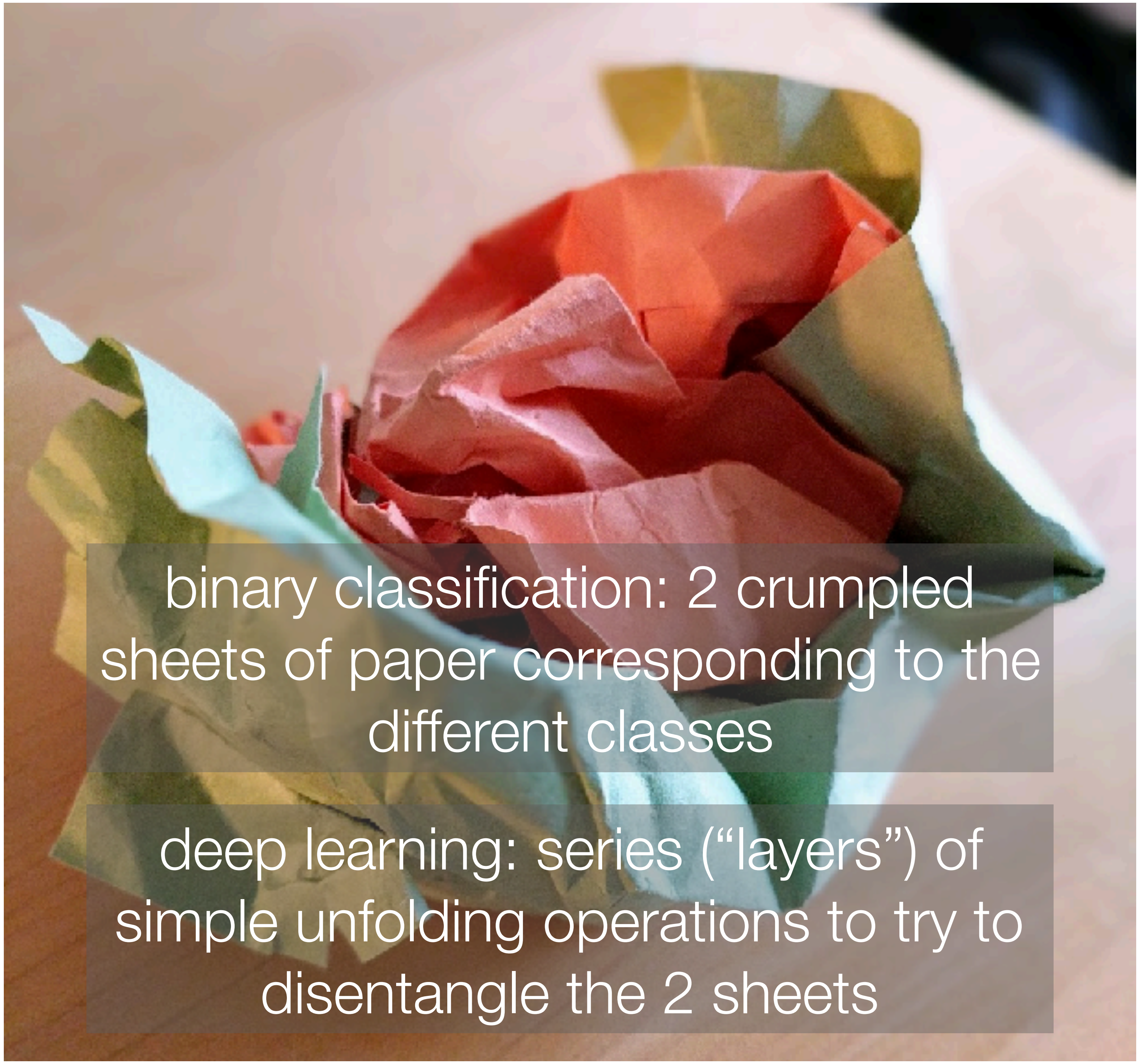
Carnegie Mellon University
Heinz College

# Neural Nets & Deep Learning

George Chen
(some neural net & deep learning slides are by Phillip Isola)

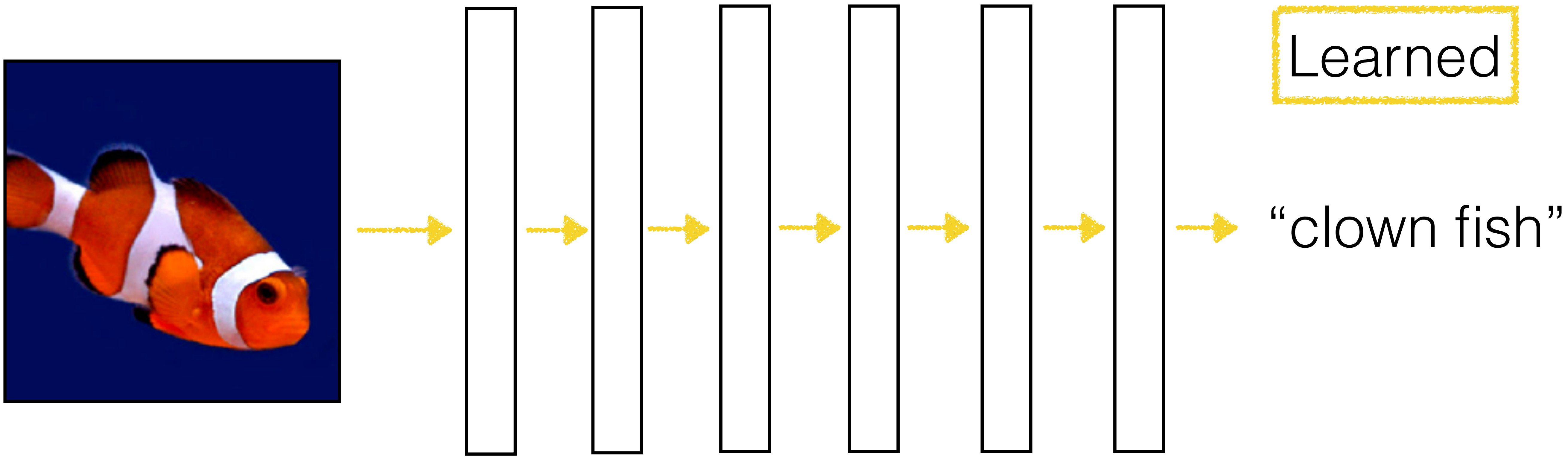
CMU 95-865 Spring 2018

# Crumpled Paper Analogy




Analogy: Francois Chollet, photo: George Chen

# Deep Learning




- Inspired by biological neural nets *but otherwise not the same at all* (biological neural nets do *not* work like deep nets)
- Learns a layered representation
    - Tries to get rid of manual feature engineering

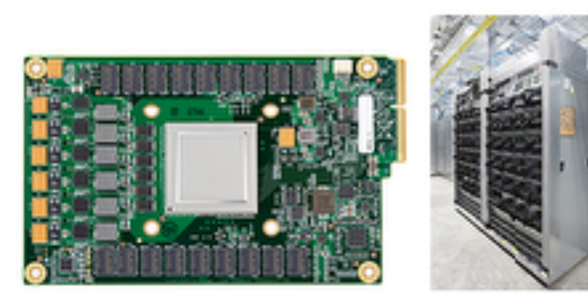
# Why Does Deep Learning Work?

Actually the ideas behind deep learning are old (~1980's)

- Big data
- Better hardware

CPU's
& Moore's law


GPU's



TPU's

- Better algorithms

# Structure Present in Data Matters

Neural nets aren't doing black magic

- **Image analysis:** convolutional neural networks (convnets) neatly <u>incorporates basic image processing structure</u>

- **Time series analysis:** recurrent neural networks (RNNs) <u>incorporates ability to remember and forget things over time</u>

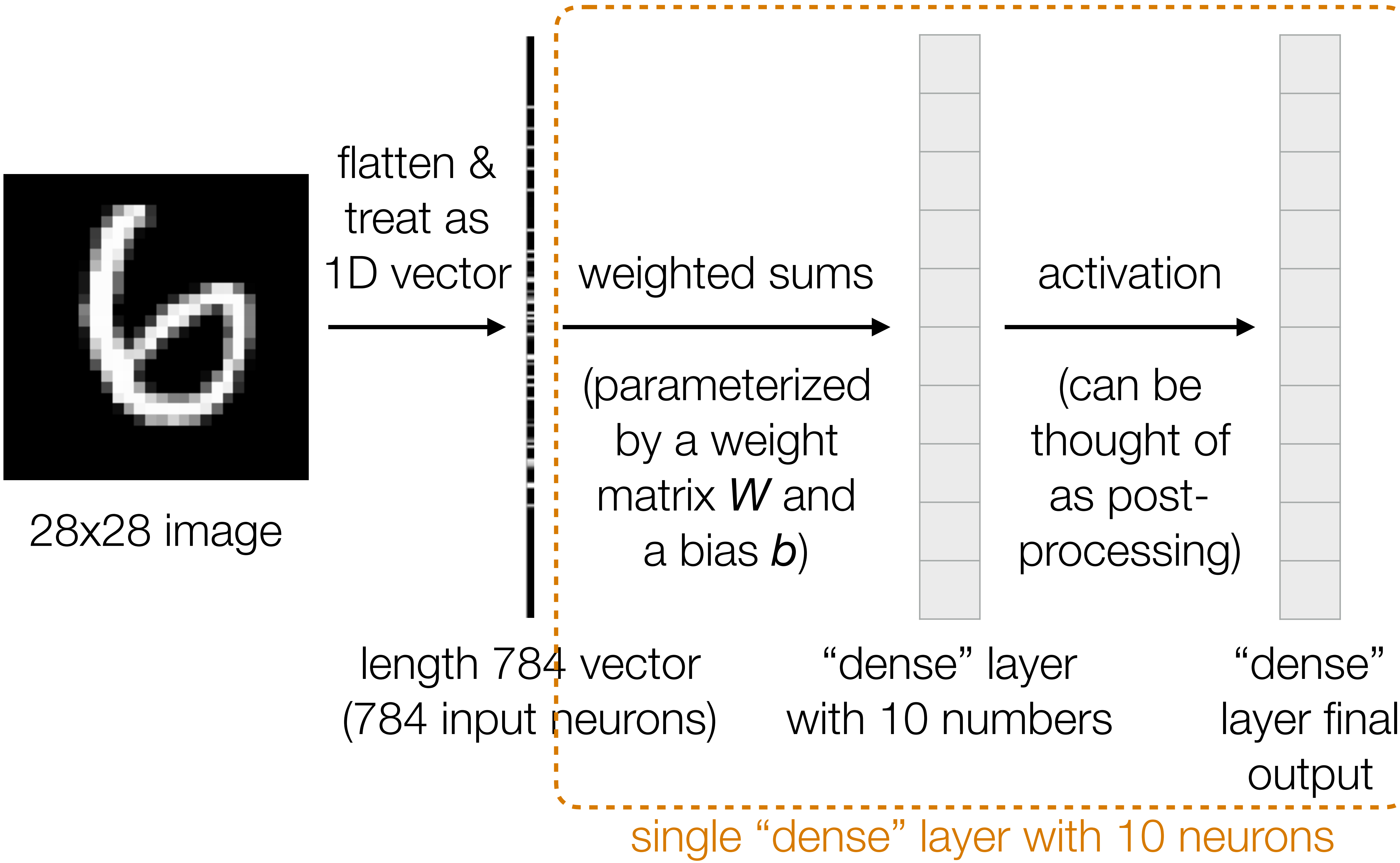
    - Note: text is a time series

    - Note: video is a time series

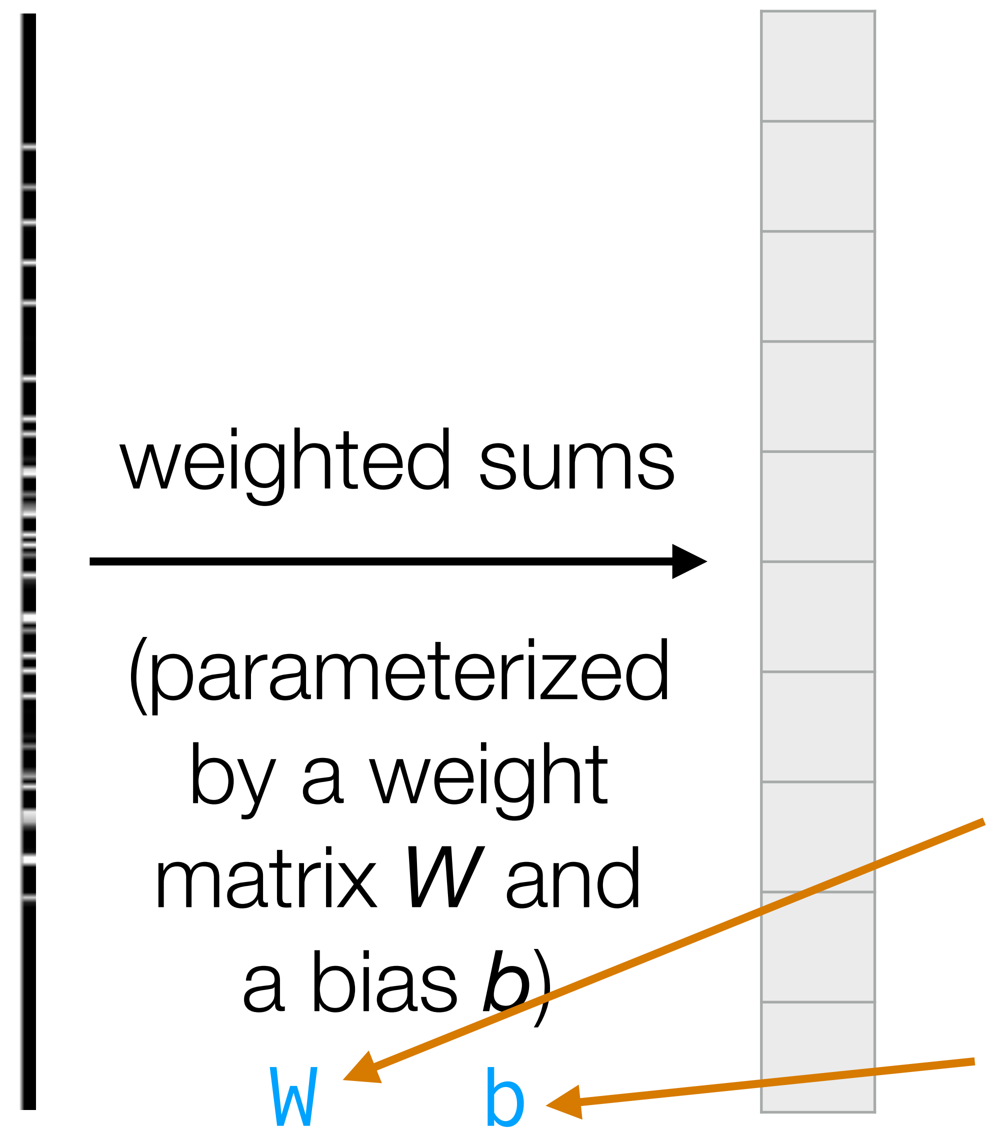
# Handwritten Digit Recognition Example

Walkthrough of building a 1-layer and then a 2-layer neural net

# Handwritten Digit Recognition



# Handwritten Digit Recognition



# Handwritten Digit Recognition

```
dense[0] = np.dot(input, W[:, 0]) + b[0]
dense[1] = np.dot(input, W[:, 1]) + b[1]
```

⋮

$$\texttt{dense[j]} = \sum_{i=0}^{783} \texttt{input[i]} \times \texttt{W[i, j]} + \texttt{b[j]}$$

weighted sums

(parameterized by a weight matrix *W* and a bias *b*)

W b

(2D numpy array of dimensions 784-by-10)

(1D numpy array with 10 entries)

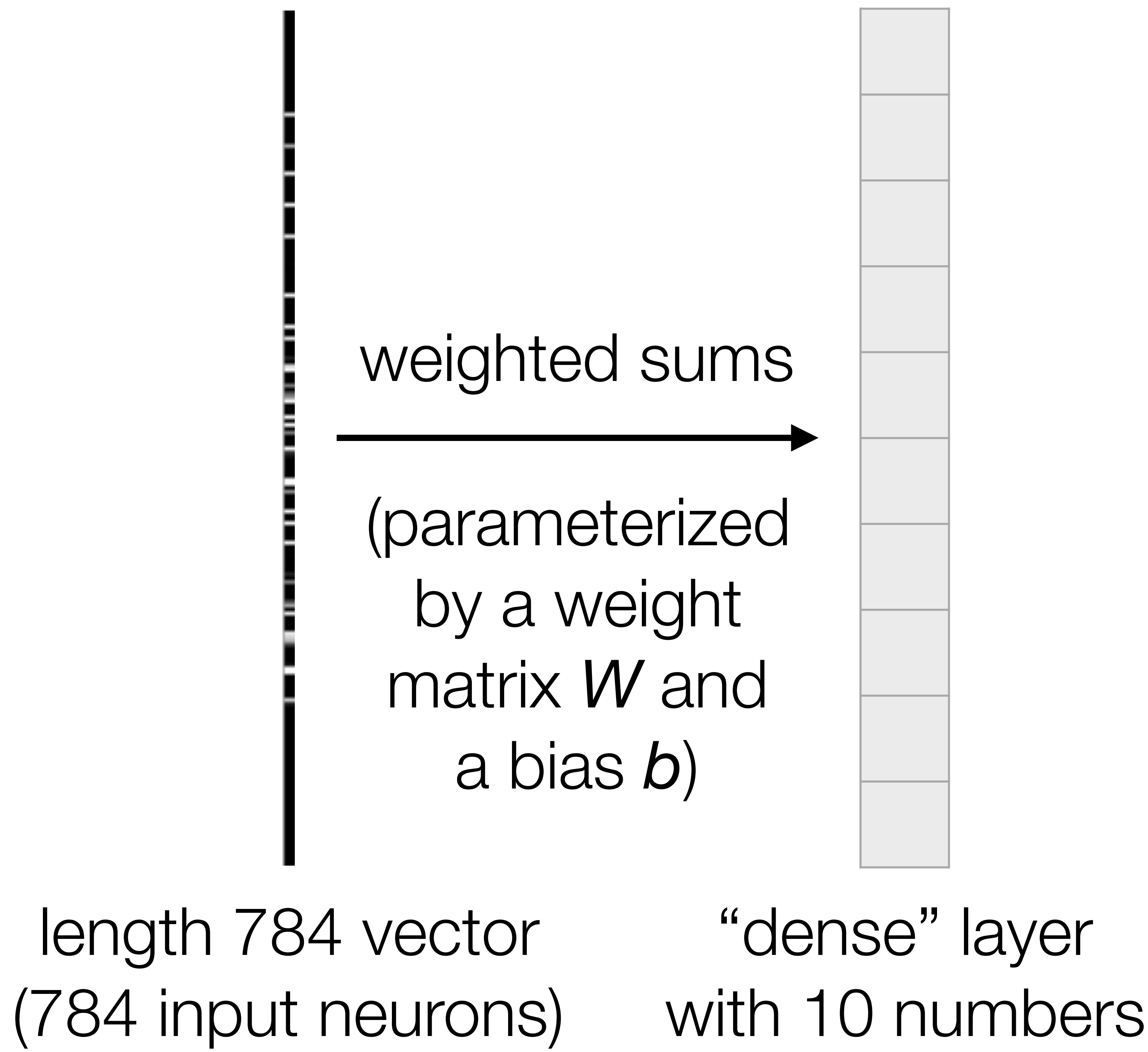
n 784 vector
nput neurons)

“dense” layer with 10 numbers

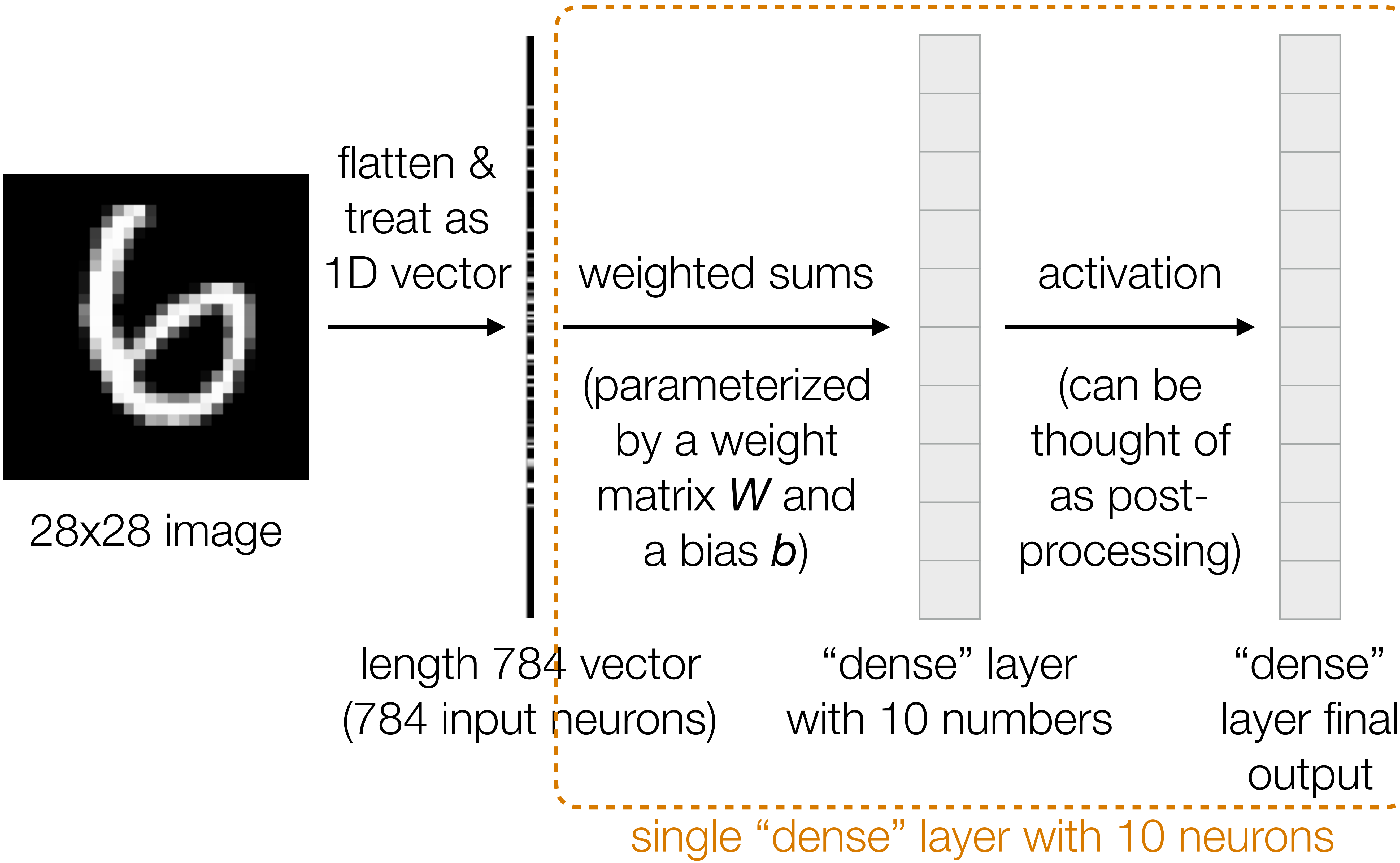
input
84 entries)

dense
(1D numpy array with 10 entries)

# Handwritten Digit Recognition



# Handwritten Digit Recognition



# Handwritten Digit Recognition

Many different activation functions possible

Example: **Rectified linear unit (ReLU)**
zeros out entries that are negative

```
dense_final = np.maximum(0, dense)
```

| “dense” layer with 10 numbers `dense` | ReLU → (can be thought of as post-processing) | “dense” layer final output `dense_final` |
|---|---|---|
| 4 | | 4 |
| 3.5 | | 3.5 |
| 4 | | 4 |
| -1 | | 0 |
| 0.5 | | 0.5 |
| 2 | | 2 |
| -4 | | 0 |
| 3 | | 3 |
| -2 | | 0 |
| 5 | | 5 |

# Handwritten Digit Recognition

Many different activation functions possible

Example: **softmax** turns the entries in the dense layer (prior to activation) into a probability distribution (using the “softmax” transformation)

```
dense_exp = np.exp(dense)
dense_exp /= np.sum(dense_exp)
dense_final = dense_exp
```

| | |
|---|---|
| 4 | 0.17 |
| 3.5 | 0.10 |
| 4 | 0.17 |
| -1 | 0.00 |
| 0.5 | 0.01 |
| 2 | 0.02 |
| -4 | 0.00 |
| 3 | 0.06 |
| -2 | 0.00 |
| 5 | 0.46 |

**softmax**

(can be thought of as post-processing)

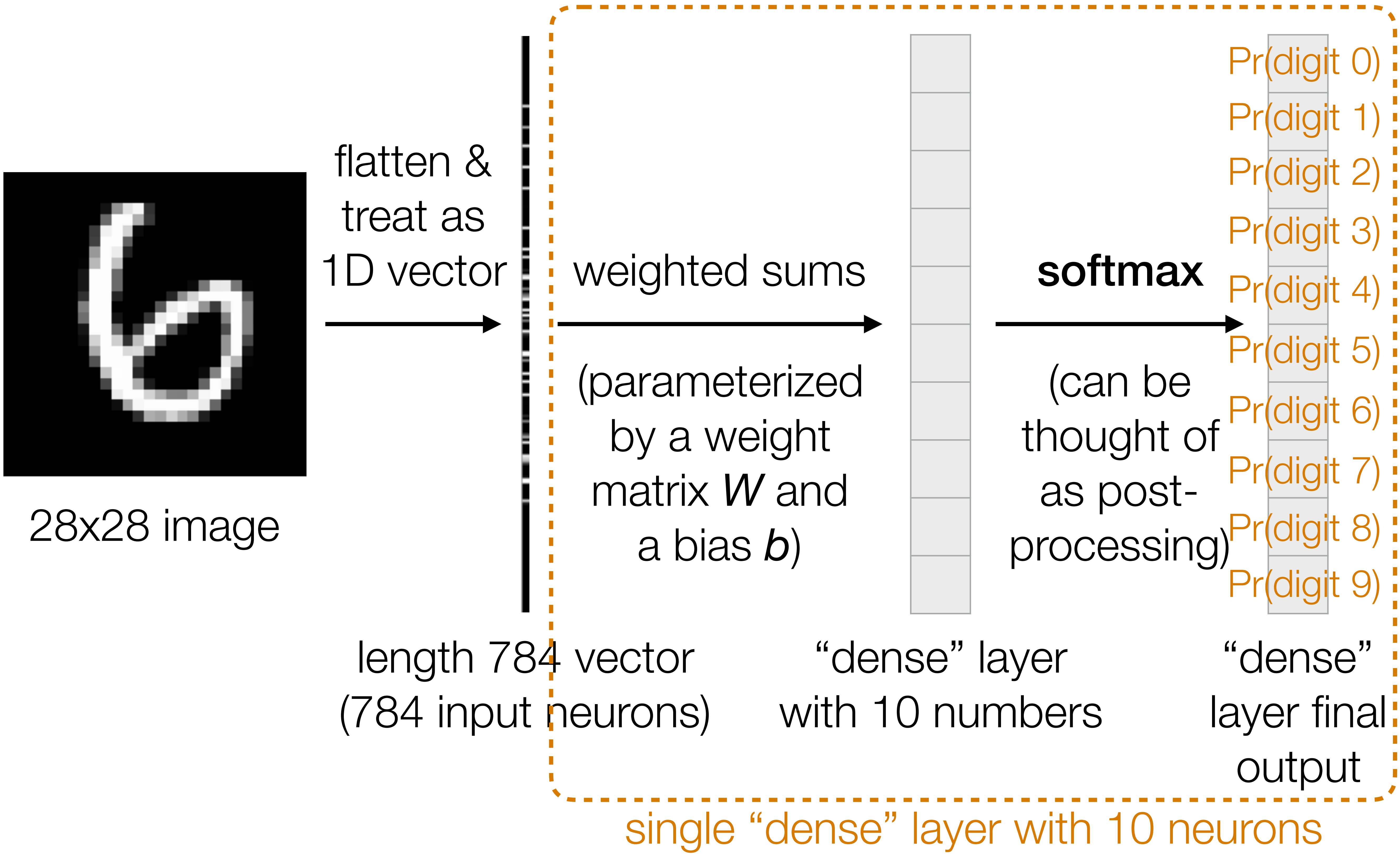
“dense” layer with 10 numbers

`dense`

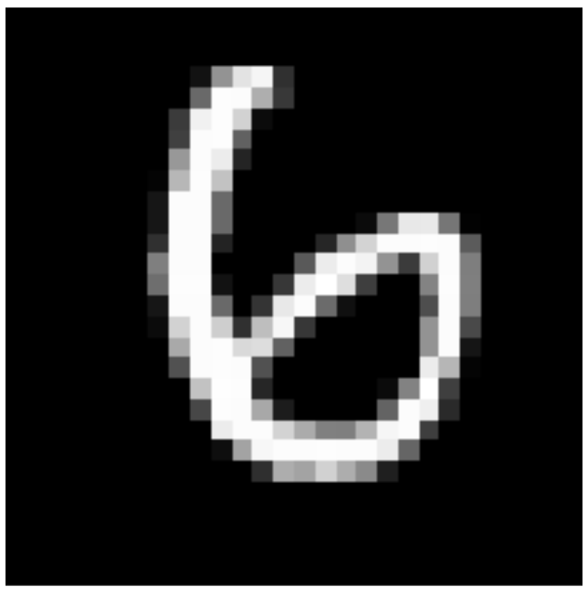
“dense” layer final output

`dense_final`

# Handwritten Digit Recognition



# Handwritten Digit Recognition



28x28 image

flatten & treat as 1D vector

length 784 vector (784 input neurons)

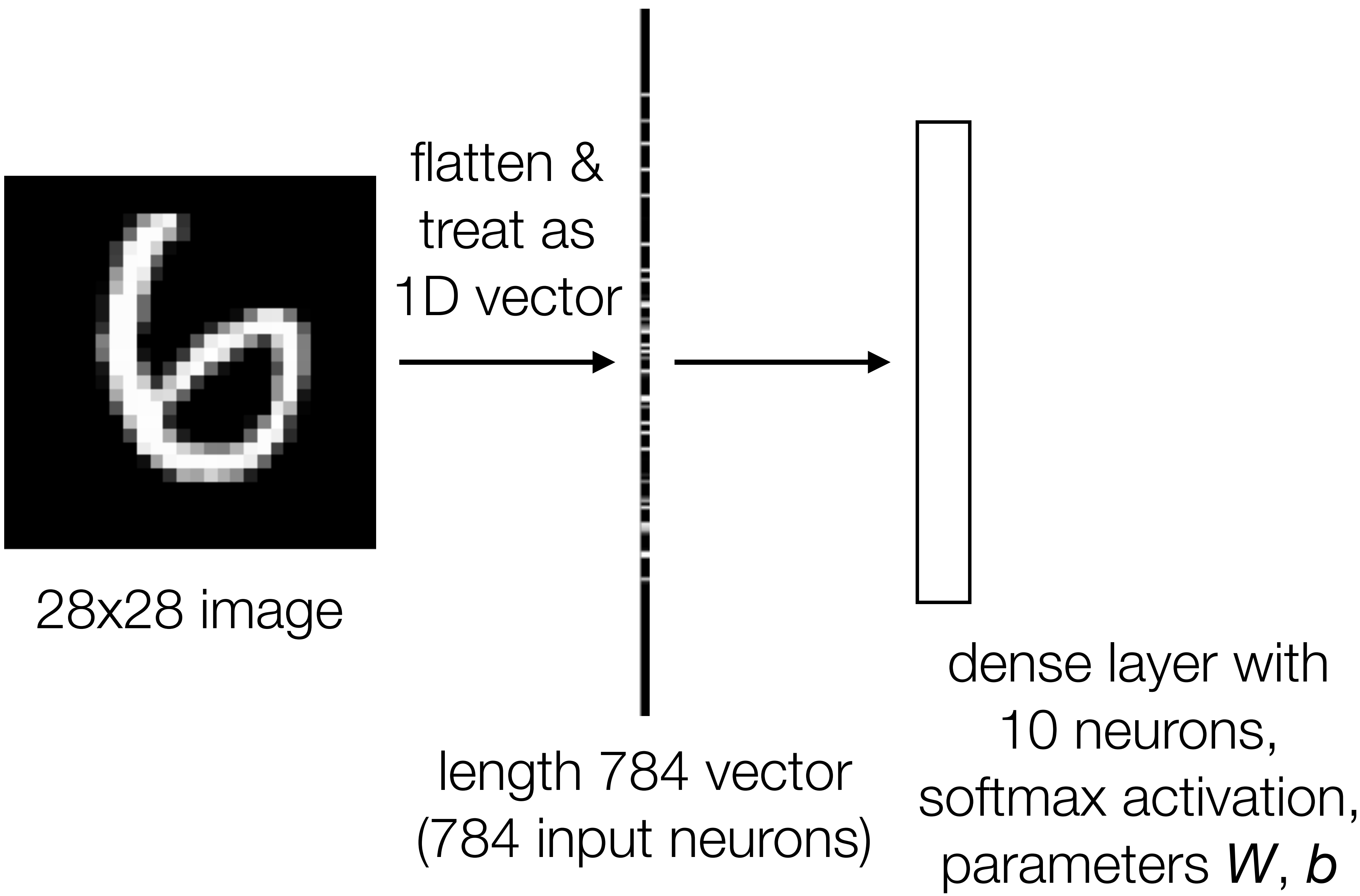
dense layer with 10 neurons, softmax activation, parameters *W*, *b*

We want the output of the dense layer to encode probabilities for whether the input image is a 0, 1, 2, …, 9 *but as of now we aren't providing any sort of information to enforce this*

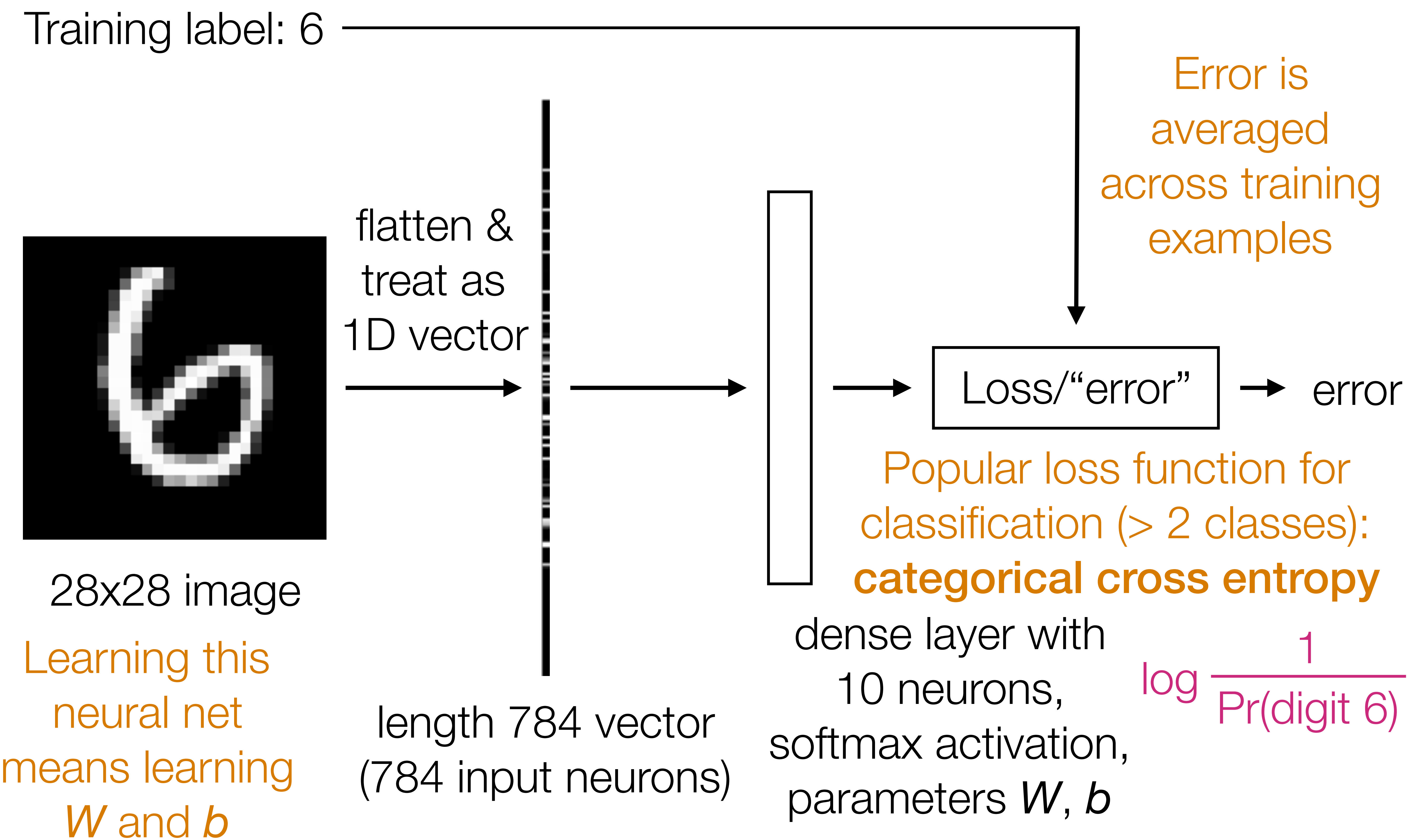
# Handwritten Digit Recognition

Demo part 1

# Handwritten Digit Recognition



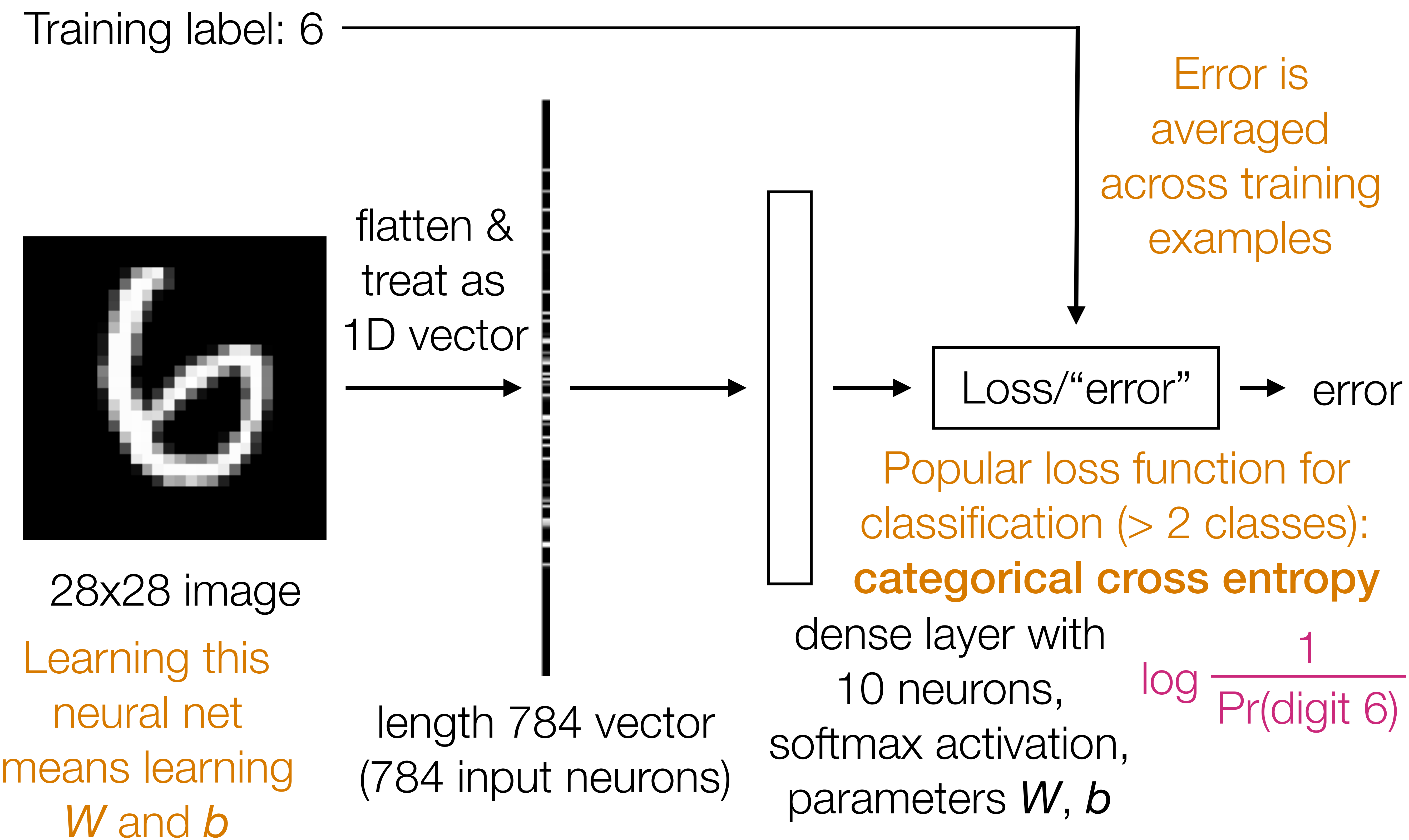
# Handwritten Digit Recognition



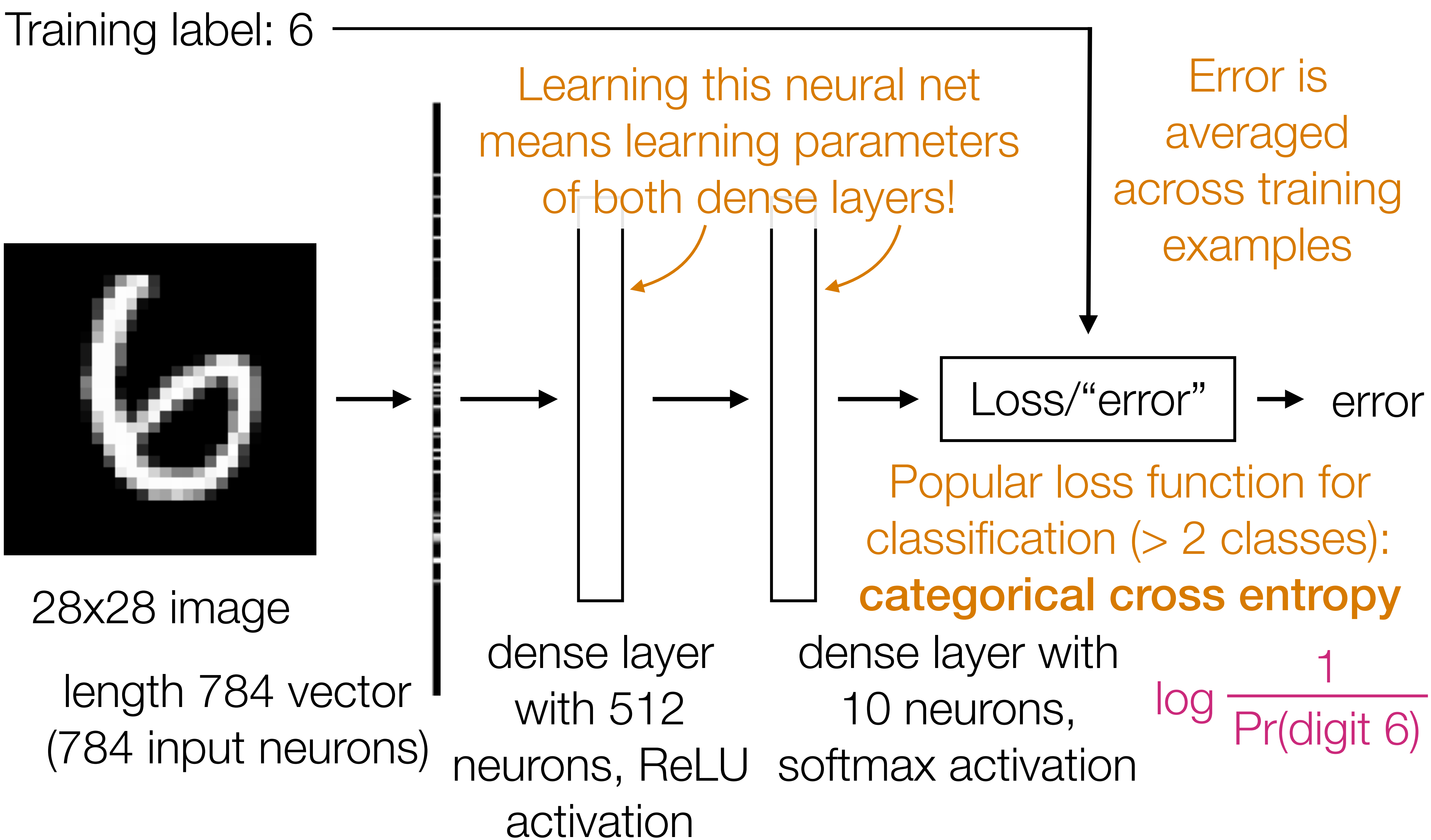
# Handwritten Digit Recognition

Demo part 2

# Handwritten Digit Recognition



# Handwritten Digit Recognition

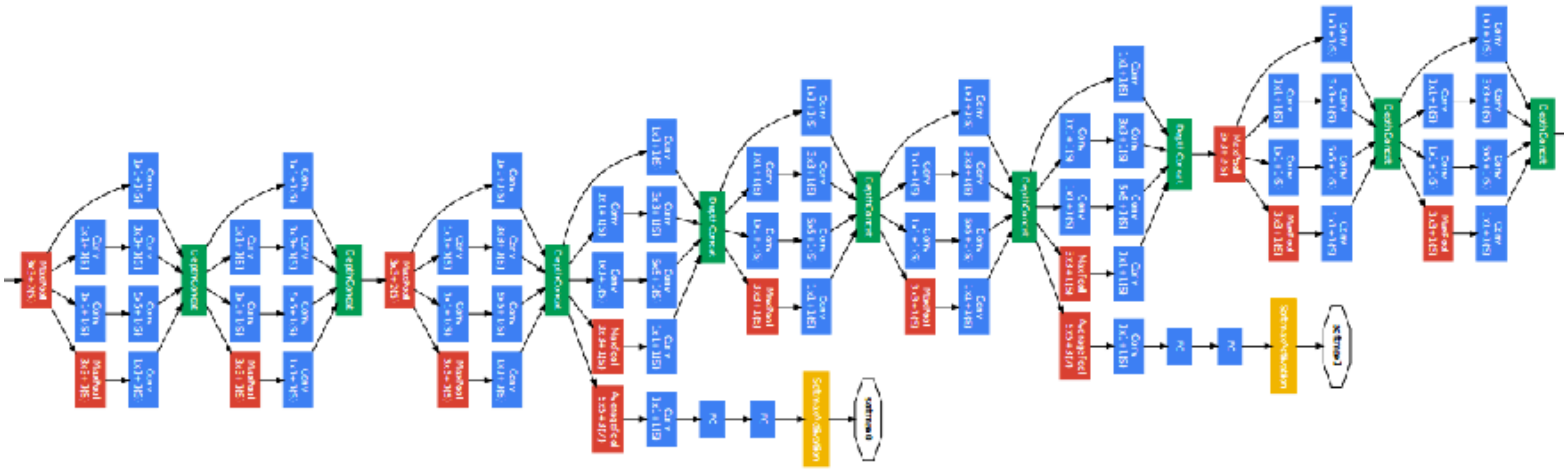


# Handwritten Digit Recognition

Demo part 3

# Architecting Neural Nets

- Increasing number of layers (depth) makes neural net more complex
  - Can approximate more functions
  - More parameters needed
    - More training data may be needed
- Designing neural net architectures is a bit of an art
  - How to select the number of neurons for intermediate layers?
  - Very common in practice: modify existing architectures that are known to work well (e.g., VGG-16 for computer vision/image processing)



GoogLeNet 2014