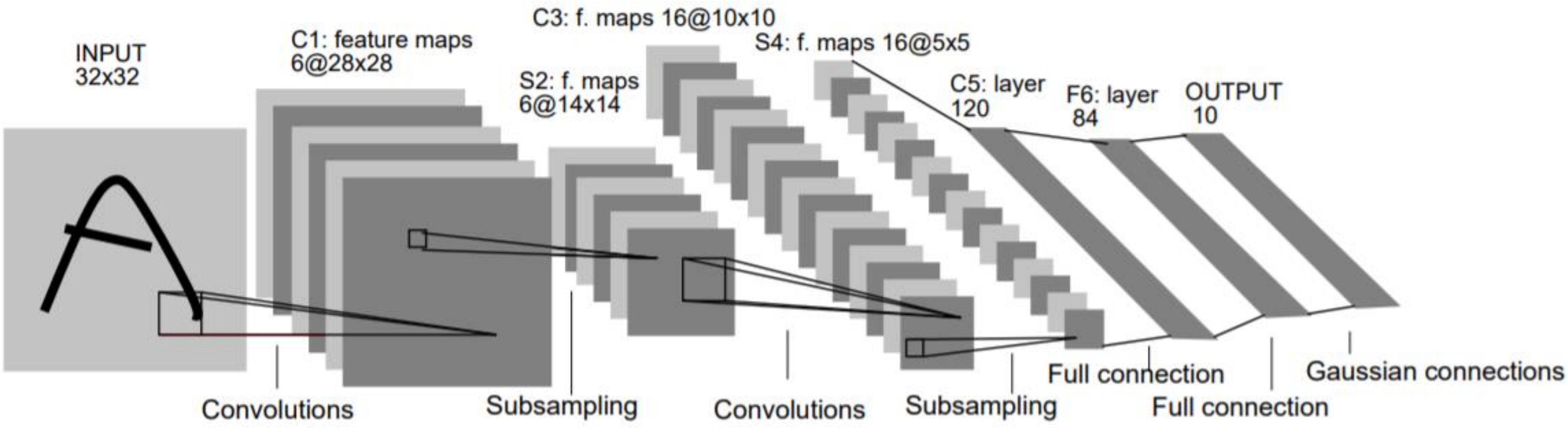
1	2	1	4
5	1	2	6
3	0	3	1
7	4	2	1

4 x 4 activation map

5	6
7	3

2 x 2 activation map

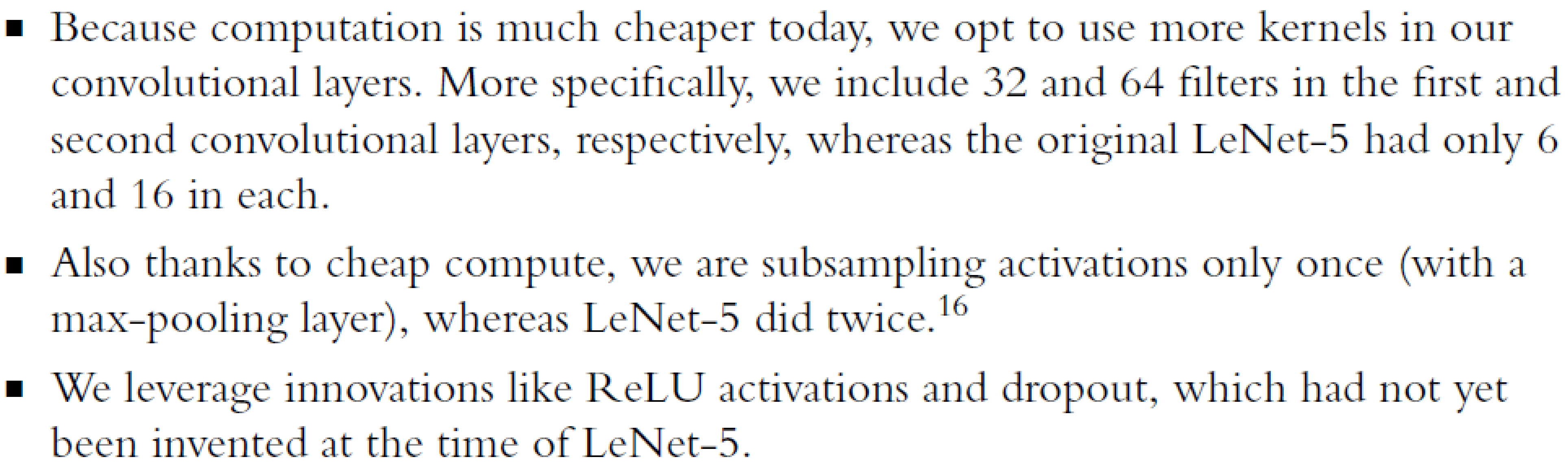
LeNet-5



5x5 convolution filters in layers C1, C3, and C5 [“valid” padding]
2x2 subsampling in layers S2 and S4

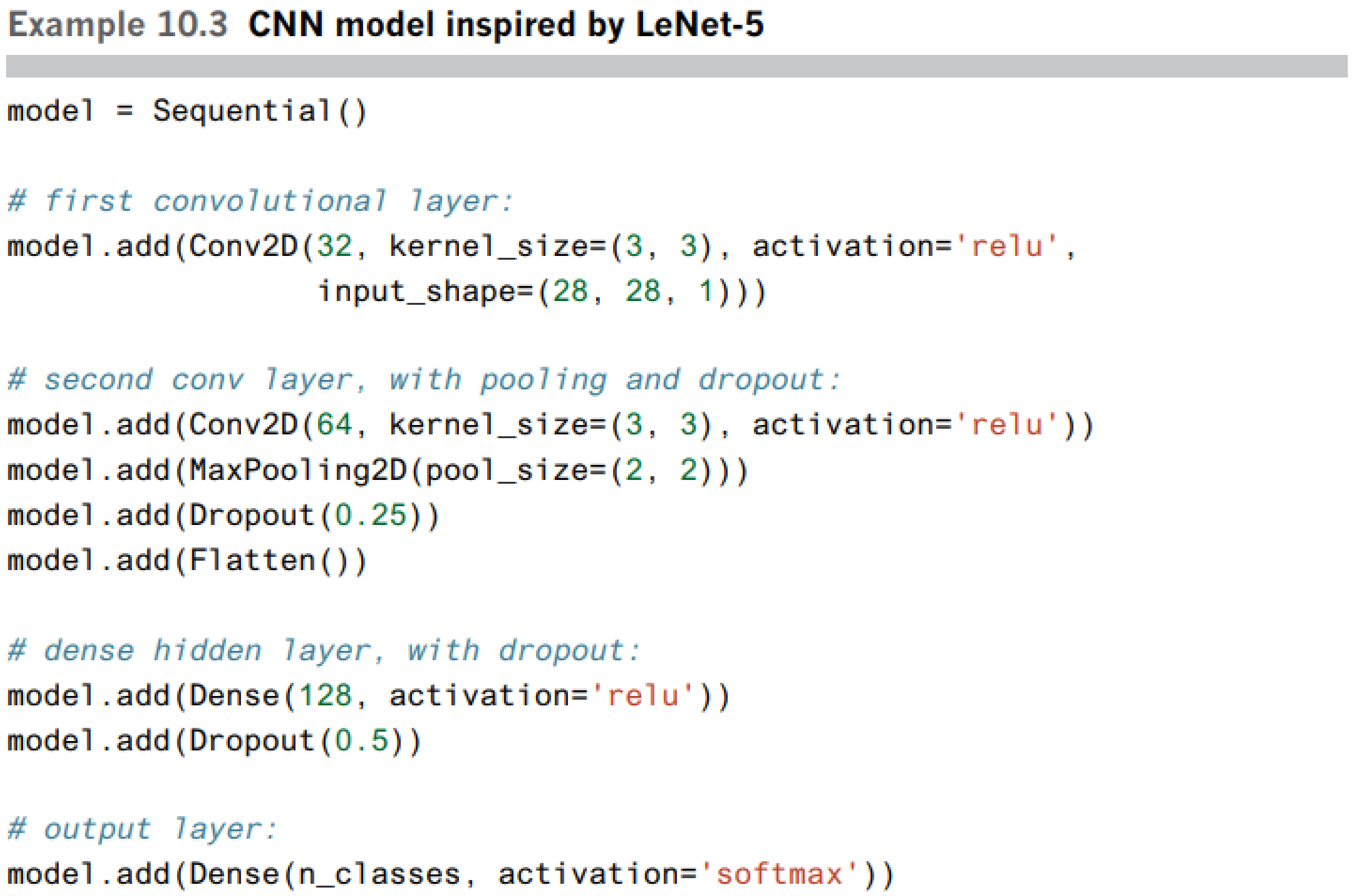
http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf

Twists for LeNet-5 in Keras



- Because computation is much cheaper today, we opt to use more kernels in our convolutional layers. More specifically, we include 32 and 64 filters in the first and second convolutional layers, respectively, whereas the original LeNet-5 had only 6 and 16 in each.
- Also thanks to cheap compute, we are subsampling activations only once (with a max-pooling layer), whereas LeNet-5 did twice.[16]
- We leverage innovations like ReLU activations and dropout, which had not yet been invented at the time of LeNet-5.

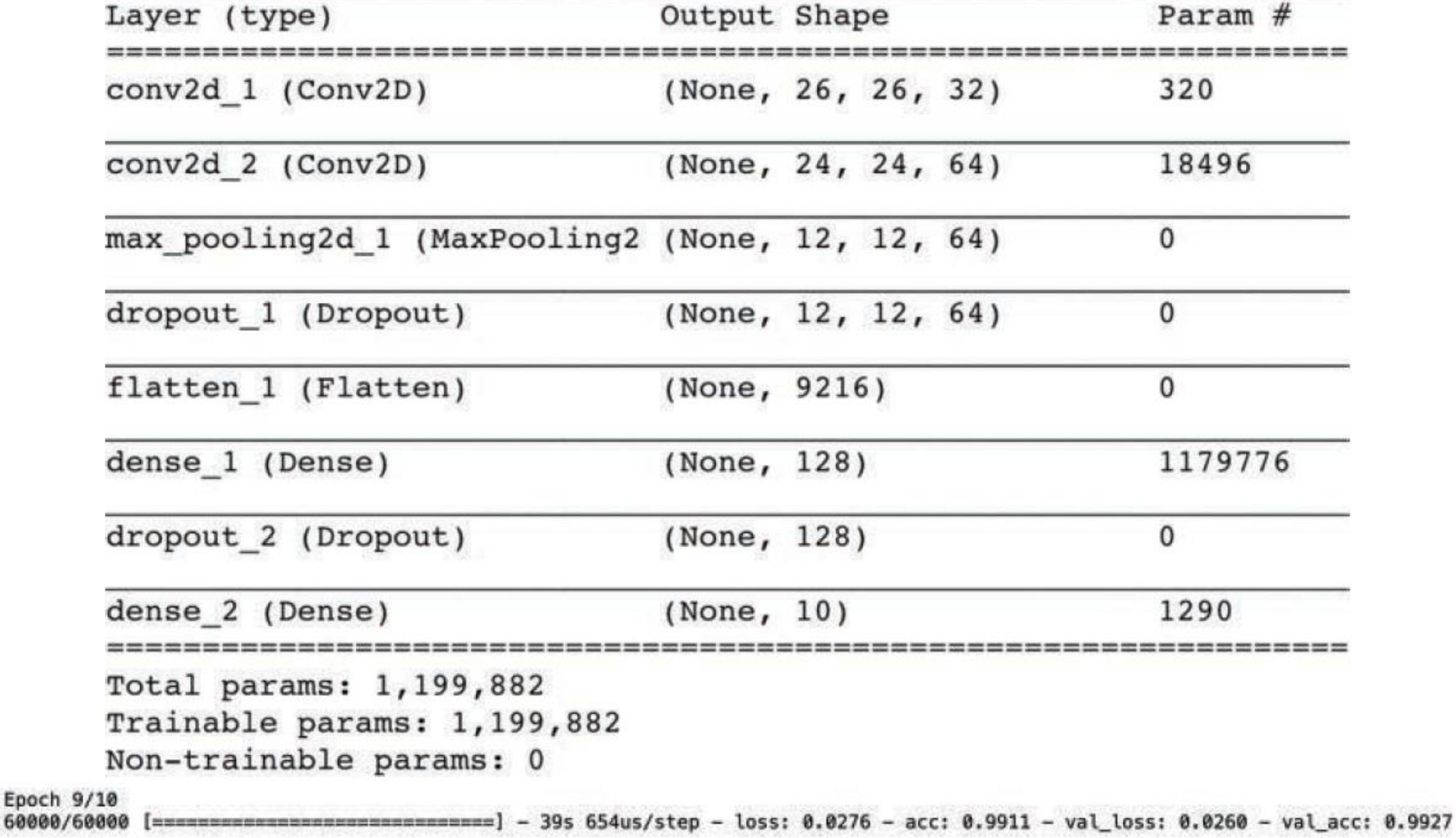
CNN Inspired by LeNet-5



Example 10.3 CNN model inspired by LeNet-5

```
model = Sequential()

# first convolutional layer:
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                 input_shape=(28, 28, 1)))

# second conv layer, with pooling and dropout:
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())

# dense hidden layer, with dropout:
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

# output layer:
model.add(Dense(n_classes, activation='softmax'))
```

CNN Inspired by LeNet-5

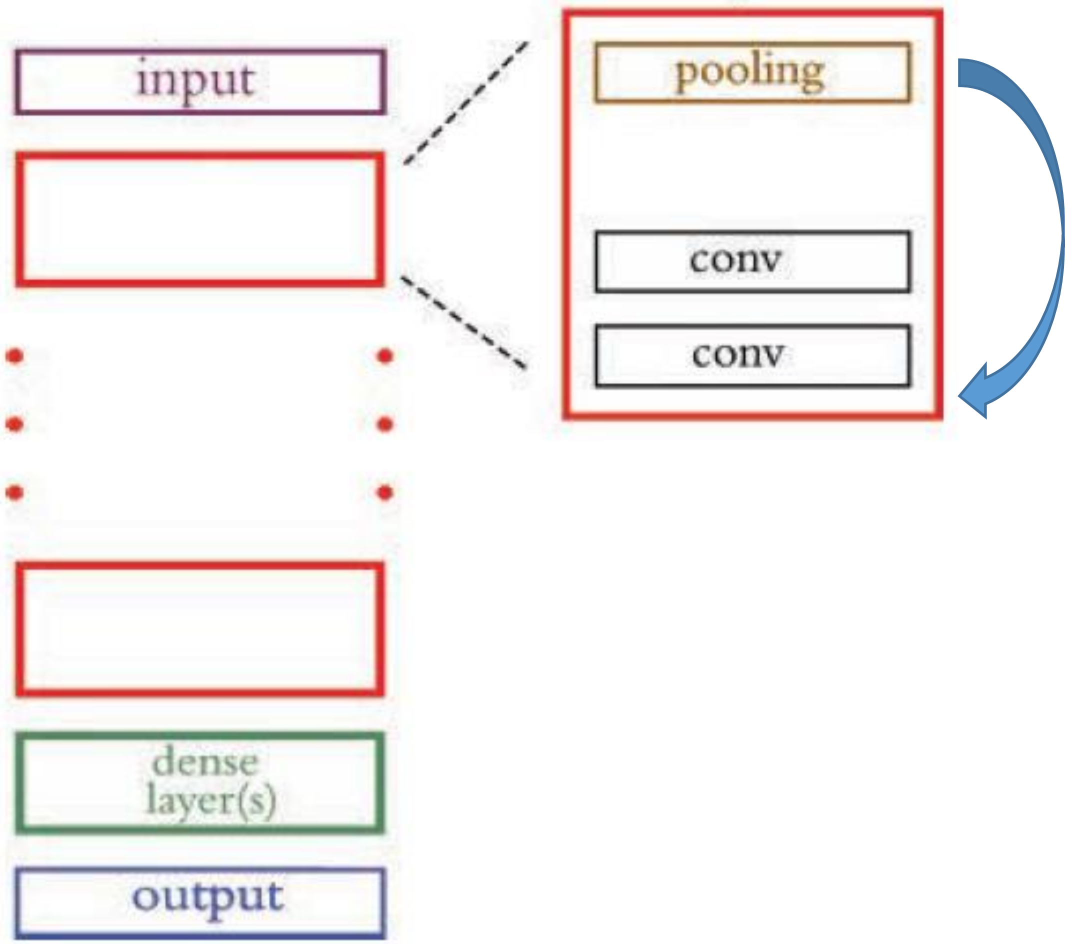


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
```

```
Epoch 9/10
60000/60000 [==============================] - 39s 654us/step - loss: 0.0276 - acc: 0.9911 - val_loss: 0.0260 - val_acc: 0.9927
```

General Approach to CNN Design

“conv > conv > pooling” rather than “pooling > conv > conv”



CNN Inspired by Alex Net

```
model = Sequential()

# first conv-pool block:
model.add(Conv2D(96, kernel_size=(11, 11),
   strides=(4, 4), activation='relu',
   input_shape=(224, 224, 3)))

model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(BatchNormalization())

# second conv-pool block:
model.add(Conv2D(256, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(BatchNormalization())

# third conv-pool block:
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(384, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(384, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(BatchNormalization())

# dense layers:
model.add(Flatten())
model.add(Dense(4096, activation='tanh'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='tanh'))
model.add(Dropout(0.5))

# output layer:
model.add(Dense(17, activation='softmax'))
```

The Oxford Flowers dataset has 17 classes:
https://www.robots.ox.ac.uk/~vgg/data/flowers/17/

CNN Inspired by VGGNet

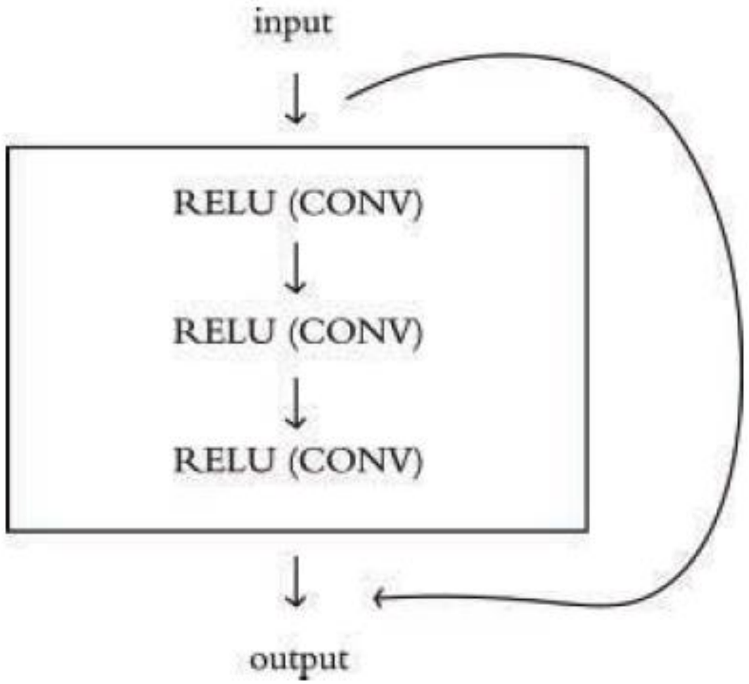
```
model = Sequential()

model.add(Conv2D(64, 3, activation='relu',
   input_shape=(224, 224, 3)))

model.add(Conv2D(64, 3, activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(BatchNormalization())

model.add(Conv2D(128, 3, activation='relu'))
model.add(Conv2D(128, 3, activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(BatchNormalization())

model.add(Conv2D(256, 3, activation='relu'))
model.add(Conv2D(256, 3, activation='relu'))
model.add(Conv2D(256, 3, activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(BatchNormalization())
```

```
model.add(Conv2D(512, 3, activation='relu'))
model.add(Conv2D(512, 3, activation='relu'))
model.add(Conv2D(512, 3, activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(BatchNormalization())

model.add(Conv2D(512, 3, activation='relu'))
model.add(Conv2D(512, 3, activation='relu'))
model.add(Conv2D(512, 3, activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(17, activation='softmax'))
```

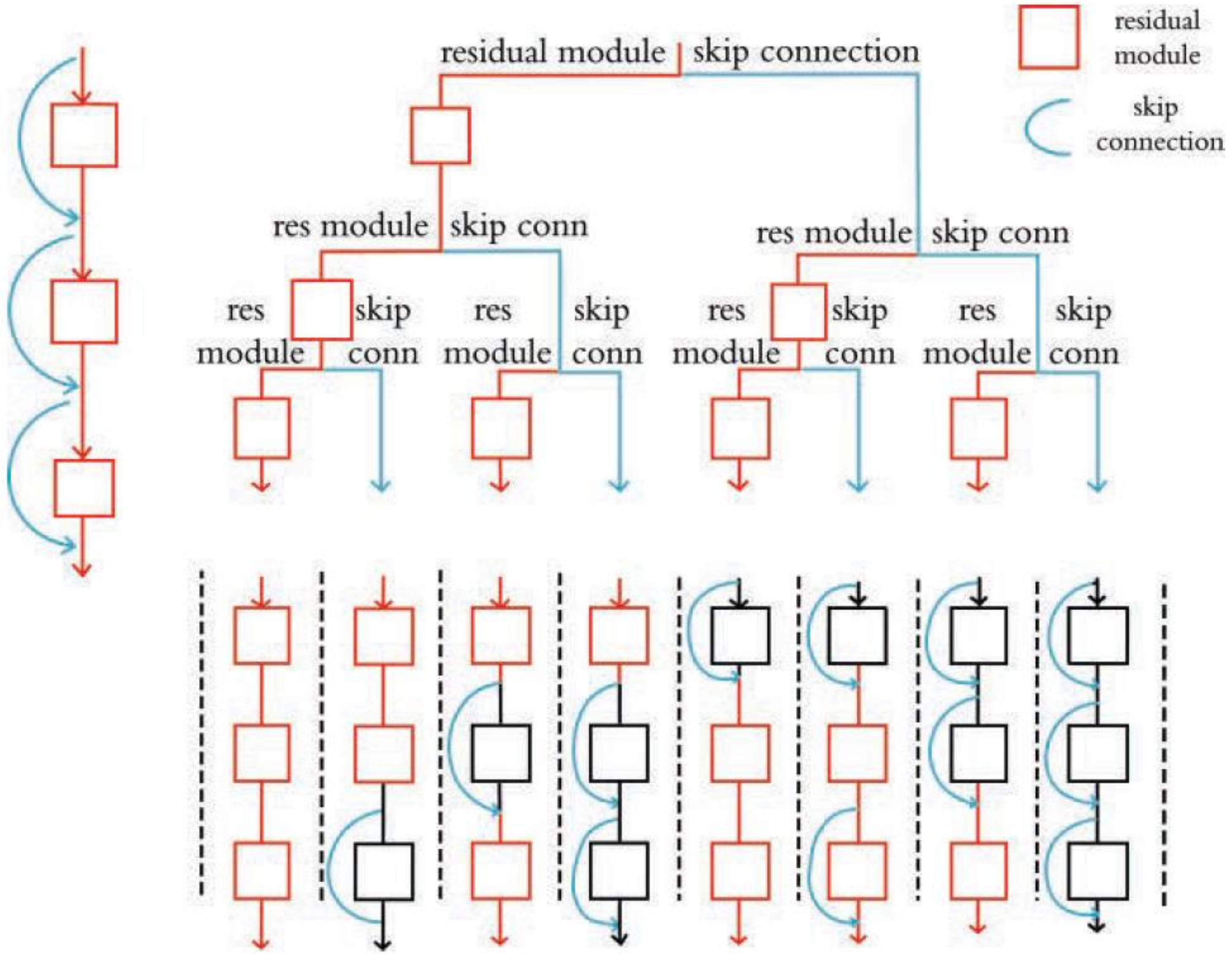
Computing Residuals

Residual = Desired – Current [Rearranging: Desired = Current + Residual]

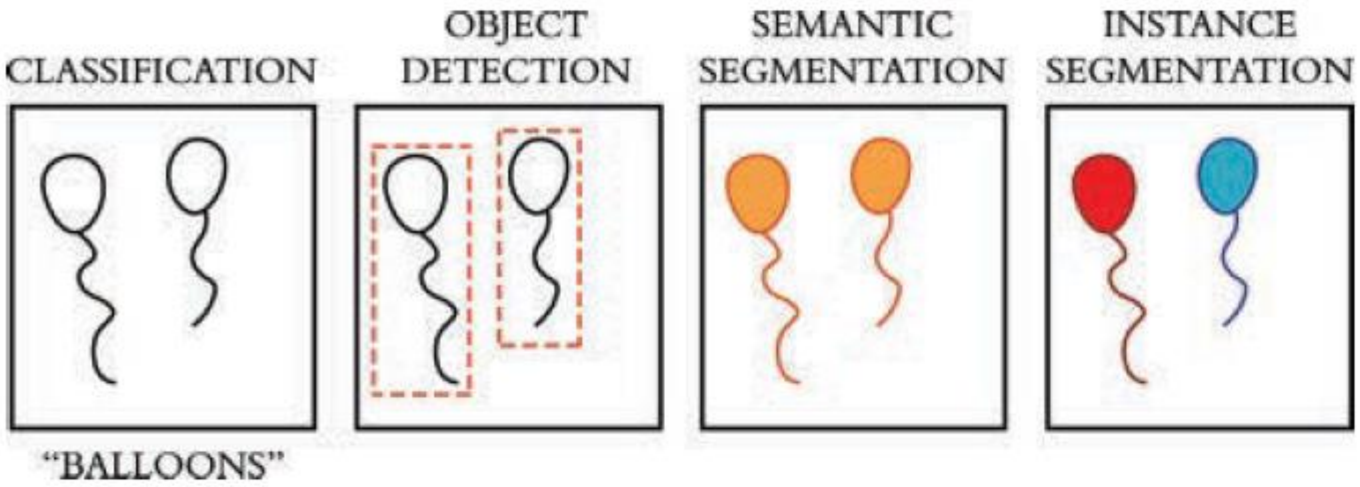
- What do we need to add to Current to get Desired
- Shortcut helps reduce vanishing gradients [shortens gradient path]



Skip Connections [residual module outputs 0]



Computer Vision Applications



Object Detection Pipeline

- A Region Of Interest (ROI) must be identified
- Automatic feature extraction is performed on this region
- The region is classified

R-CNN

Region-based Convolutional Neural Network [Berkeley, Girshick et al]:

1. Perform a selective search for regions of interest (ROIs) within the image
2. Extract features from these ROIs by using a CNN
3. Combine two “traditional” machine learning approaches—called linear regression and support vector machines—to, respectively, refine the locations of bounding boxes and classify objects within each of those boxes

Limitations:

- It was inflexible: The input size was fixed to a single specific image shape
- It was slow and computationally expensive: Both training and inference are multistage processes involving CNNs, linear regression models, and support vector machines

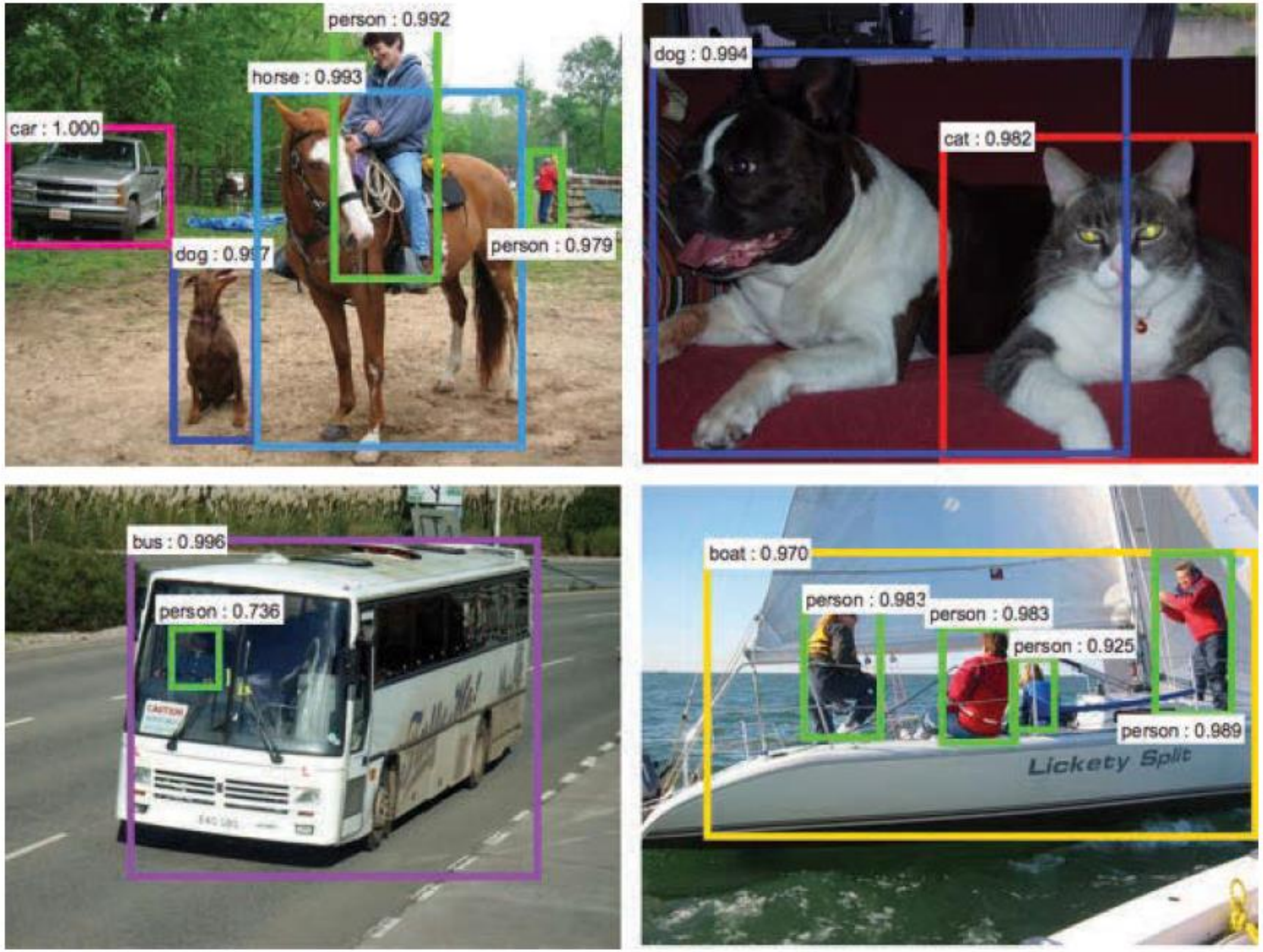
Fast R-CNN

- The chief innovation here was the realization that during step 2 of the R-CNN algorithm, the CNN was unnecessarily being run multiple times, once for each region of interest
- The Fast R-CNN model has to perform feature extraction using a CNN only once for a given image (thereby reducing computational complexity), and then the ROI search and dense layers work together to finish the object-detection task

Faster R-CNN

To overcome the ROI-search bottleneck of R-CNN and Fast R-CNN, Ren and his colleagues [at Microsoft research] had the cunning insight to leverage the feature activation maps from the model's CNN for this step, too

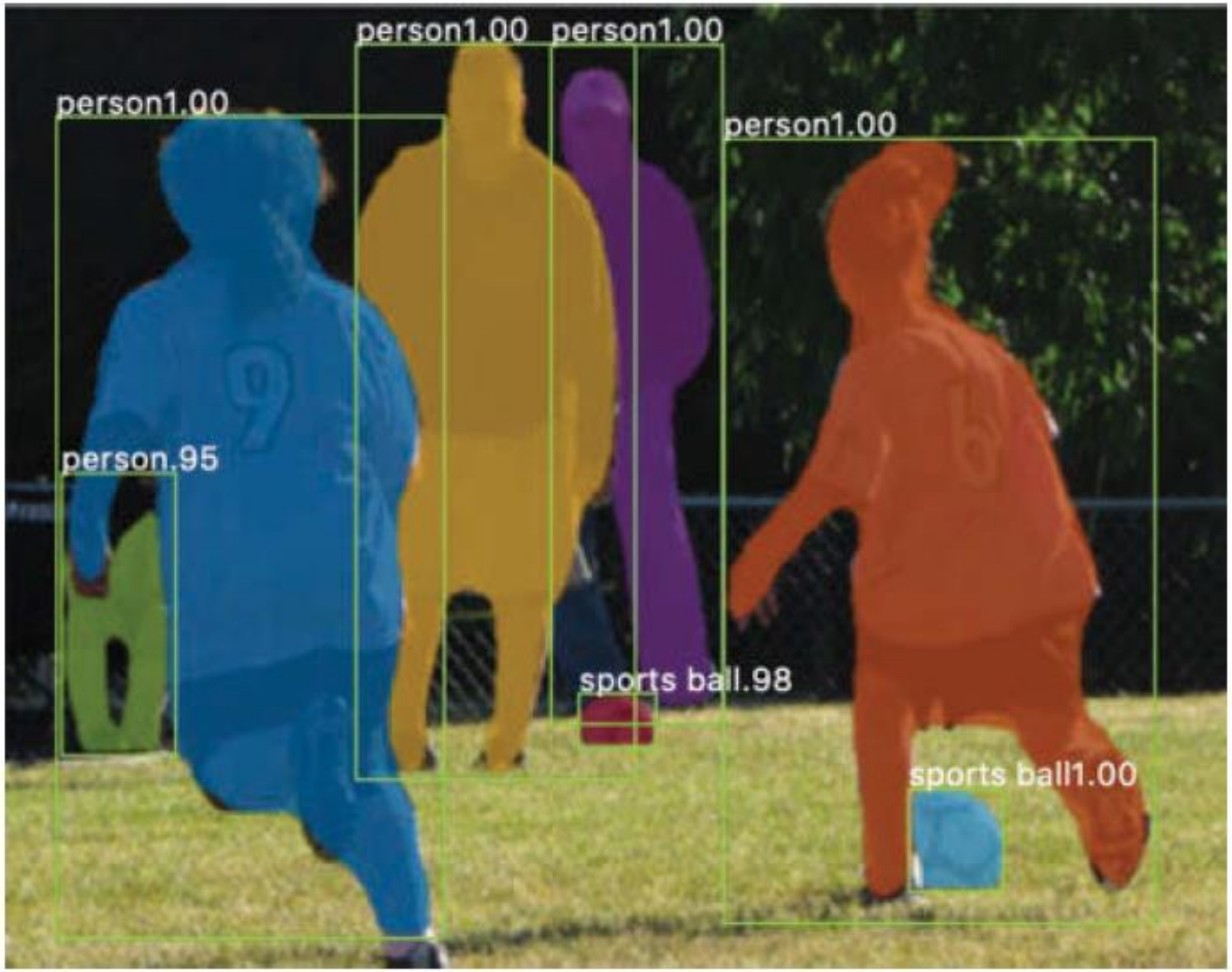
Object Detection Output from Faster R-CNN



YOLO: You Only Look Once

- YOLO begins with a pretrained CNN for feature extraction
- Next, the image is divided into a series of cells, and, for each cell, a number of bounding boxes and object-classification probabilities are predicted
- Bounding boxes with class probabilities above a threshold value are selected, and these combine to locate an object within an image
- You can think of the YOLO method as aggregating many smaller bounding boxes, but only if they have a reasonably good probability of containing any given object class

Instance Segmentation



Mask R-CNN

Facebook AI Research (FAIR):

1. Using the existing Faster R-CNN architecture to propose ROIs within the image that are likely to contain objects
2. An ROI classifier predicting what kind of object exists in the bounding box while also refining the location and size of the bounding box
3. Using the bounding box to grab the parts of the feature maps from the underlying CNN that correspond to that part of the image
4. Feeding the feature maps for each ROI into a fully convolutional network that outputs a mask indicating which pixels correspond to the object in the image

U-Net

- The U-Net model consists of a fully convolutional architecture, which begins with a contracting path that produces successively smaller and deeper activation maps through multiple convolution and max-pooling steps
- Subsequently, an expanding path restores these deep activation maps back to full resolution through multiple upsampling and convolution steps
- These two paths—the contracting and expanding paths—are symmetrical (forming a “U” shape), and because of this symmetry the activation maps from the contracting path can be concatenated onto those of the expanding path

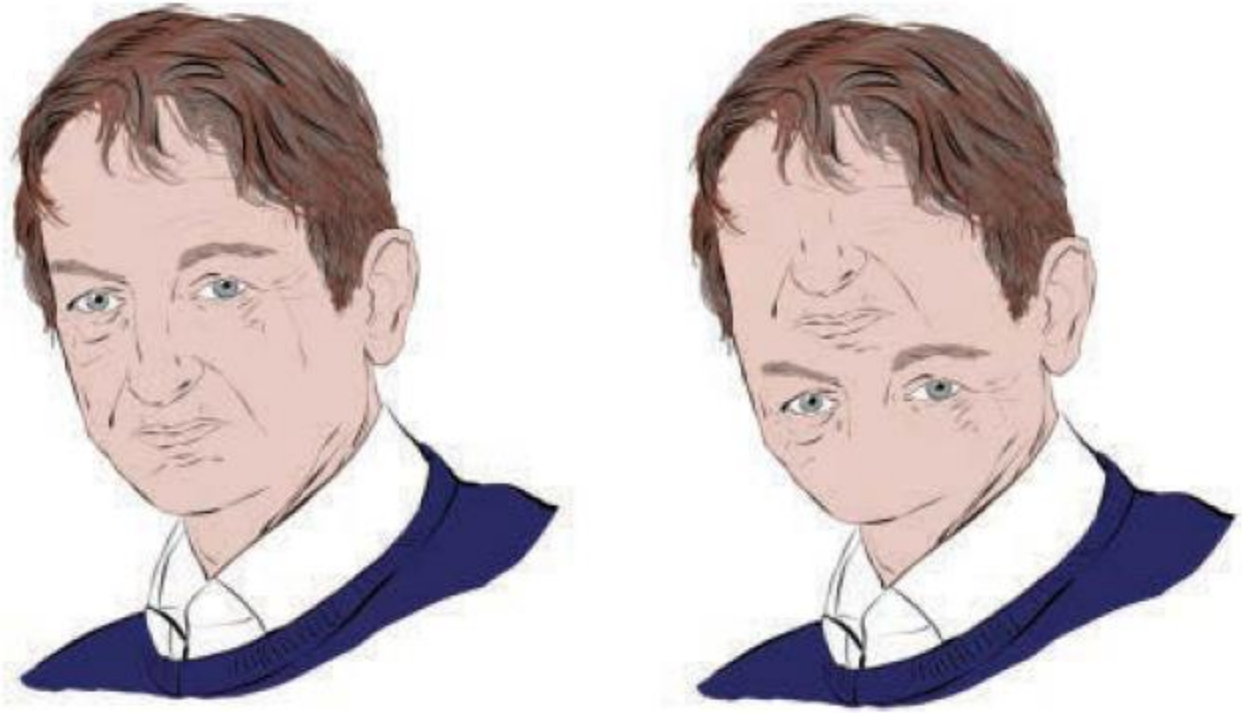
Transfer Learning

- Oxford University’s Visual Geometry Group’s 19-layer CNN used as a pretrained network (pretrained on ImageNet data) to predict whether an image contains a hot dog or not

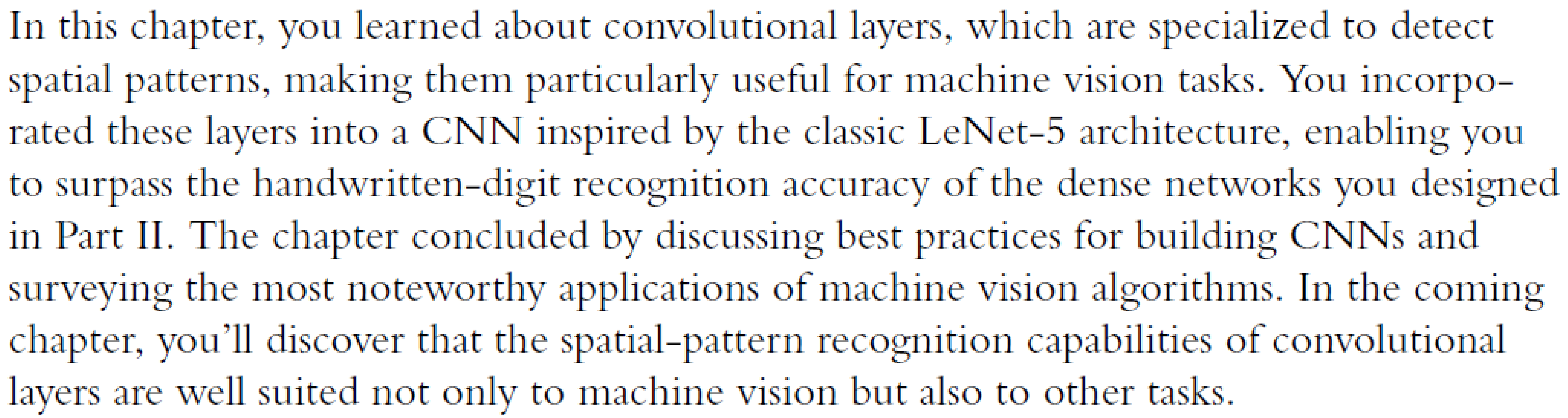
- model.add(Dense(2, activation='softmax', name='predictions’))

Capsule Networks

Unlike a CNN, a Capsule Network takes positional information into consideration

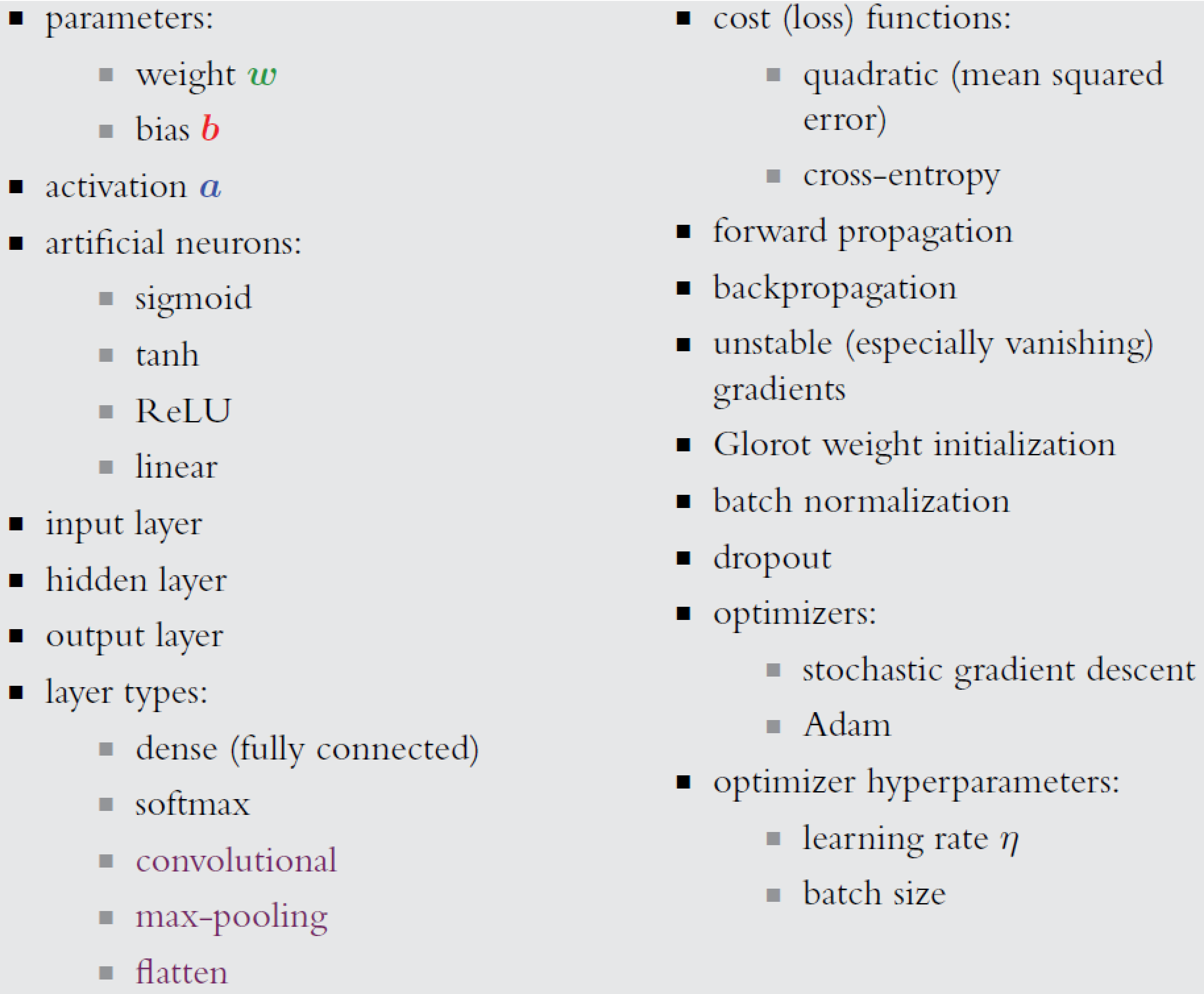


Summary

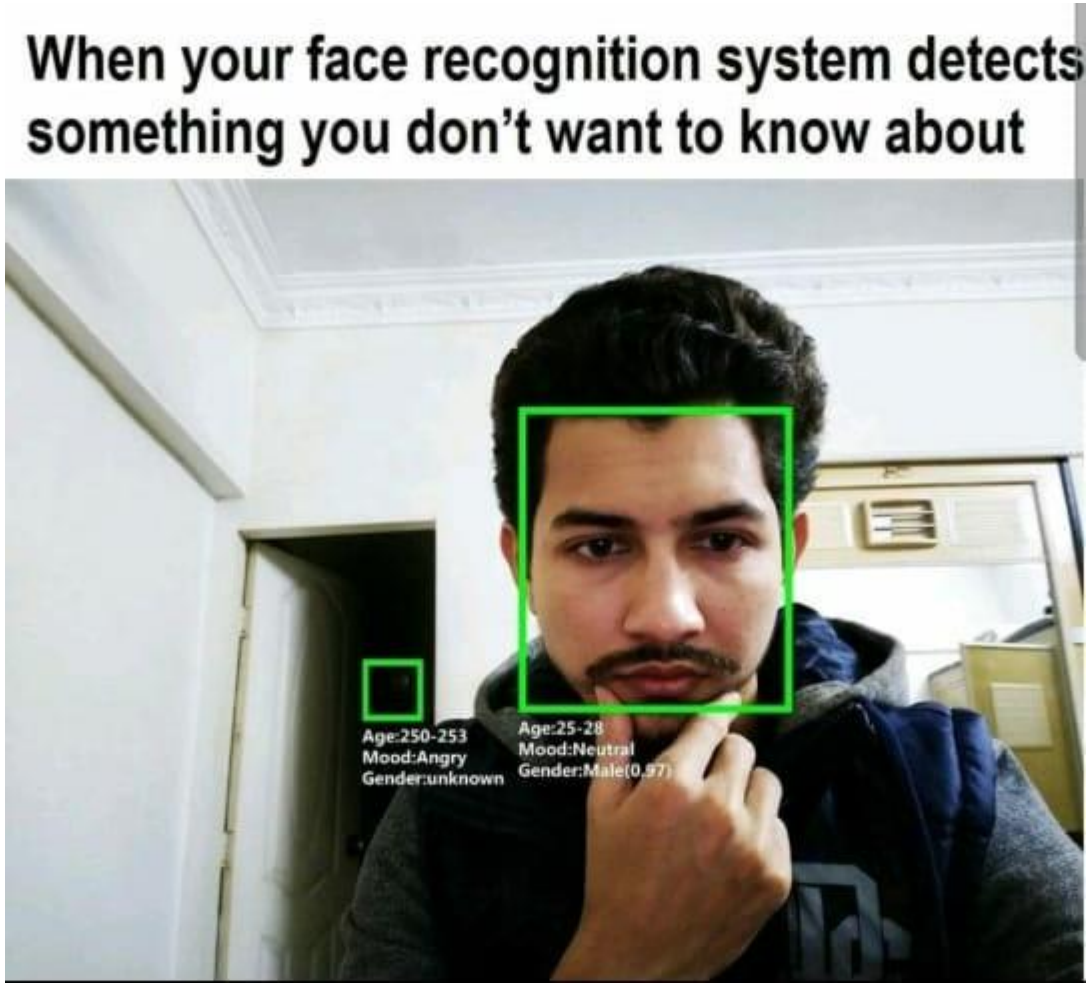


In this chapter, you learned about convolutional layers, which are specialized to detect spatial patterns, making them particularly useful for machine vision tasks. You incorporated these layers into a CNN inspired by the classic LeNet-5 architecture, enabling you to surpass the handwritten-digit recognition accuracy of the dense networks you designed in Part II. The chapter concluded by discussing best practices for building CNNs and surveying the most noteworthy applications of machine vision algorithms. In the coming chapter, you'll discover that the spatial-pattern recognition capabilities of convolutional layers are well suited not only to machine vision but also to other tasks.

Concepts



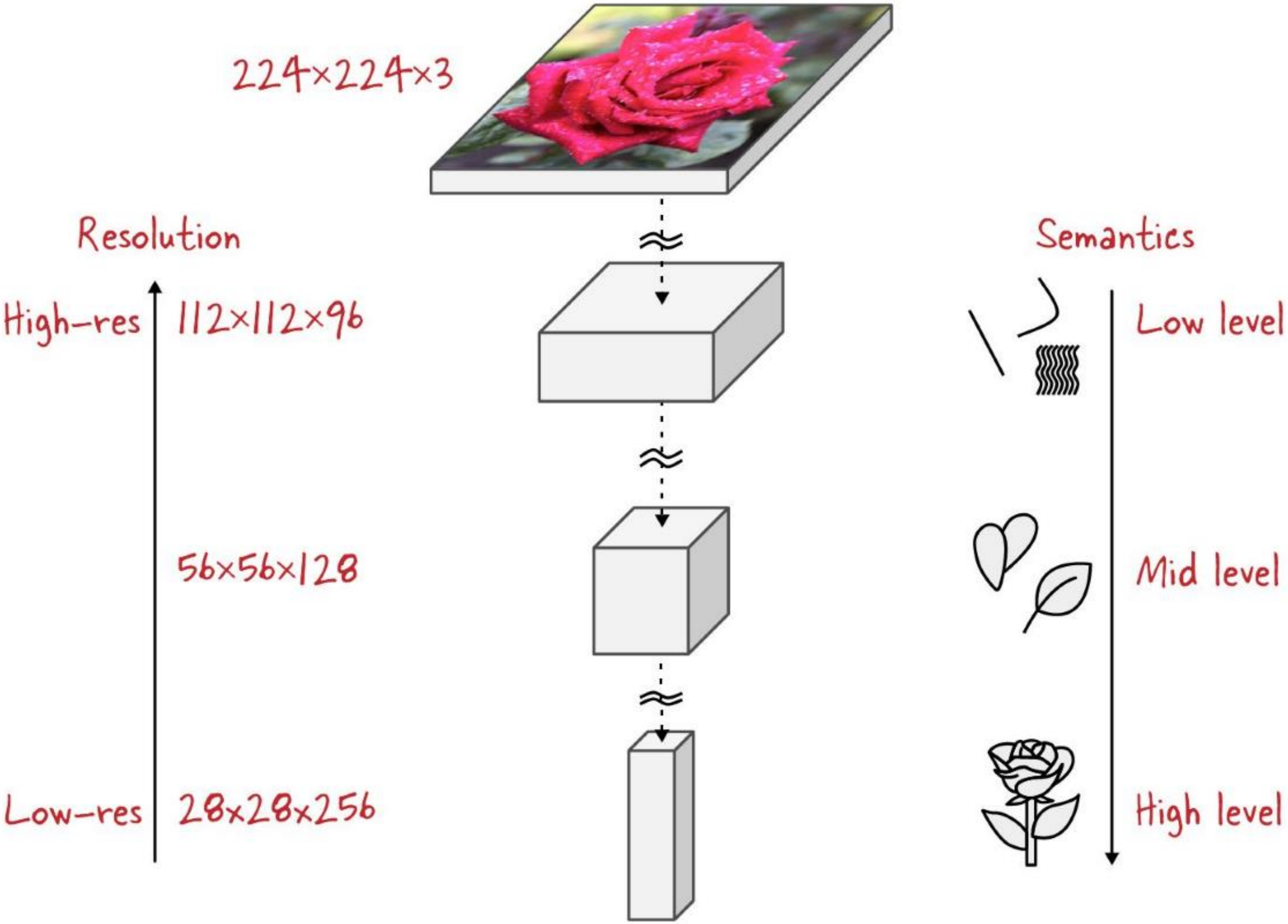
- parameters:
  - weight w
  - bias b
- activation a
- artificial neurons:
  - sigmoid
  - tanh
  - ReLU
  - linear
- input layer
- hidden layer
- output layer
- layer types:
  - dense (fully connected)
  - softmax
  - convolutional
  - max-pooling
  - flatten
- cost (loss) functions:
  - quadratic (mean squared error)
  - cross-entropy
- forward propagation
- backpropagation
- unstable (especially vanishing) gradients
- Glorot weight initialization
- batch normalization
- dropout
- optimizers:
  - stochastic gradient descent
  - Adam
- optimizer hyperparameters:
  - learning rate η
  - batch size


When your face recognition system detects something you don't want to know about
Age:250-253
Mood:Angry
Gender:unknown
Age:25-28
Mood:Neutral
Gender:Male(0.97)

https://www.reddit.com/r/FunnyandSad/comments/mvo6dz/age_250253_mood_angry_gender_unknown/

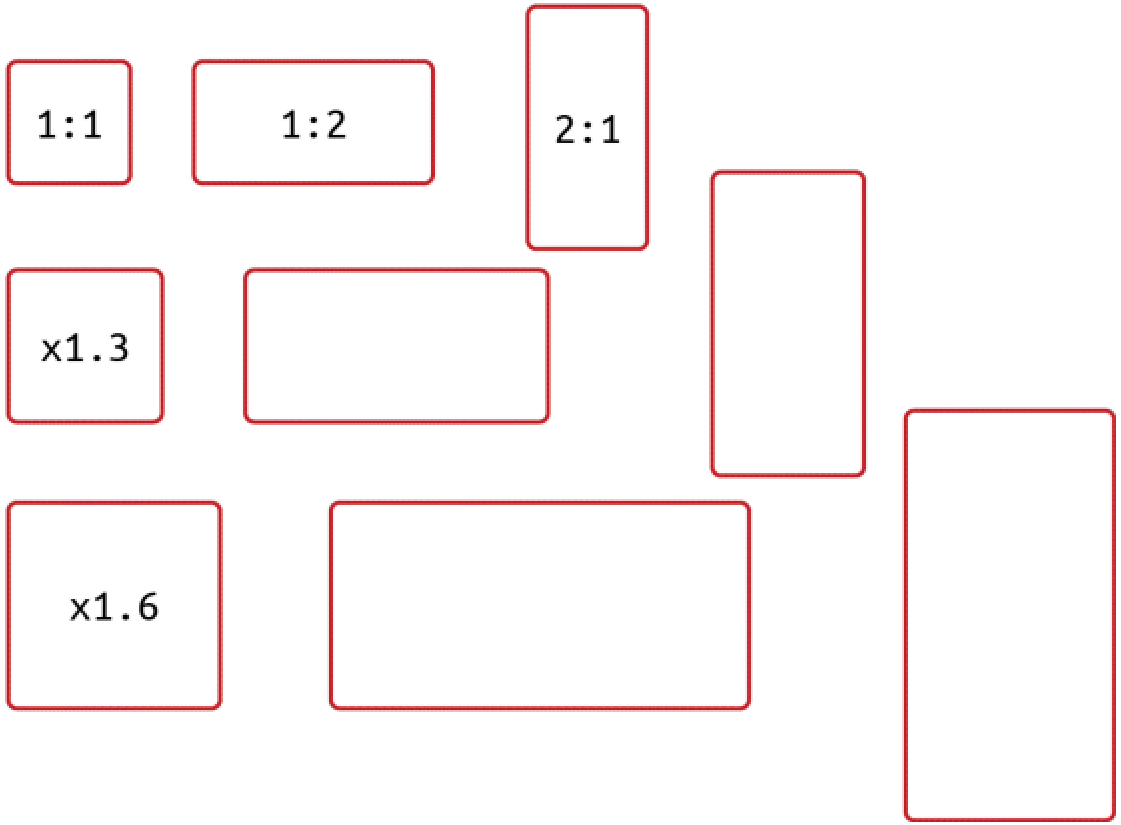
https://learning.oreilly.com/library/view/practical-machine-learning/9781098102357/

Reminder About the Convolutional Base



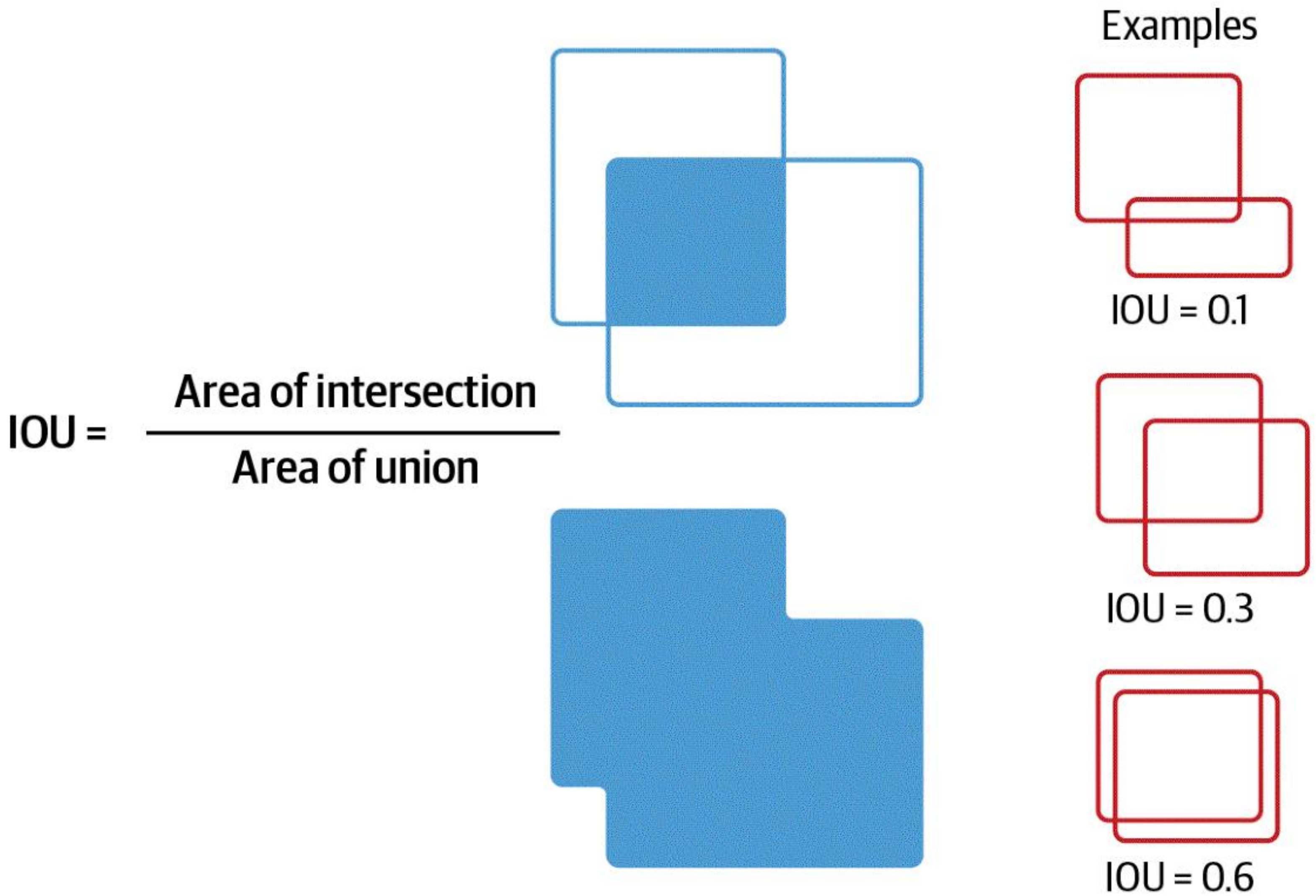
https://learning.oreilly.com/library/view/practical-machine-learning/9781098102357/

Anchor Boxes (varied aspect ratio and size)



https://learning.oreilly.com/library/view/practical-machine-learning/9781098102357/

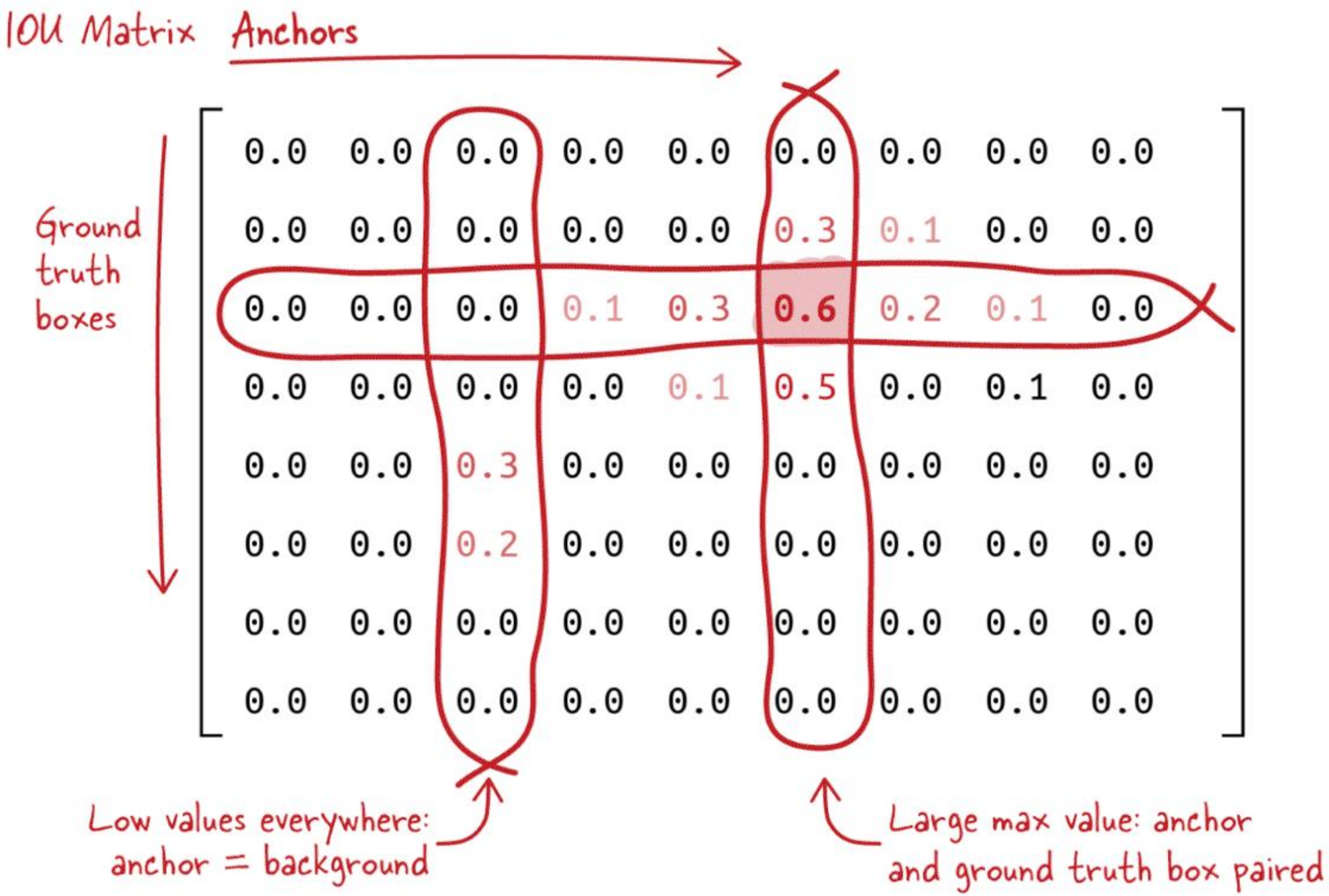
Intersection over Union (IoU)



$$\text{IOU} = \frac{\text{Area of intersection}}{\text{Area of union}}$$

https://learning.oreilly.com/library/view/practical-machine-learning/9781098102357/

Assigning Ground Truth Boxes to Anchors



IOU Matrix Anchors

Ground truth boxes

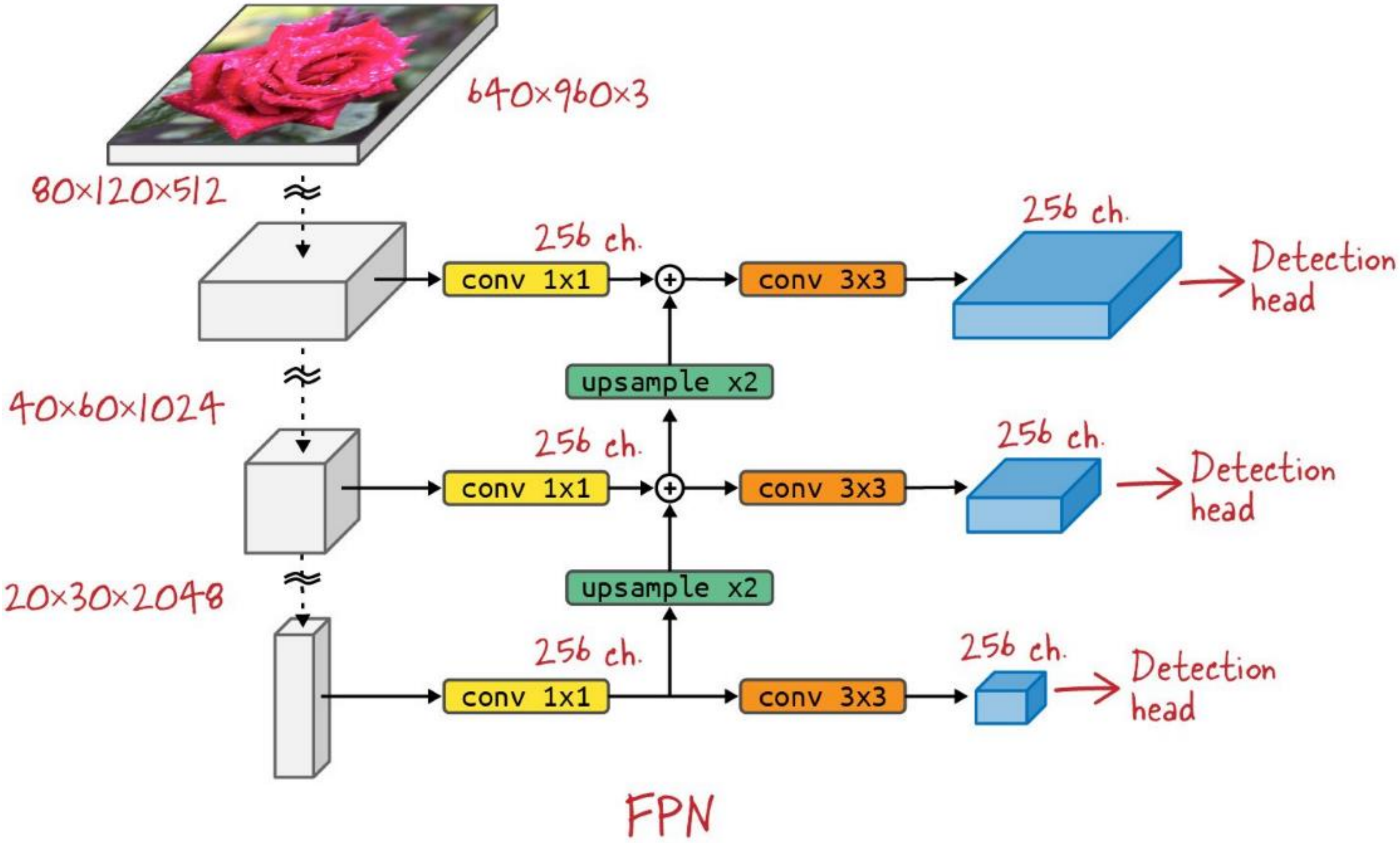
$$\begin{bmatrix}
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.3 & 0.1 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.1 & 0.3 & \mathbf{0.6} & 0.2 & 0.1 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.5 & 0.0 & 0.1 & 0.0 \\
0.0 & 0.0 & 0.3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}$$

Low values everywhere: anchor = background

Large max value: anchor and ground truth box paired

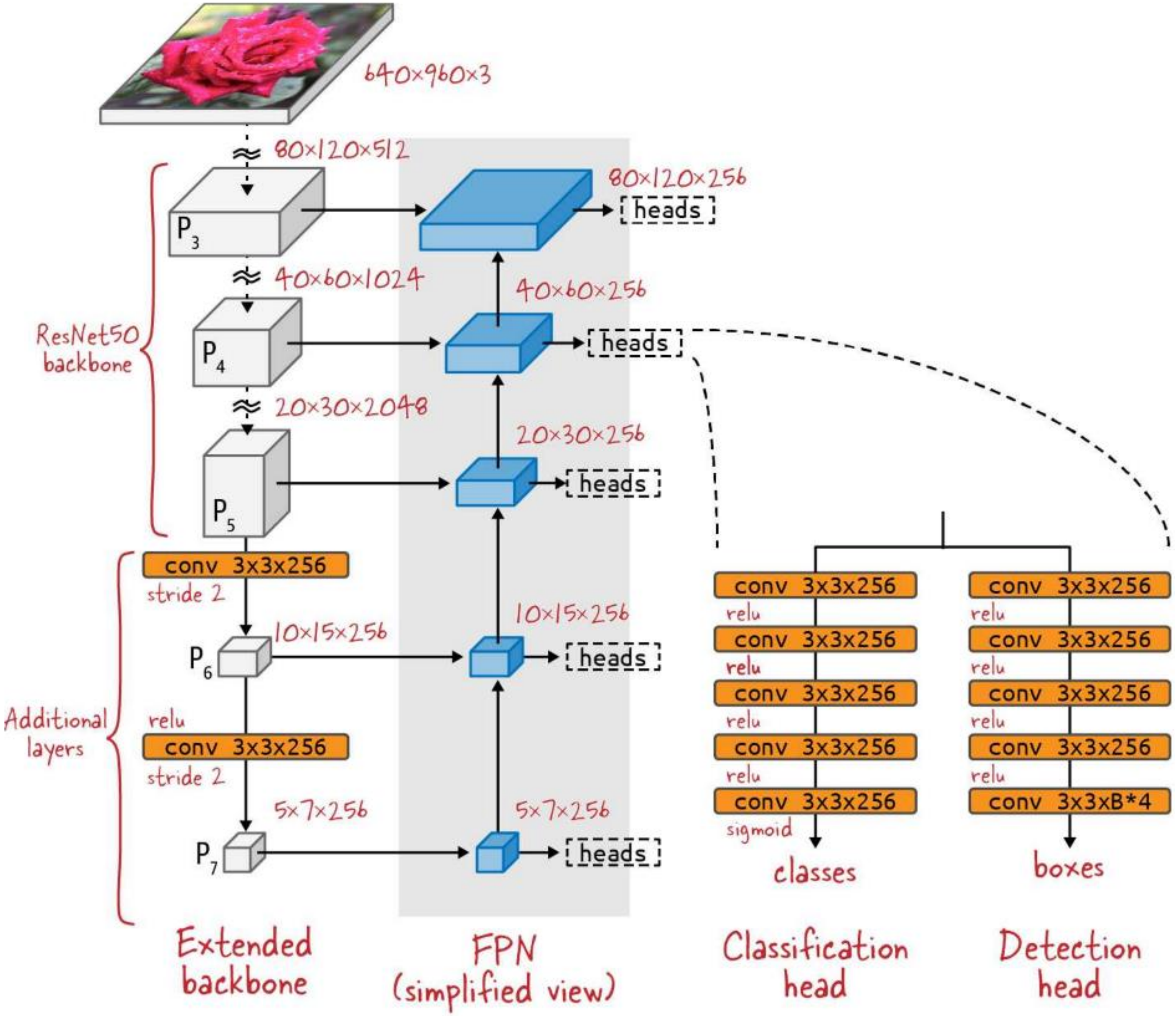
https://learning.oreilly.com/library/view/practical-machine-learning/9781098102357/

Feature Pyramid Network (FPN)



https://learning.oreilly.com/library/view/practical-machine-learning/9781098102357/

RetinaNet



https://paperswithcode.com/method/retinanet

RetinaNet

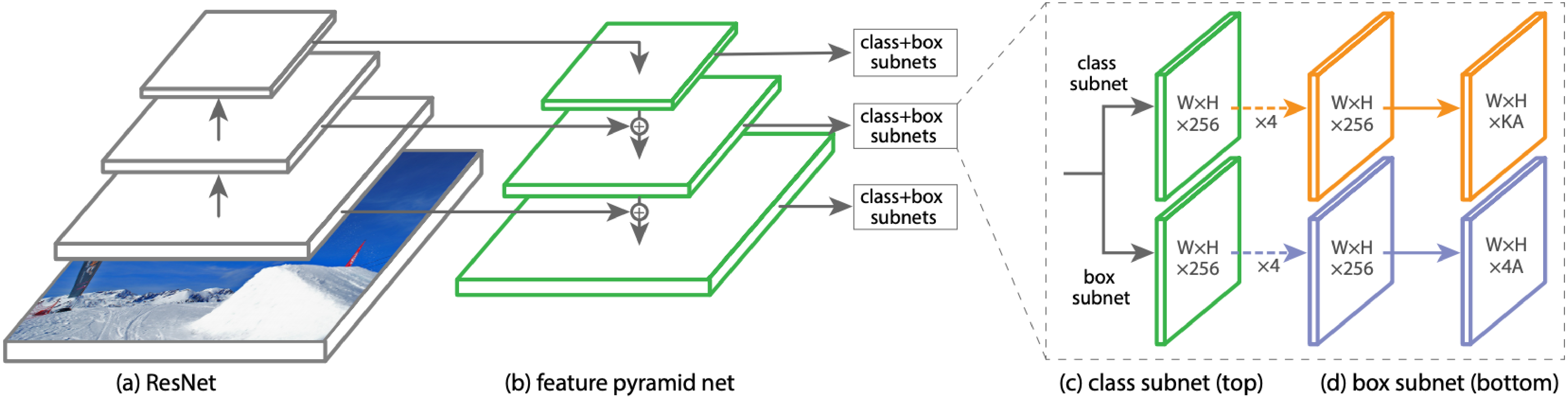


Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN [20] while running at faster speeds.

https://paperswithcode.com/method/focal-loss

Focal Loss

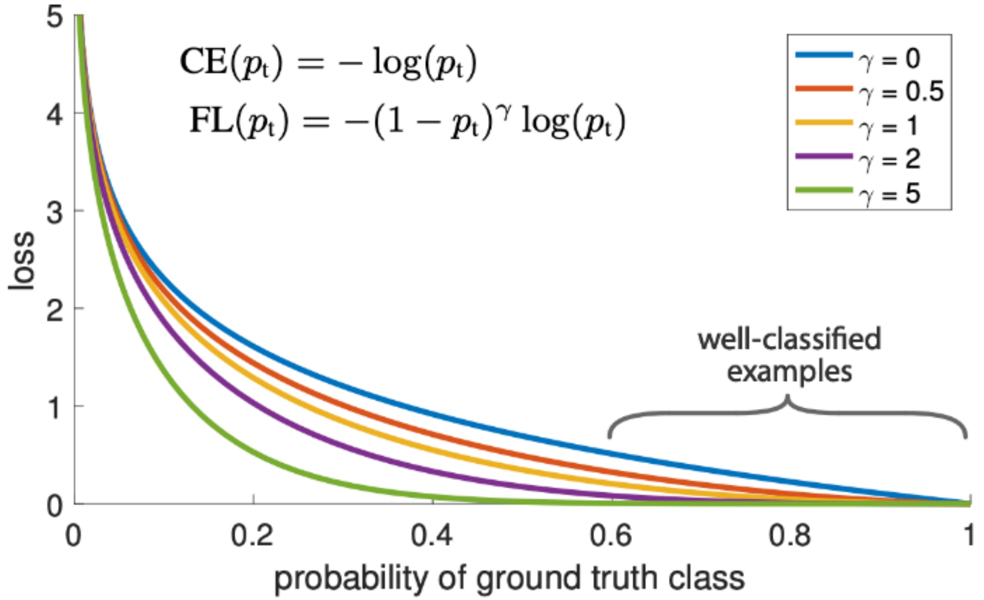


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^{\gamma}$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

https://learning.oreilly.com/library/view/practical-machine-learning/9781098102357/

Non-Max Suppression (NMS)