Deep Learning Tutorial

Esteban Guillen

University of New Mexico

https://github.com/EstebanGuillen/DeepLearningTutorial-PeaceEngineeringConference

SCHOOL OF ENGINEERING

Association for Computing Machinery

The University of New Mexico IEEE Student Branch

Outline

- What is Deep Learning
- Deep Learning Examples
- Neural Networks
  - MNIST Demo
- Convolutional Neural Networks
  - Fashion MNIST Demo

AI vs. ML vs. DL

Artificial Intelligence

Computing systems mimicking Human behavior

Machine Learning

Learning (improving) without being explicitly programmed

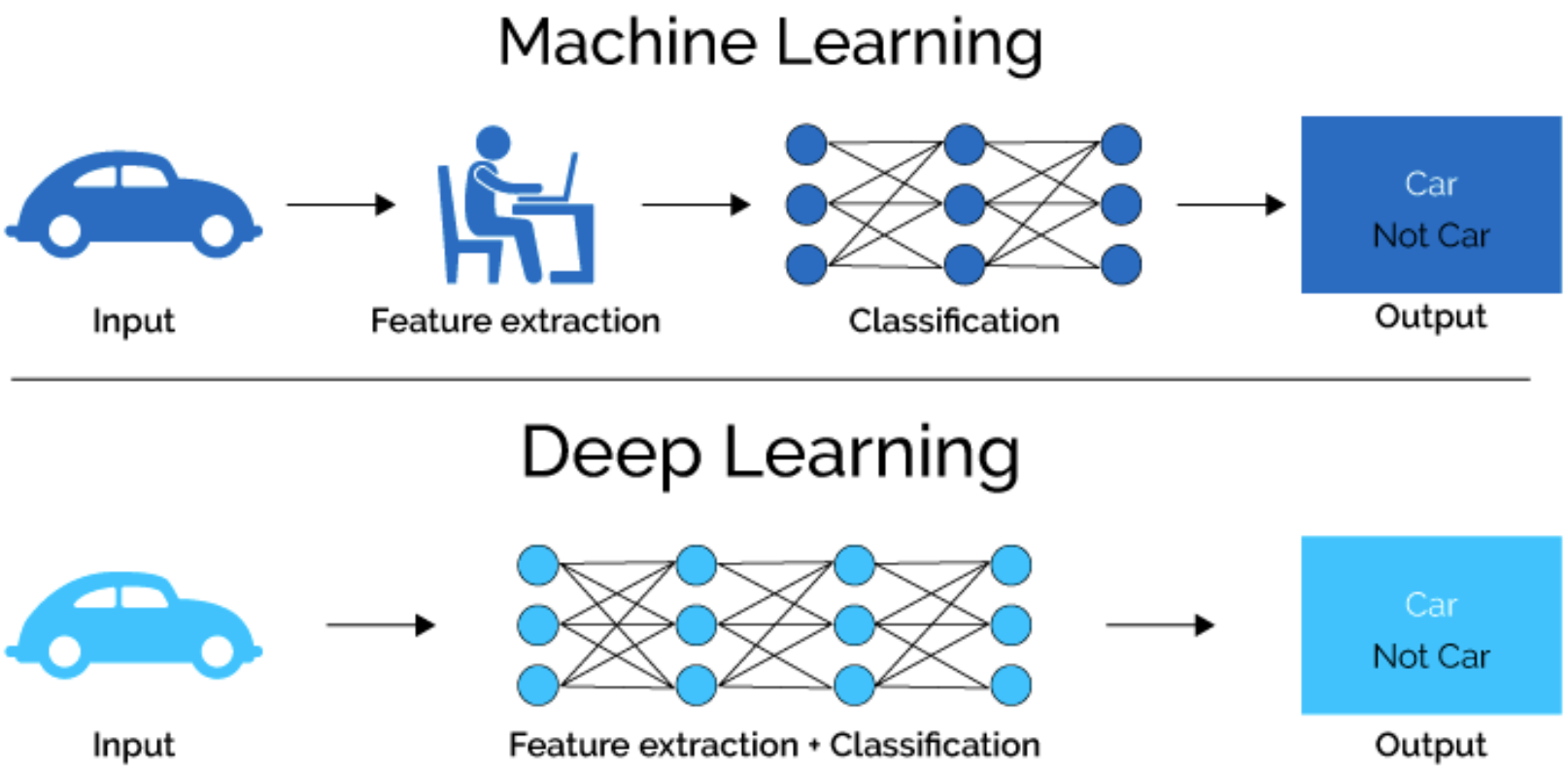
SVM, Logistic Regression, Decision Tree, Linear Regression

Deep Learning

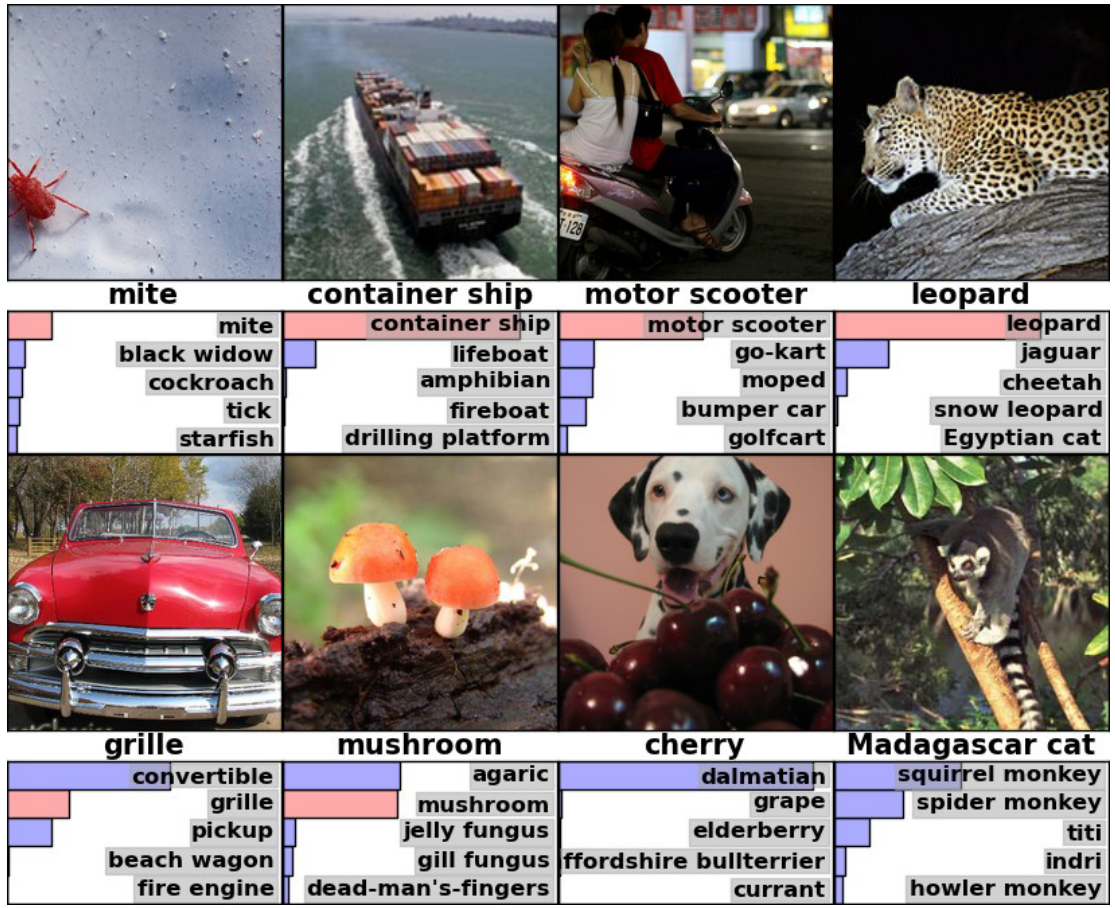
Learning representations directly from data using neural networks

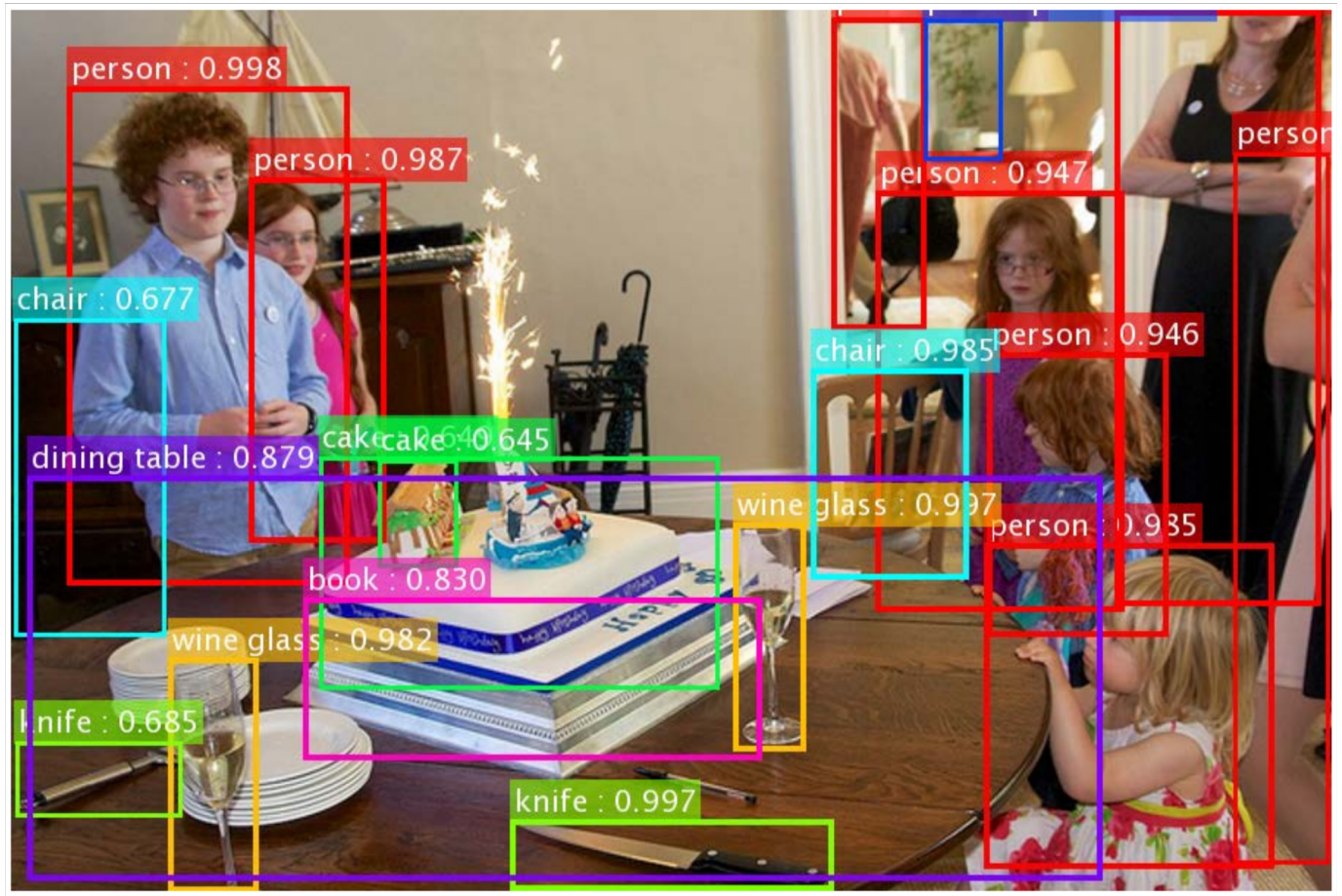
CNN, GAN, RNN,

ML vs DL


Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

Deep Learning Example – Computer Vision



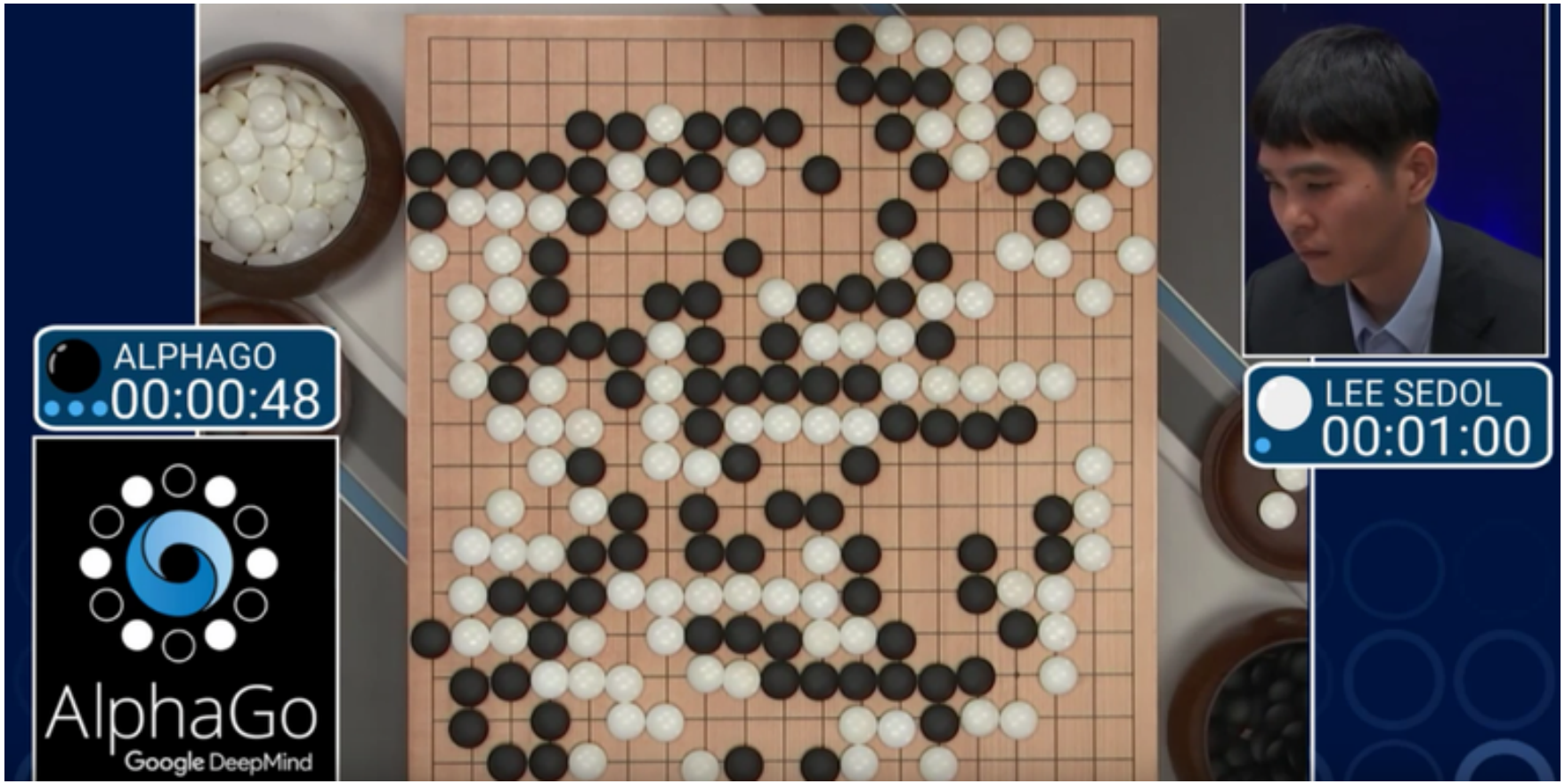


Deep Learning Example – Speech Recognition

Deep Learning Example – NLP

Deep Learning Example – Machine Translation

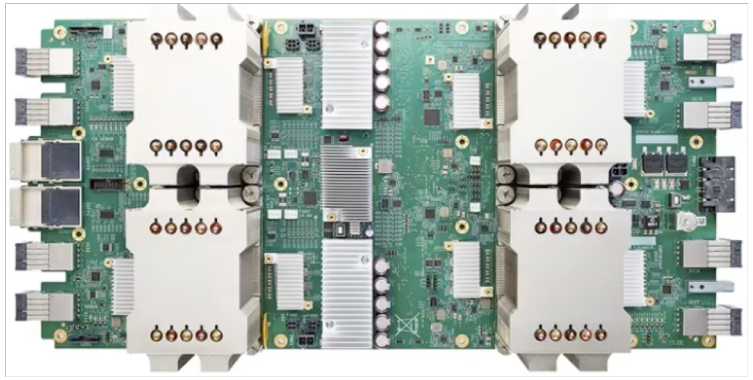
Deep Learning Example – Game Play

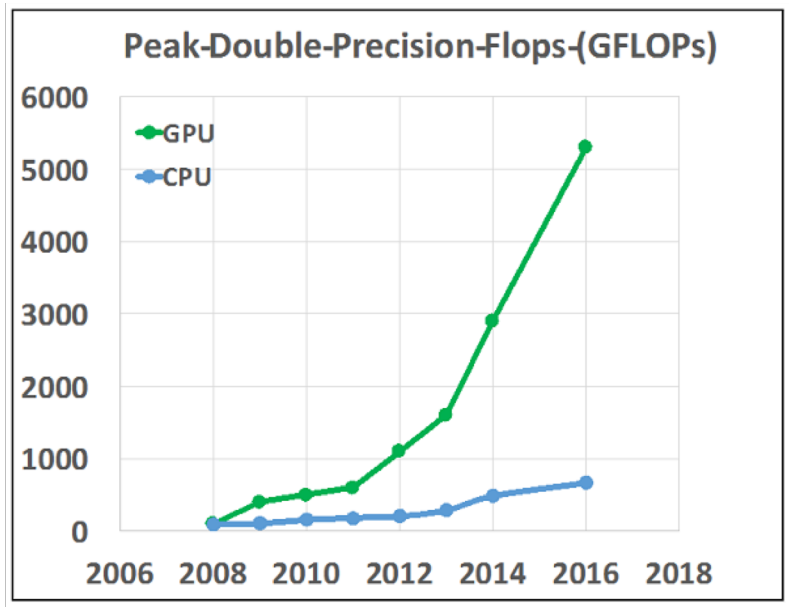


Deep Learning Example – Self-driving cars

Why Deep Learning Now – Big Data

Why Deep Learning Now - Hardware





Why Deep Learning Now - Software

Keras

fast.ai

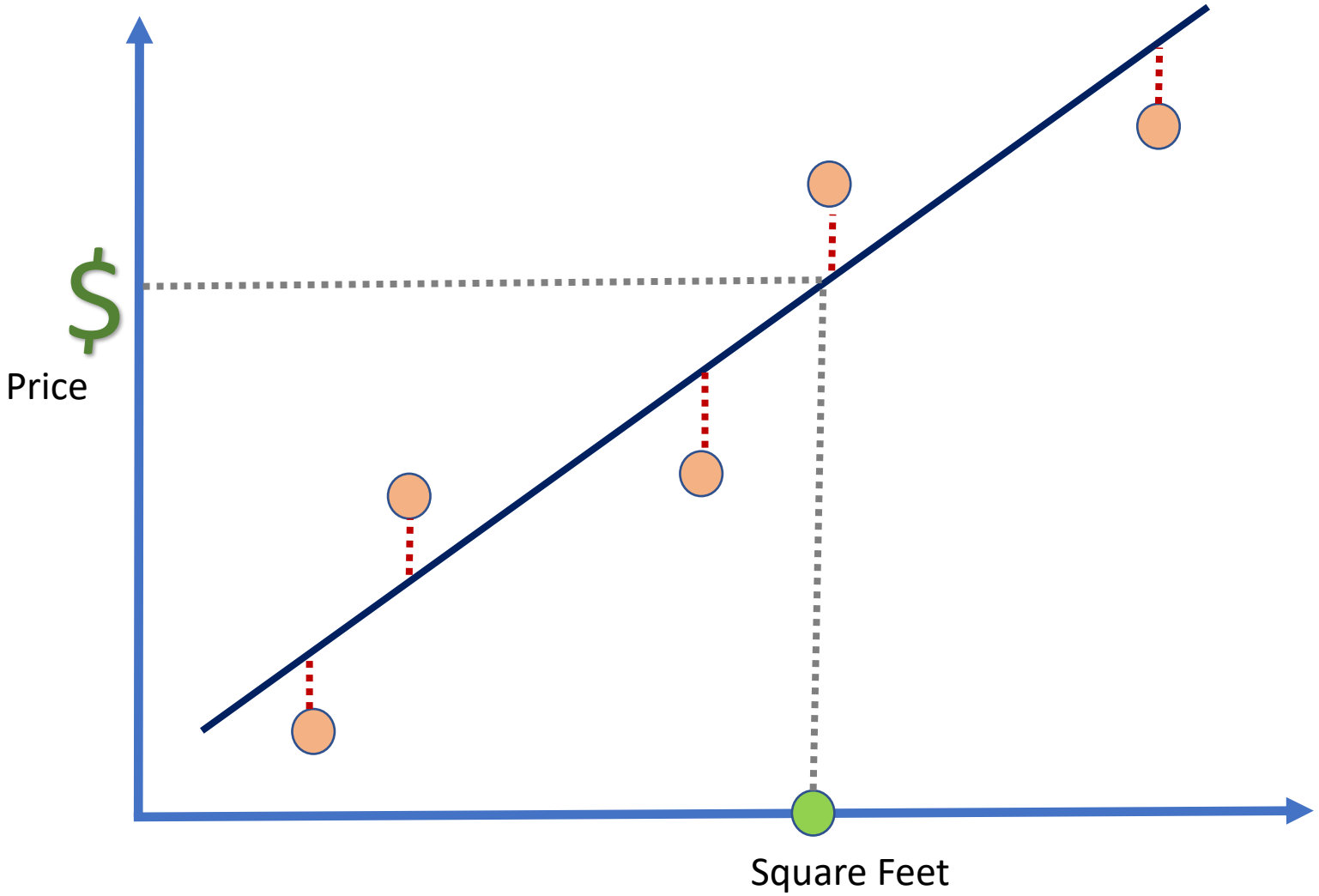
mxnet

TensorFlow

PYTORCH

NVIDIA

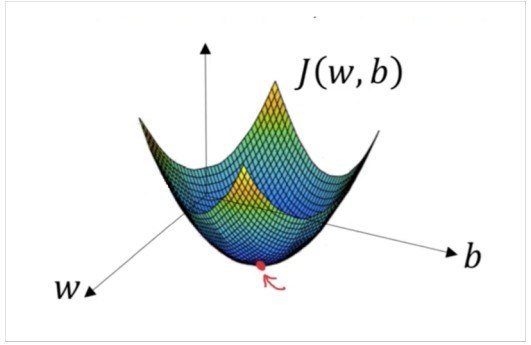
A Learning Example



$$\hat{y} = b + Wx$$

$$J(W, b\,) = \frac{1}{2m}\sum_{i=1}^{m}\left((b + Wx) - y^{i}\right)^{2}$$

$$\underset{W,b}{\text{minimize}}\ J(W, b\,)$$



repeat until convergence {

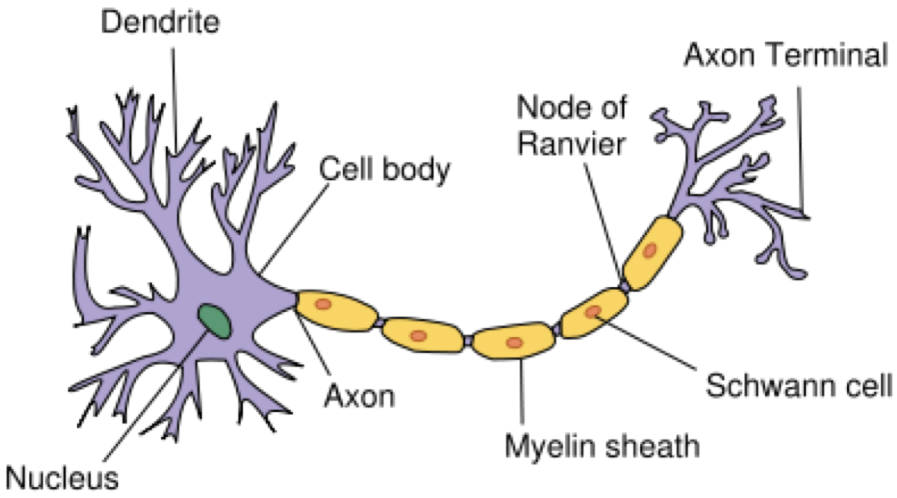
$$b \coloneqq b\ - \alpha \frac{\partial}{\partial b} J(W, b\,)$$

$$\mathrm{W} \coloneqq W - \alpha \frac{\partial}{\partial W} J(W, b\,)$$

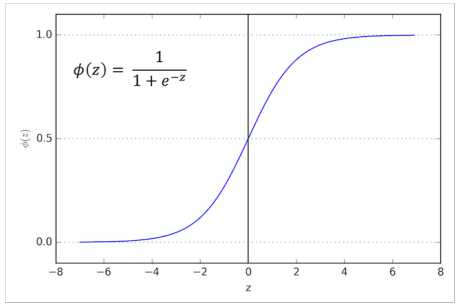
}

Deep Learning – Early Inspiration

Biological Neuron

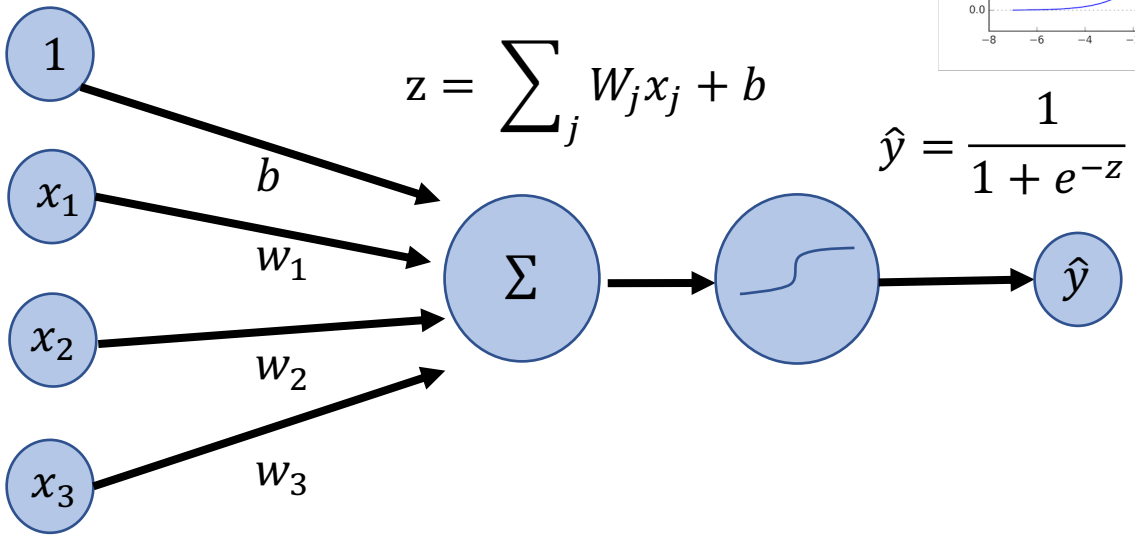


Artificial Neuron



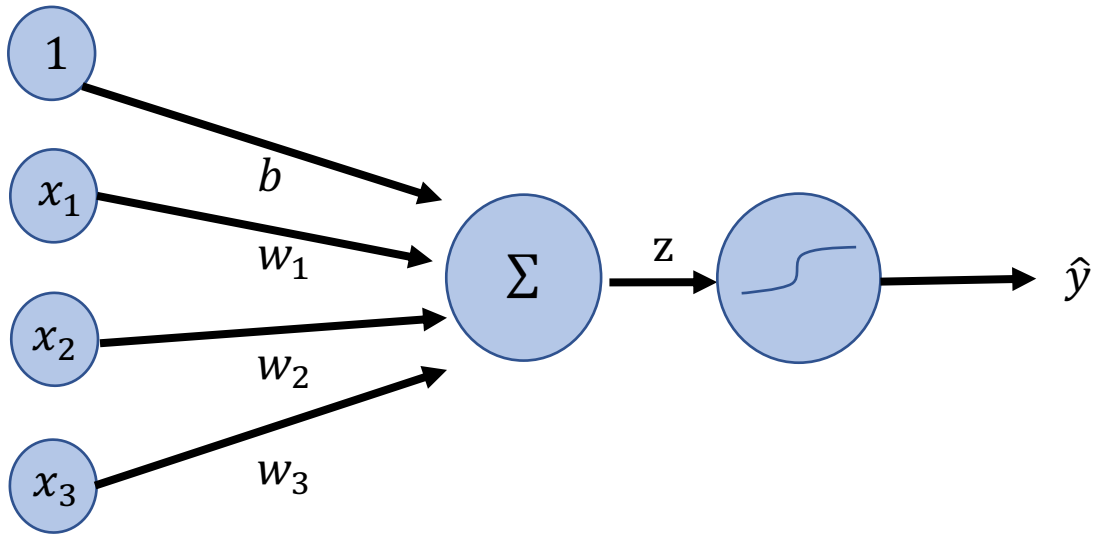
$$z = \sum_j W_j x_j + b$$

$$\hat{y} = \frac{1}{1+e^{-z}}$$



Inputs | Parameters/ Weights | Sum | Non-linearity | Output

Training a Artificial Neuron



$$z = \sum_j W_j x_j + b$$

$$\hat{y} = \frac{1}{1 + e^{-z}}$$

$$J(W, b) = -\frac{1}{m}[\sum_{i=1}^{m} y^i \log(\widehat{y^i}) + (1 - y^i \log(1 - \widehat{y^i})]$$

$$\underset{W,b}{\text{minimize}}\ J(W, b\ )$$

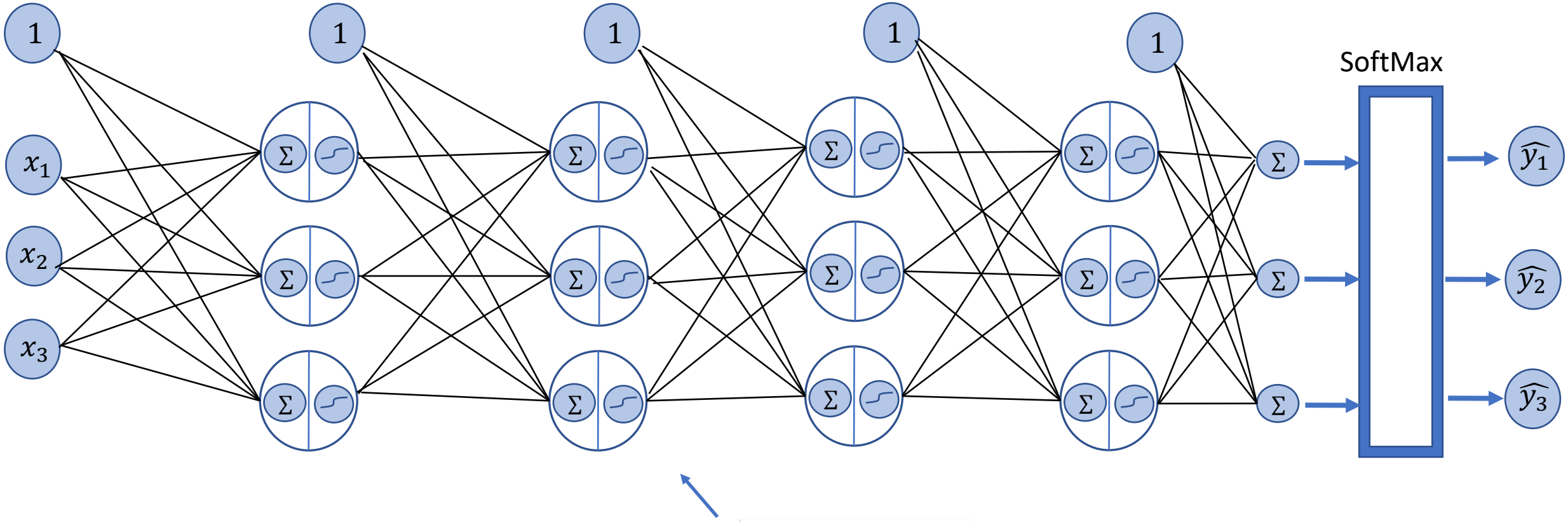
repeat until convergence {

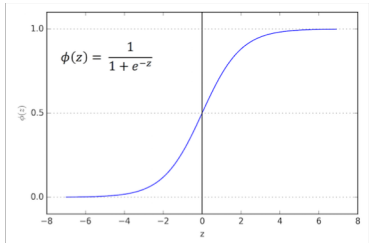
$$b := b\ - \alpha \frac{\partial}{\partial b} J(W, b\ )$$

$$\mathrm{W_j} := W_j - \alpha \frac{\partial}{\partial W_j} J(W, b\ )$$

}

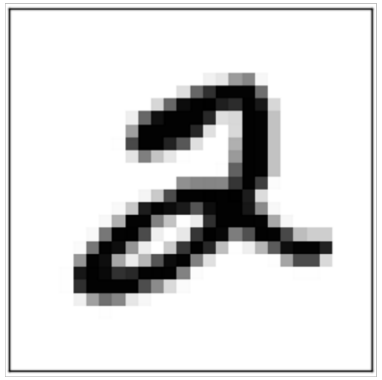
Deep Learning – Fully Connected Network

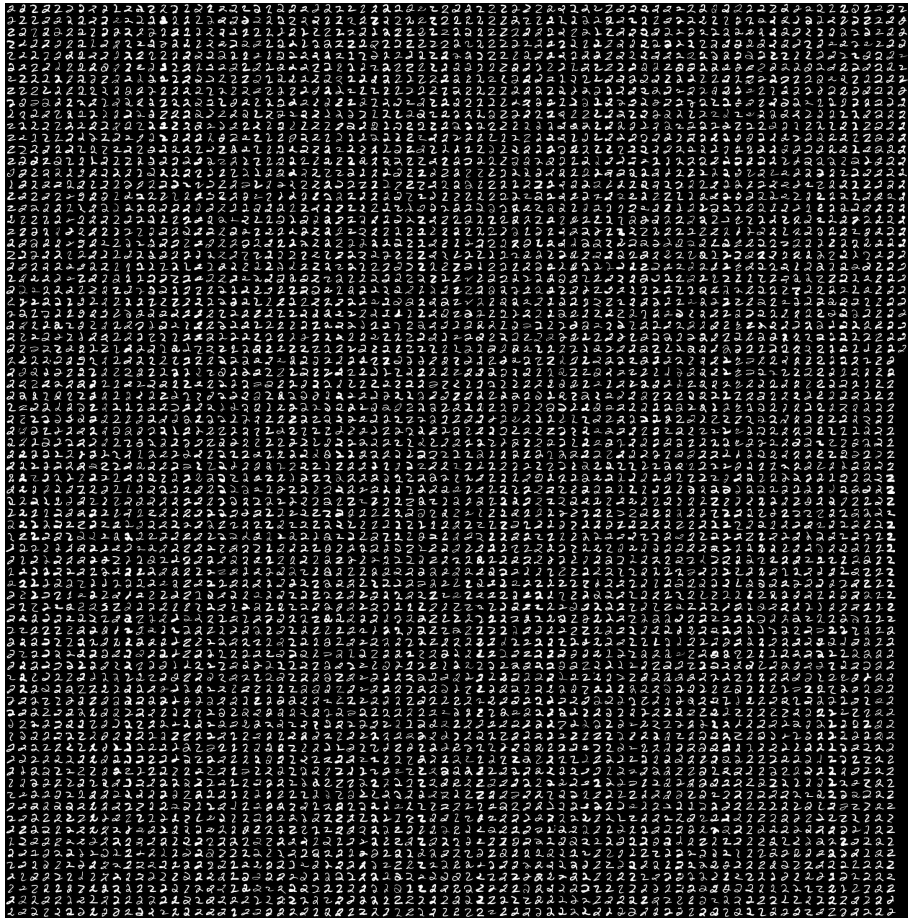




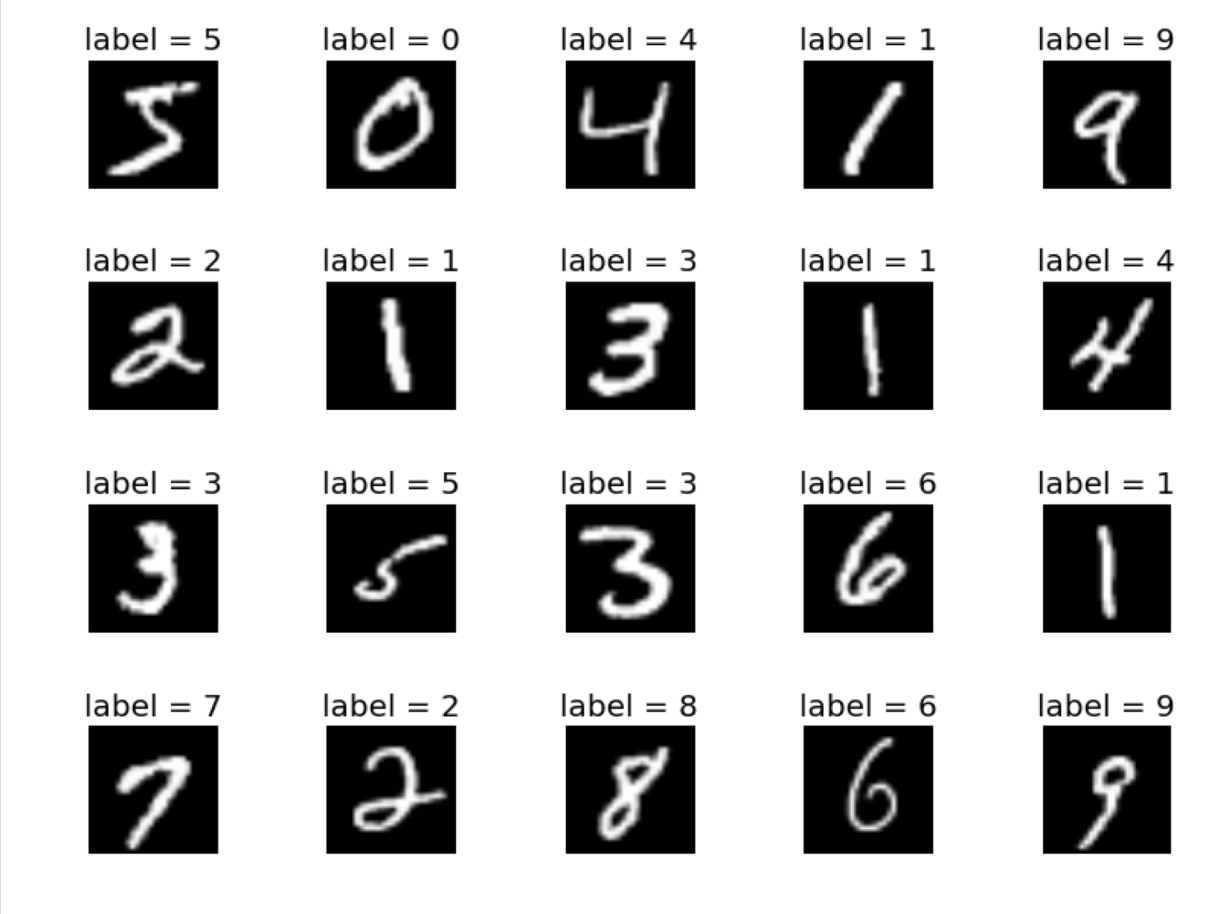
Neural Network Example - MNIST

- Recognize Hard Written Digits (28x28 grey scale)
  - 60,000 training images
  - 10,000 testing images

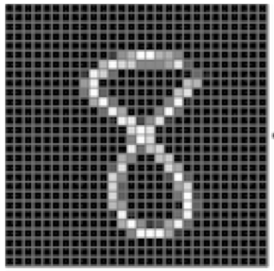




MNIST Data

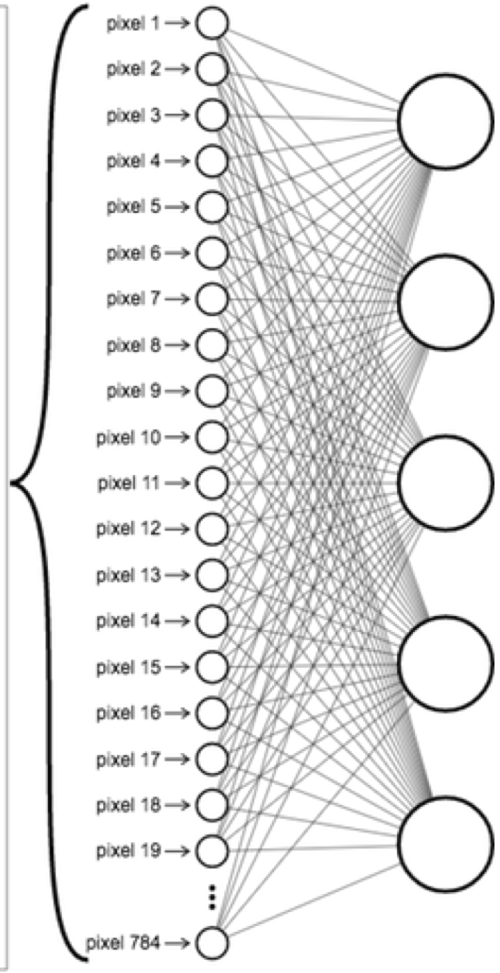

label = 5
label = 0
label = 4
label = 1
label = 9
label = 2
label = 1
label = 3
label = 1
label = 4
label = 3
label = 5
label = 3
label = 6
label = 1
label = 7
label = 2
label = 8
label = 6
label = 9

Flatten Data and feed into network

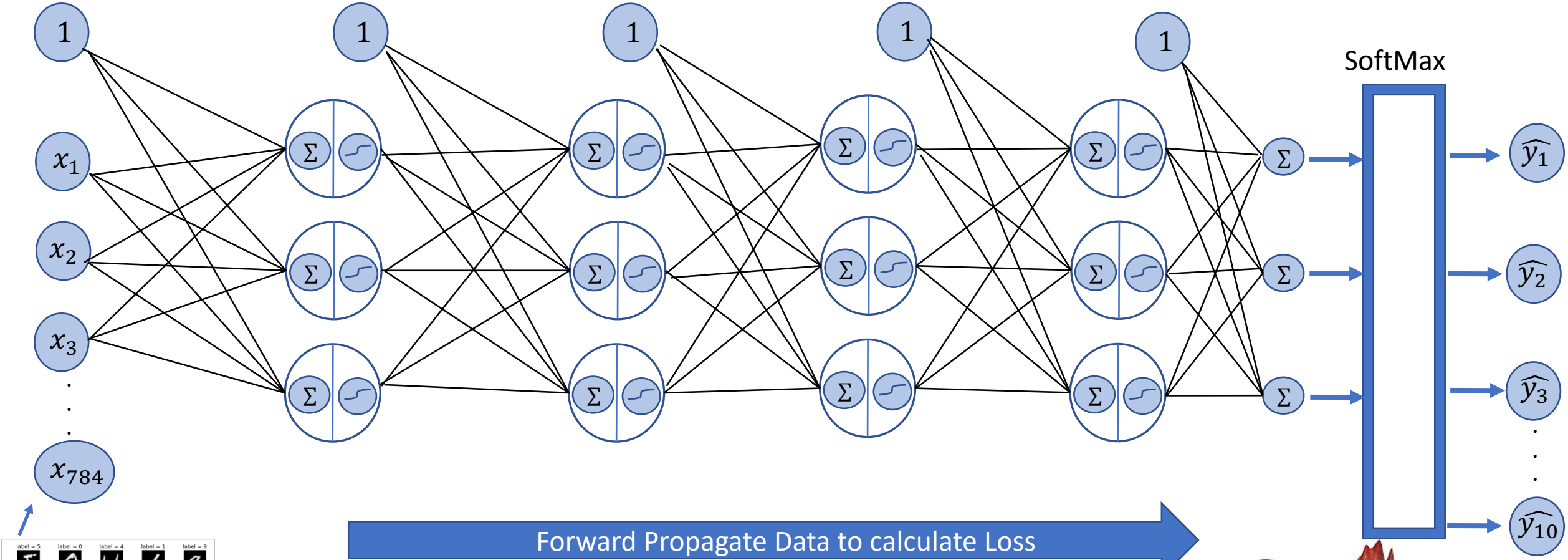


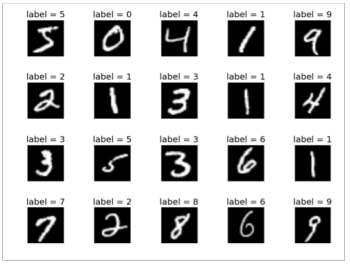
28 x 28
784 pixels

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	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	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	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	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	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



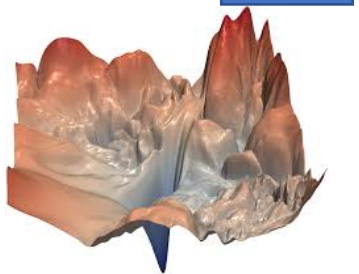
Training a Neural Network





Forward Propagate Data to calculate Loss

Back Propagate the Loss to calculate Gradient



Intro to Keras

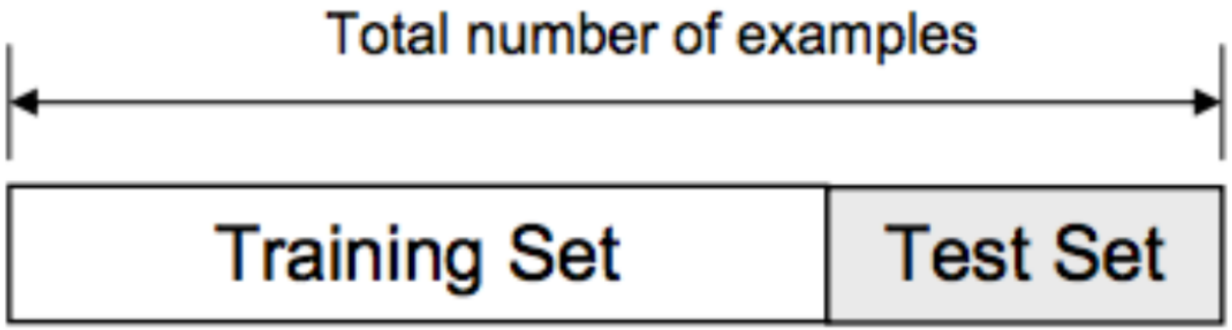
- High-level neural network library
- Written in Python
- Sits on top of Tensorflow
- Allows for quick prototyping
- Run seamlessly on CPU, GPU or TPU

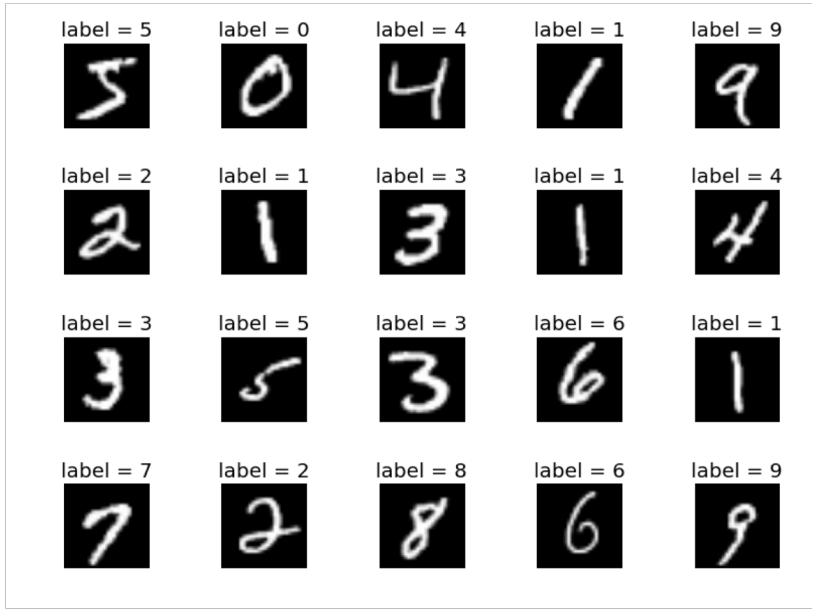
Demo – Train Neural Network

- MNIST Data
- Keras
- Running on Google Colab (GPU K80)
  - https://colab.research.google.com (free)
  - https://colab.research.google.com/drive/1Ys-wm7uY5ZUYsAWsCr8NnlLAUbR7Arej

Demo - Load libraries and data

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

mnist = tf.keras.datasets.mnist #28x28 images

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```



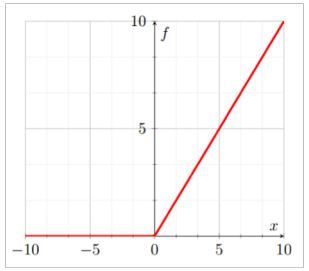


Demo – Normalize data

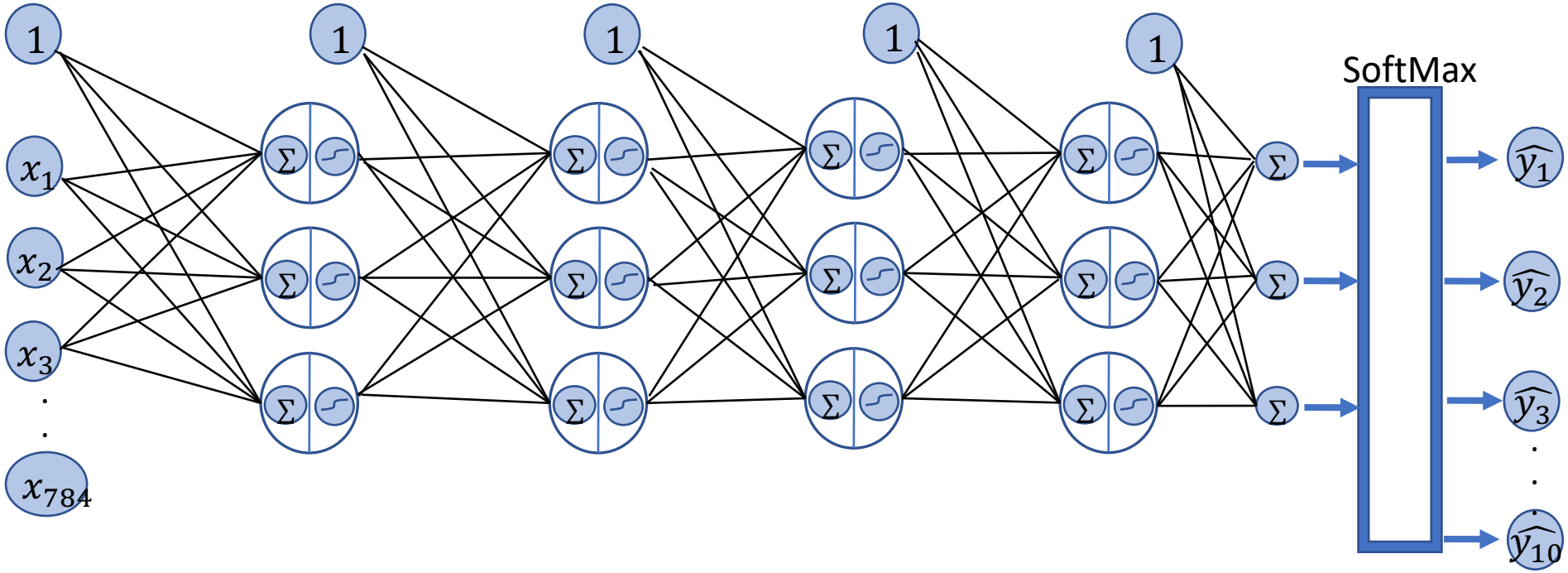
```
x_train = tf.keras.utils.normalize(x_train,axis=1)
x_test = tf.keras.utils.normalize(x_test,axis=1)
```

Now each data input ranges from 0 to 1 instead of 0 to 255

Demo - Construct the Neural Network




```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```



Demo - Train and Test the Neural Network

```
model.fit(x_train,y_train, epochs=5)
```

```
Epoch 1/5
60000/60000 [==============================] - 11s 186us/step - loss: 0.2665 - acc: 0.9229
Epoch 2/5
60000/60000 [==============================] - 9s 157us/step - loss: 0.1088 - acc: 0.9667
Epoch 3/5
60000/60000 [==============================] - 10s 160us/step - loss: 0.0727 - acc: 0.9769
Epoch 4/5
60000/60000 [==============================] - 10s 158us/step - loss: 0.0532 - acc: 0.9830
Epoch 5/5
60000/60000 [==============================] - 9s 158us/step - loss: 0.0402 - acc: 0.9865
<tensorflow.python.keras.callbacks.History at 0x7fded32dae48>
```

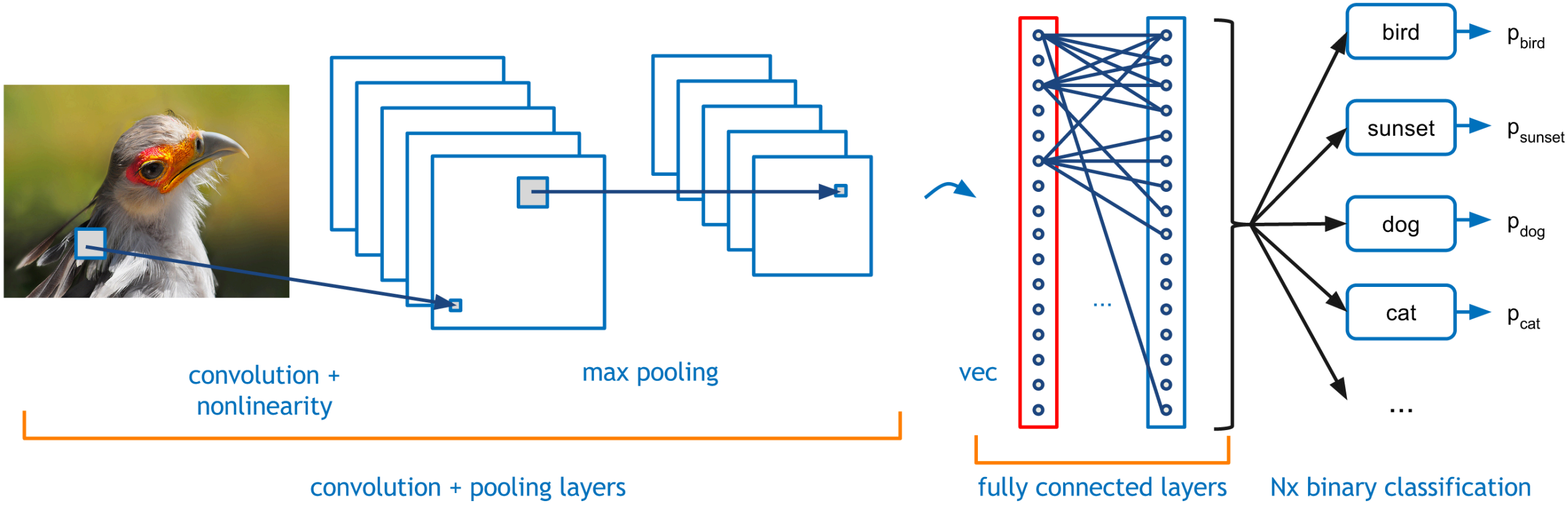
```
val_loss, val_acc = model.evaluate(x_test,y_test)
```

```
10000/10000 [==============================] - 1s 72us/step
```

```
print(val_loss, val_acc)
```

```
0.09539085898089689 0.9724
```

Convolutional Neural Networks (CNN)



Convolution Operation

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Input

*

1	0	-1
1	0	-1
1	0	-1

Filter (Kernel)

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

Feature map

Source: Andrew Ng

Vertical Edge Detection

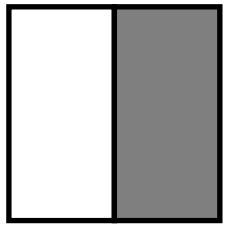
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

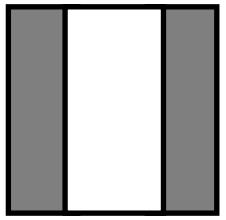
1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



*



Source: Andrew Ng

Vertical/Horizontal Edge Detection

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$*$

1	1	1
0	0	0
-1	-1	-1

$=$

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Source: Andrew Ng

Learning to detect edges

1	0	-1
1	0	-1
1	0	-1

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

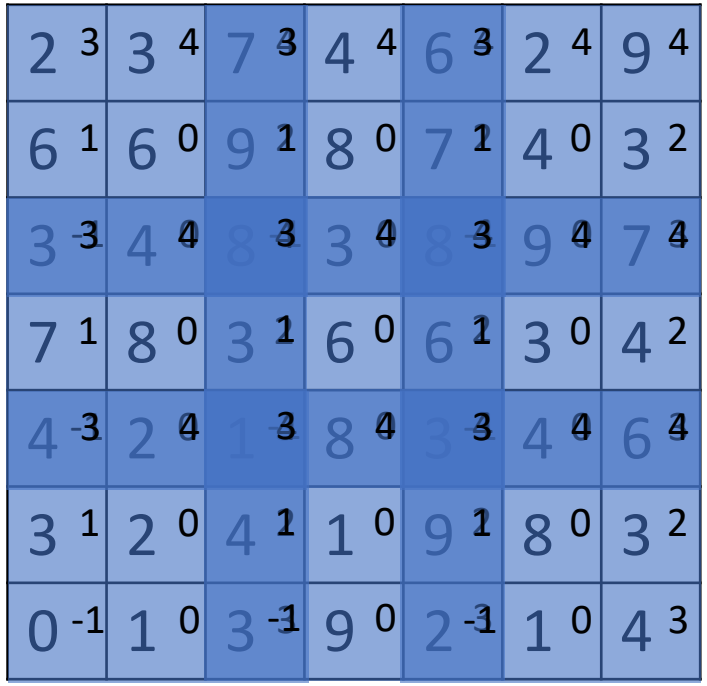
$*$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$=$

Source: Andrew Ng

Strided convolution



7×7

$*$

3	4	4
1	0	2
-1	0	3

3×3

$=$

91	100	83
69	91	127
44	71	74

3×3

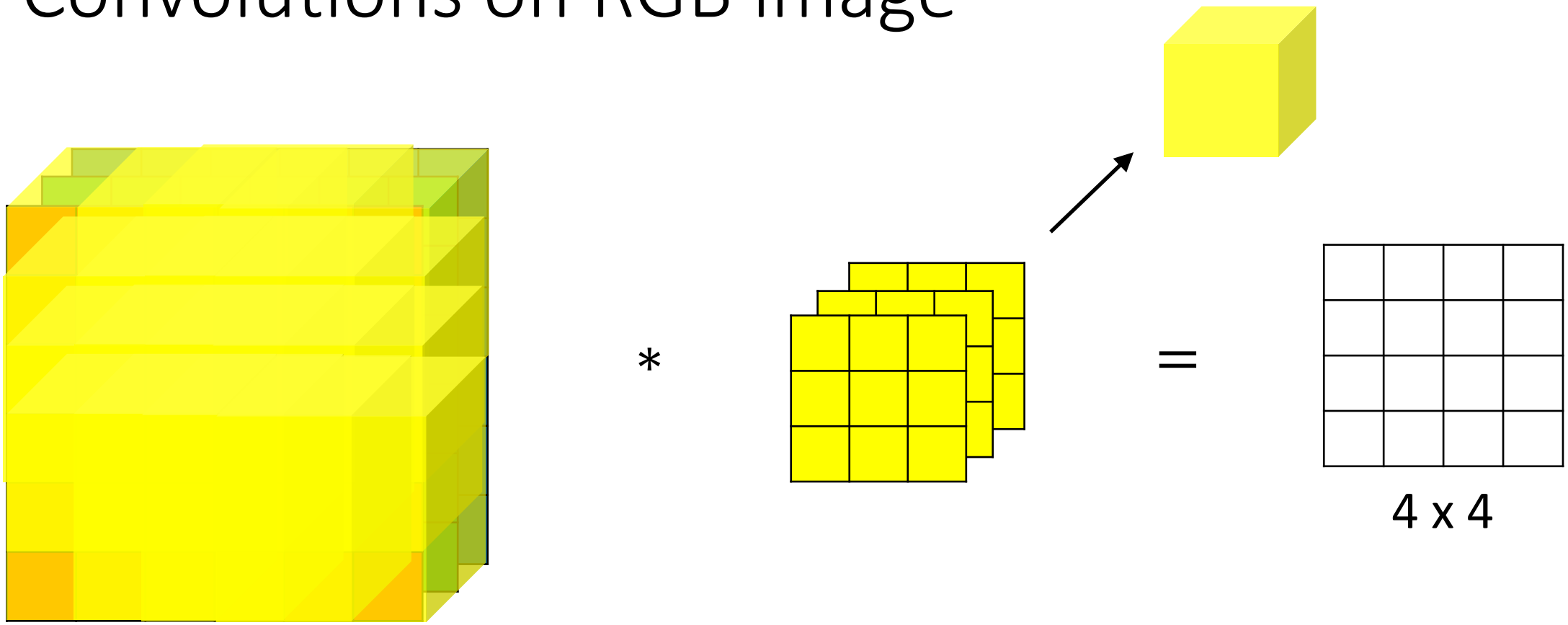
Source: Andrew Ng

Zero Padding

0	0	0	0	0	0	0	0
0	3	0	1	2	7	4	0
0	1	5	8	9	3	1	0
0	2	7	2	5	1	3	0
0	0	1	3	1	7	8	0
0	4	2	1	6	2	8	0
0	2	4	5	2	3	9	0
0	0	0	0	0	0	0	0

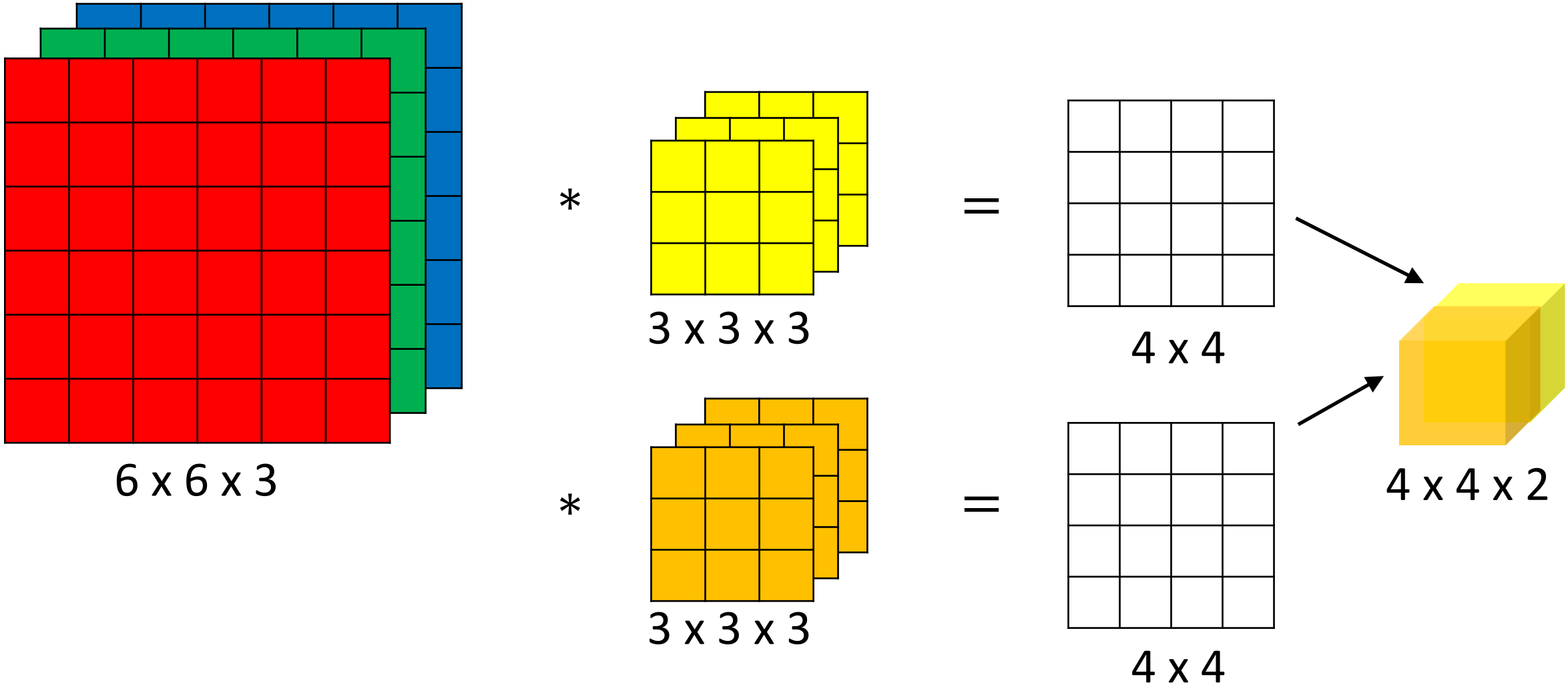
Source: Andrew Ng

Convolutions on RGB image



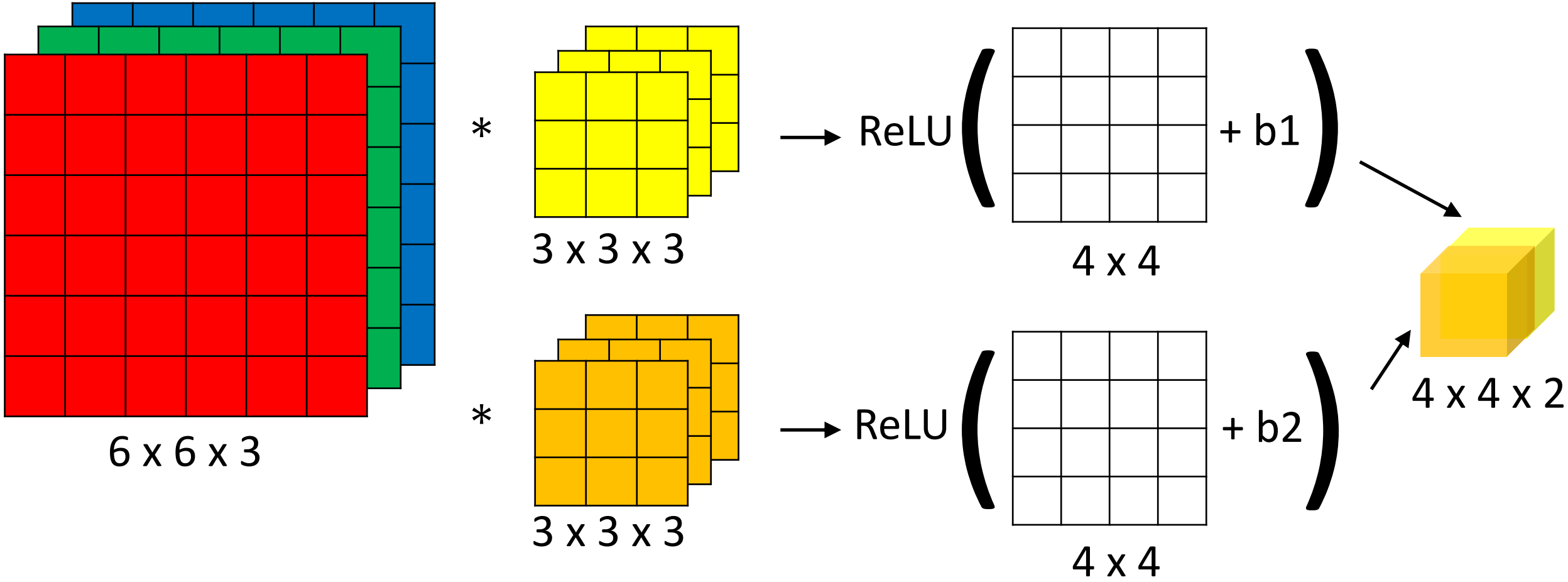
Source: Andrew Ng

Multiple filters



Source: Andrew Ng

Single Convolutional Layer



Source: Andrew Ng

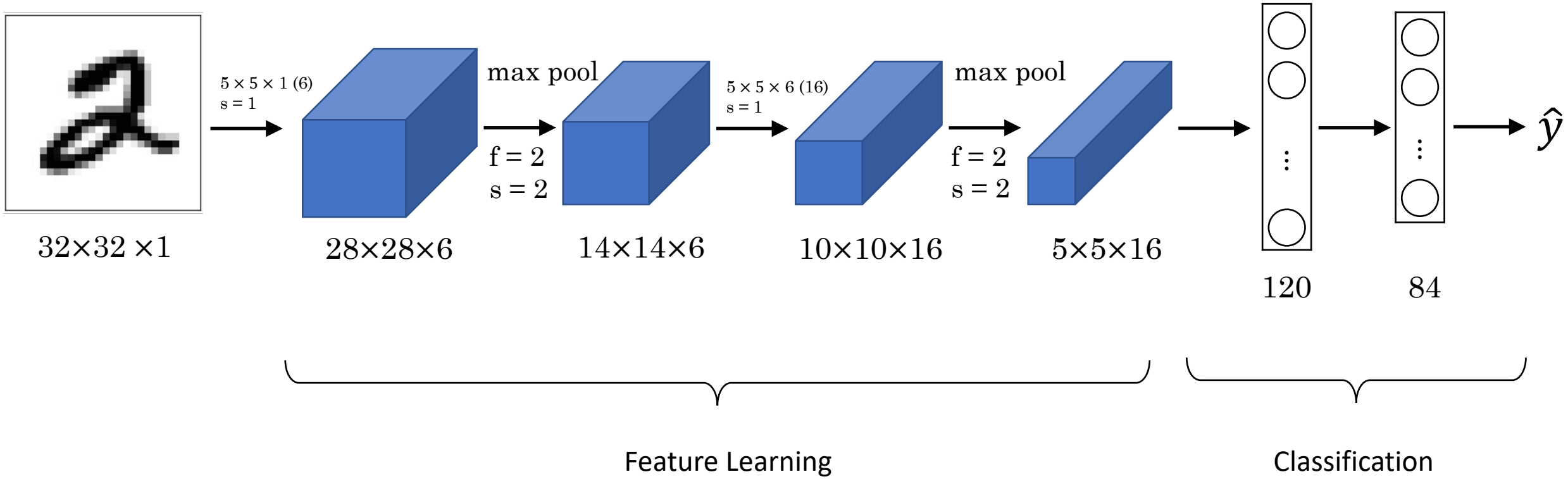
Max pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

9	2
6	3

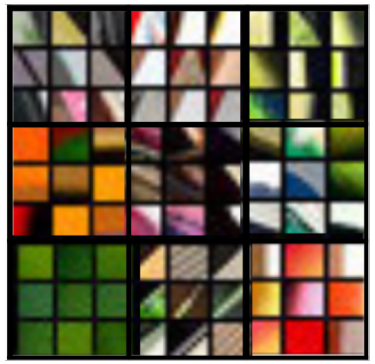
Source: Andrew Ng

Convolutional Neural Network

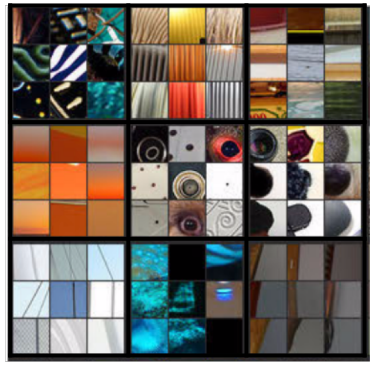


Source: Andrew Ng

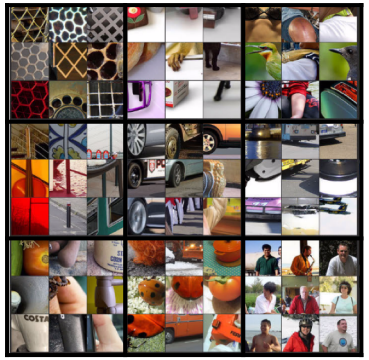
Visualizing deep layers



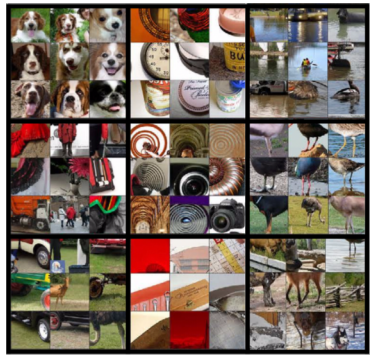
Layer 1



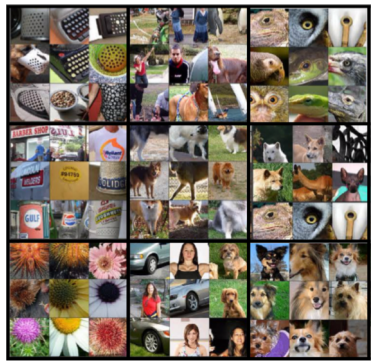
Layer 2



Layer 3



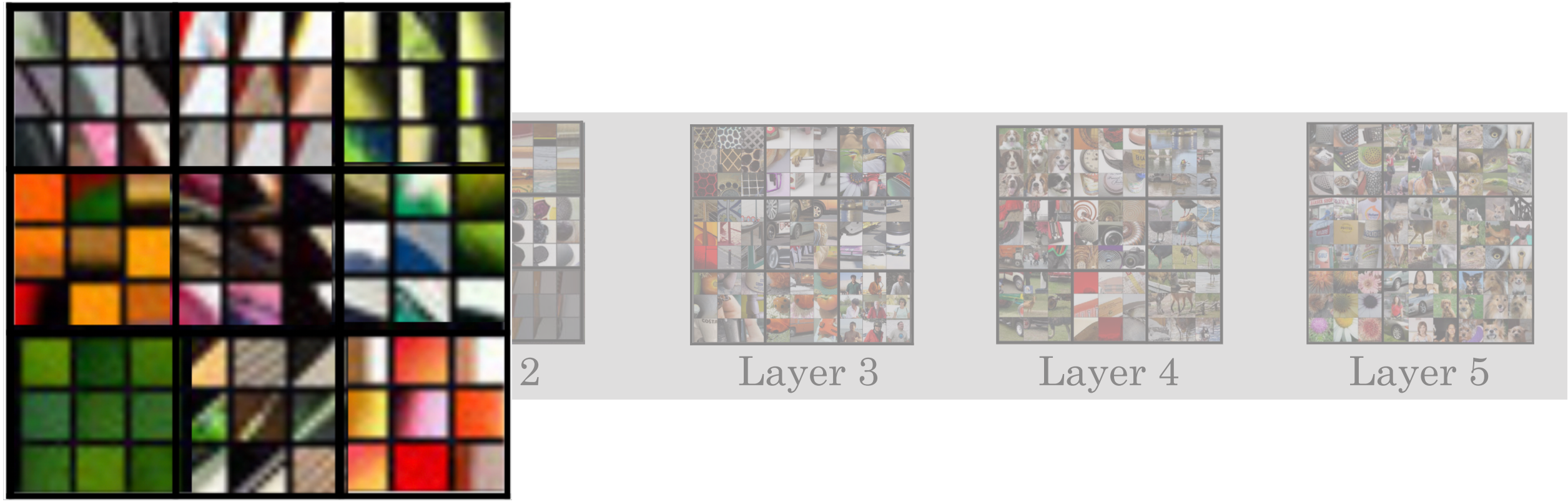
Layer 4



Layer 5

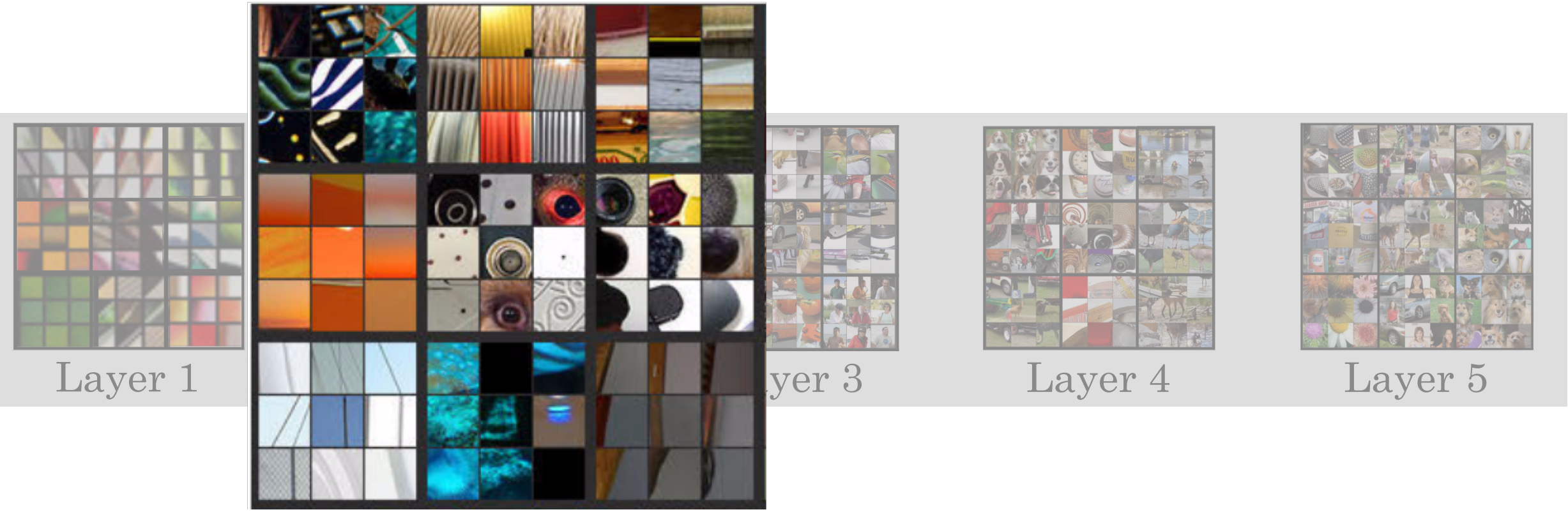
Source: Andrew Ng

Visualizing deep layers: Layer 1



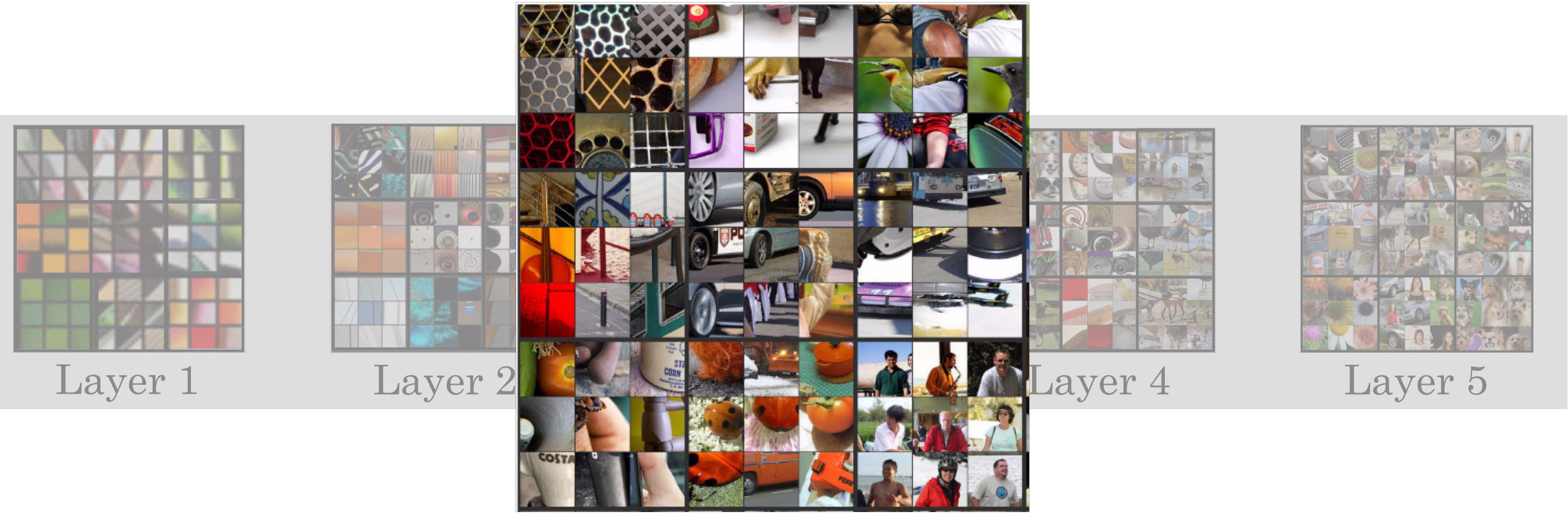
Source: Andrew Ng

Visualizing deep layers: Layer 2



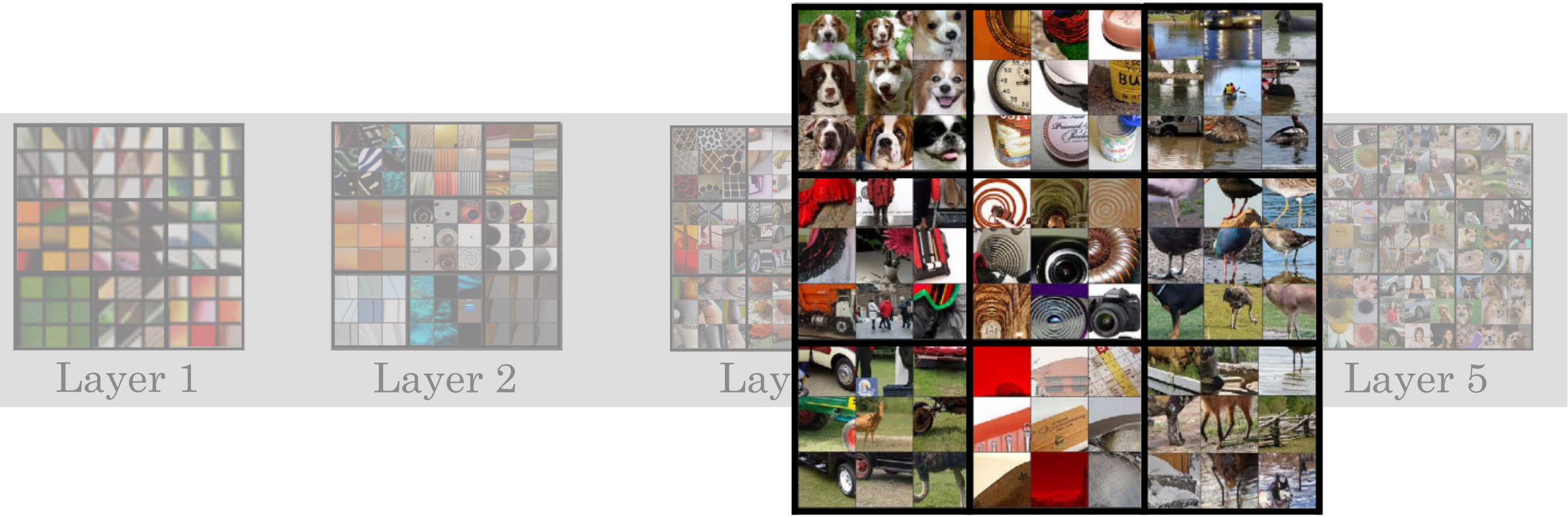
Source: Andrew Ng

Visualizing deep layers: Layer 3



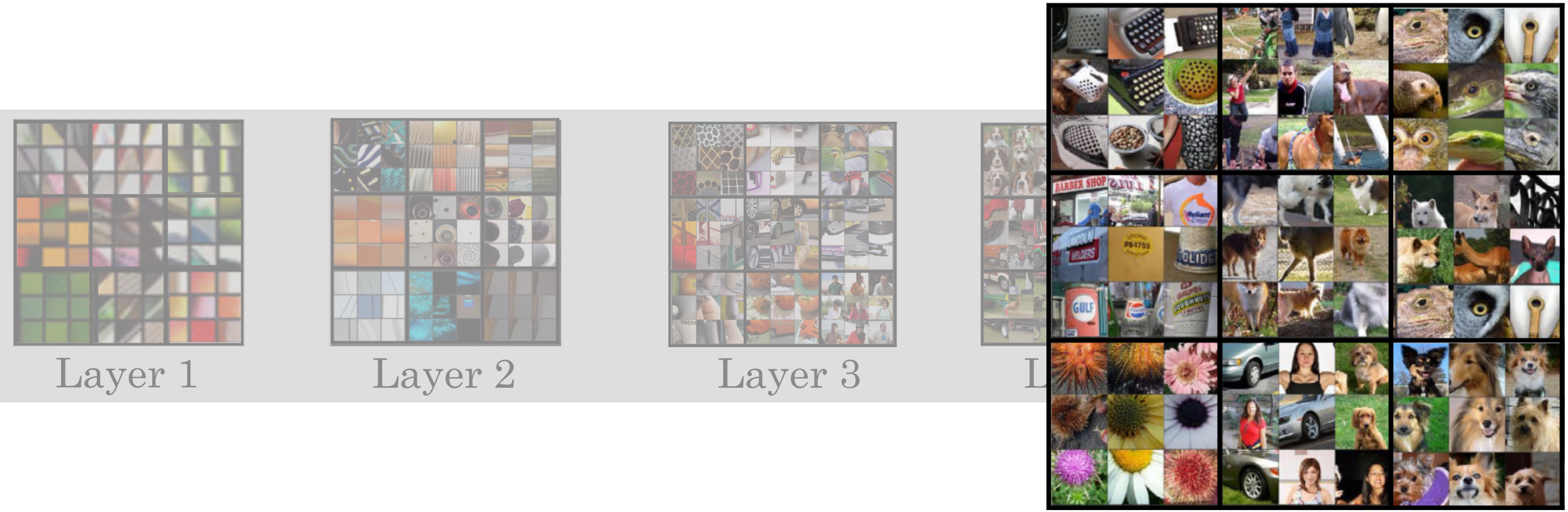
Source: Andrew Ng

Visualizing deep layers: Layer 4



Source: Andrew Ng

Visualizing deep layers: Layer 5



Source: Andrew Ng

Demo – Train Convolutional Neural Network

- Fashion MNIST Data
- Keras
- Running on Google Colab (GPU TPU)
  - https://colab.research.google.com (free)
  - https://colab.research.google.com/drive/1kQWS89EnID2eqpm68dS0dhuNUW_nbp8Z

Demo – Load libraries

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os

from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

from tensorflow.keras.utils import to_categorical
import tensorflow.keras.datasets.fashion_mnist as ds
```

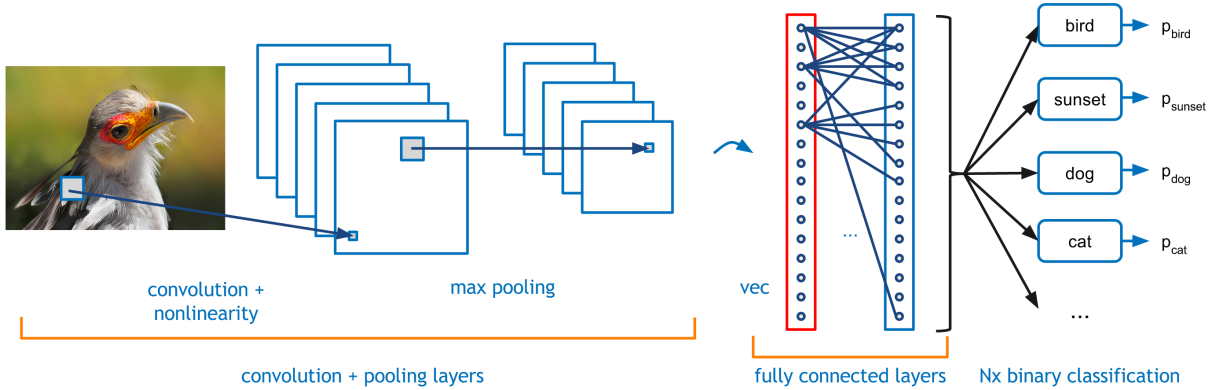
Demo – Load data

```
(train_images, train_labels),(test_images, test_labels) = ds.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Demo – Construct the CNN

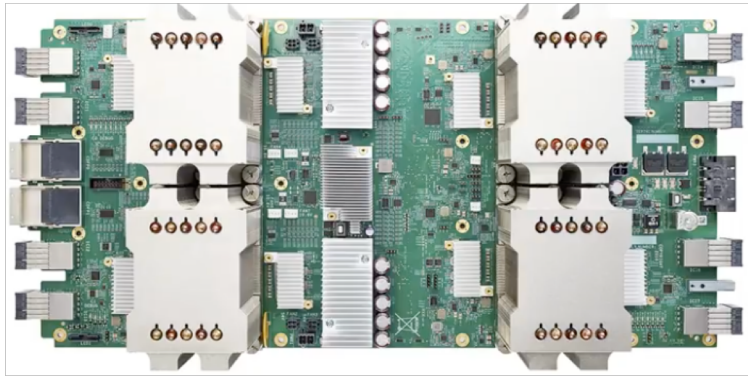
```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```



Demo – Setup the TPU instance

```
TPU_WORKER = 'grpc://' + os.environ['COLAB_TPU_ADDR']
tf.logging.set_verbosity(tf.logging.ERROR)

tpu_model = tf.contrib.tpu.keras_to_tpu_model(
    model,
    strategy=tf.contrib.tpu.TPUDistributionStrategy(
        tf.contrib.cluster_resolver.TPUClusterResolver(TPU_WORKER)))
```



Demo – Train

```
history = tpu_model.fit(train_images, train_labels,
                        epochs=20,
                        batch_size=128 * 8)
```

```
Epoch 1/20
60000/60000 [==============================] - 11s 186us/step - loss: 1.1575 - acc: 0.5836
Epoch 2/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.7061 - acc: 0.7379
Epoch 3/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.5649 - acc: 0.7875
Epoch 4/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.4991 - acc: 0.8145
Epoch 5/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.4503 - acc: 0.8355
```

Demo – Evaluate

```
tpu_model.evaluate(test_images, test_labels, batch_size=128 * 8)
```

```
10000/10000 [==============================] - 4s 389us/step
[0.28891048345565795, 0.8972999990463257]
```

Questions