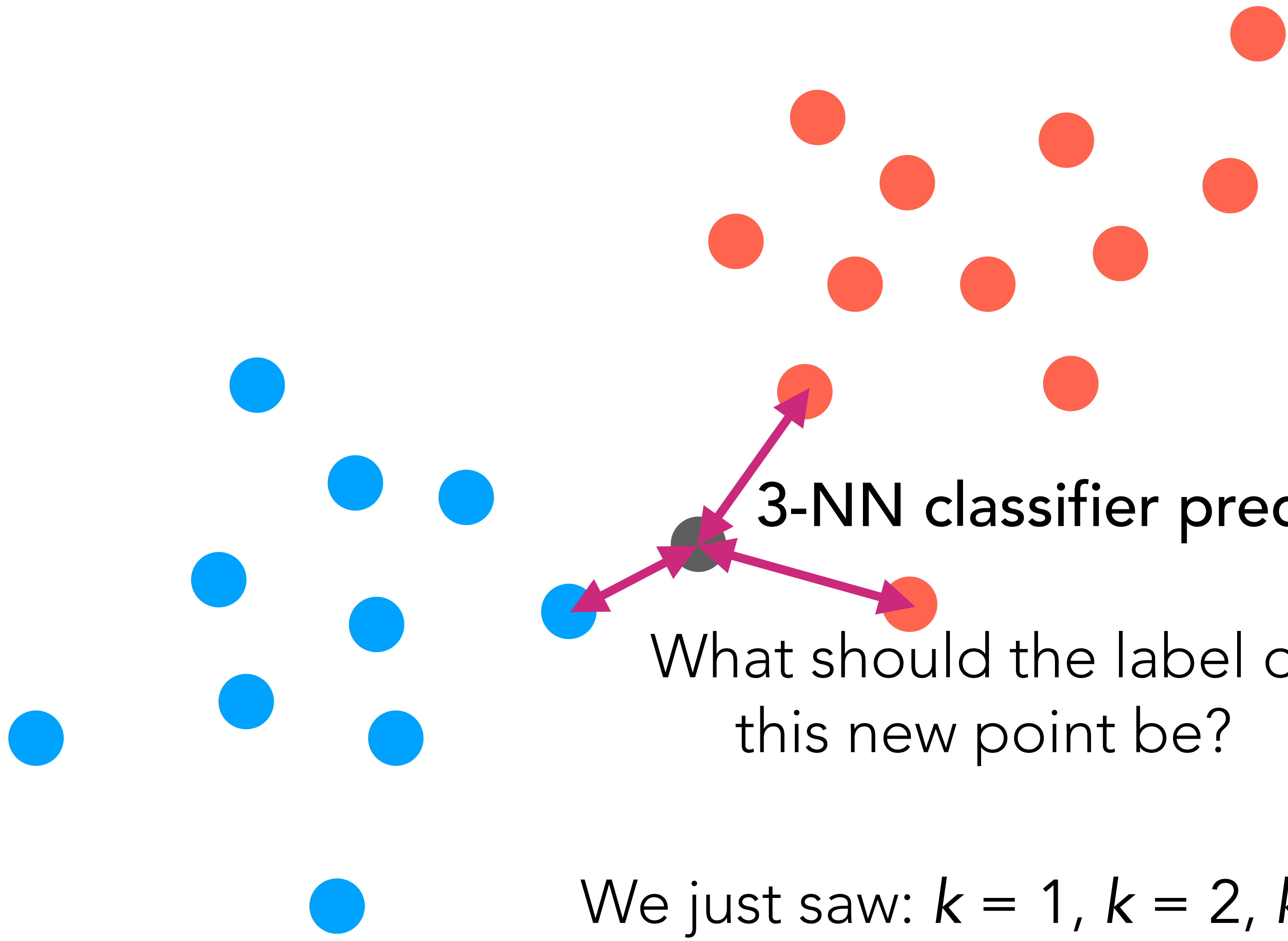
Carnegie Mellon University
Heinz College

95-865 Unstructured Data Analytics

Lecture 11: Wrap up basic prediction concepts; intro to neural nets & deep learning

Nearly all slides by George H. Chen
with a few by Phillip Isola

(Flashback) Example: *k*-NN Classification



We just saw: $k = 1$, $k = 2$, $k = 3$

What happens if $k = n$?

(Flashback)

How do we choose *k*?

What I'll describe next can be used to select hyperparameter(s) for any prediction method

Fundamental question:
How do we assess how good a prediction method is?

(Flashback) Hyperparameters vs. Parameters

- We fit a model's parameters to training data (terminology: we "learn" the parameters)

- We pick values of hyperparameters and they do *not* automatically get fit to training data

- Example: Gaussian mixture model
  - Hyperparameter: number of clusters *k*
  - Parameters: cluster probabilities, means, covariance matrices

- Example: *k*-NN classification
  - Hyperparameter: number of nearest neighbors *k*
  - Parameters: N/A

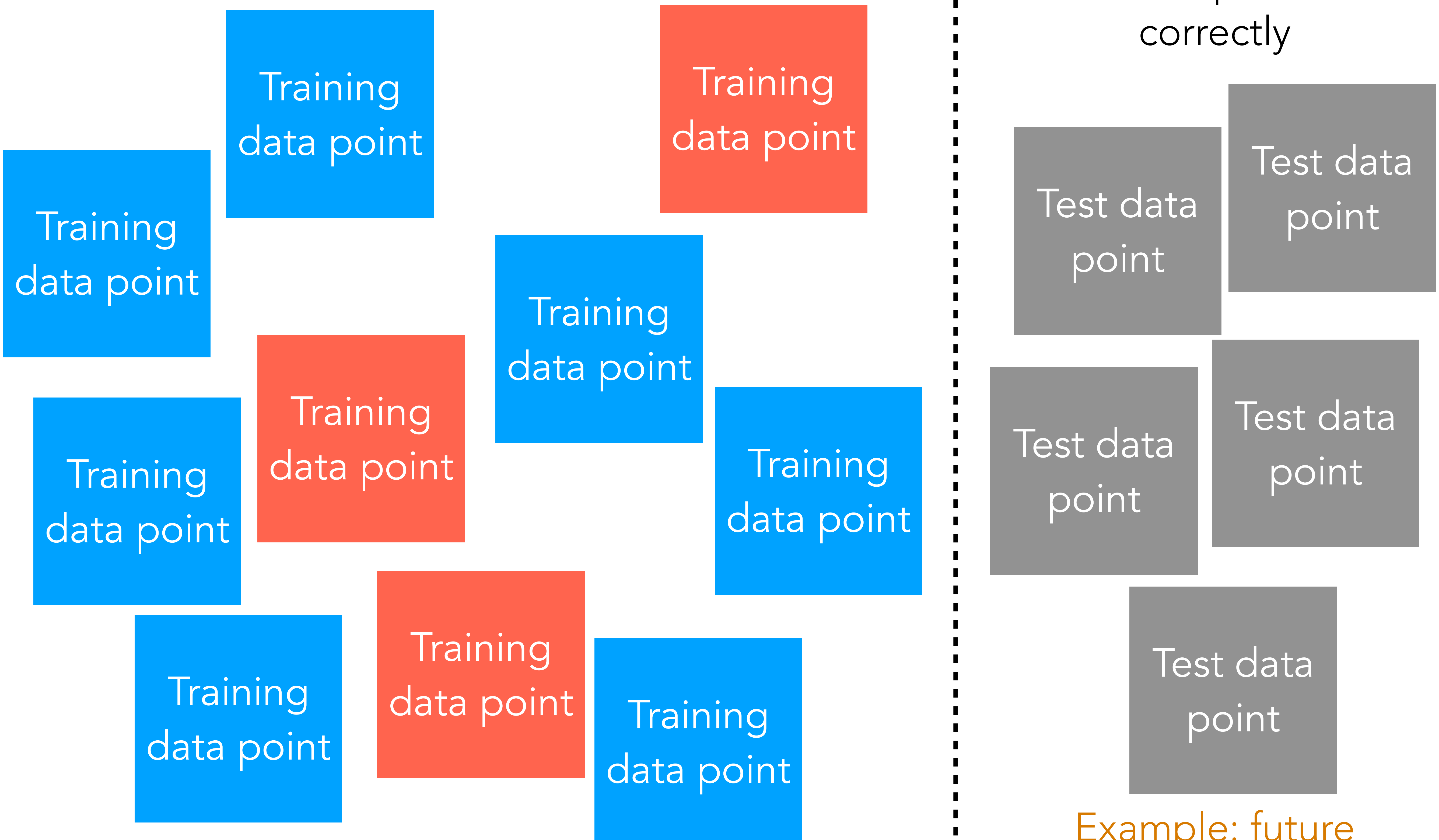
Actually, there's another hyperparameter: distance function to use (for simplicity, we assume Euclidean distance for now)

⚠️ Major assumption:
training and test data "look alike"
(technically: training and test data are i.i.d. sampled from the same underlying distribution)

Prediction is harder when training and test data appear quite different!

Training data

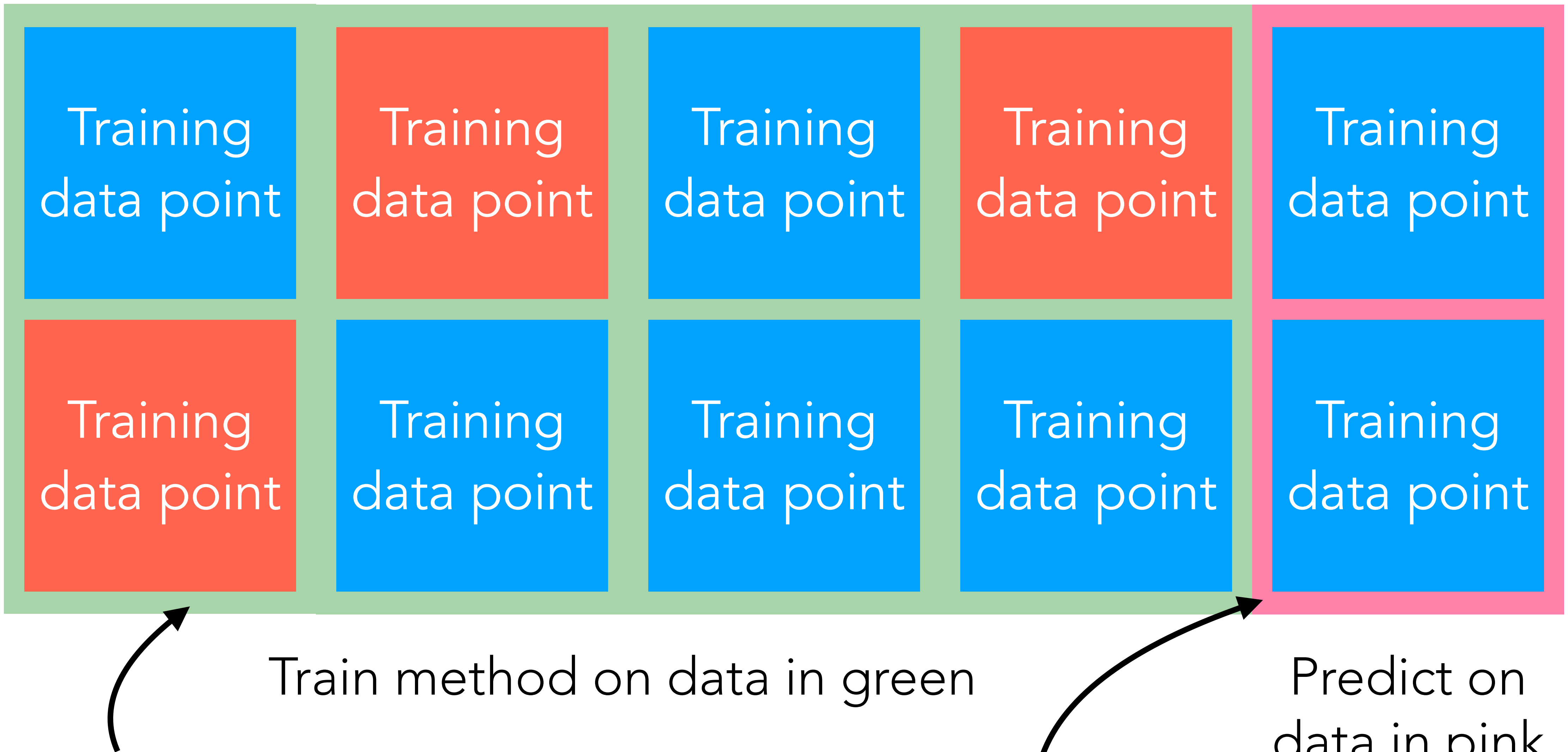
Want to classify these points correctly



Example: Each data point is an email and we know whether it is spam/ham

Example: future emails to classify as spam/ham

(shuffling makes sense since we assume data are i.i.d.)



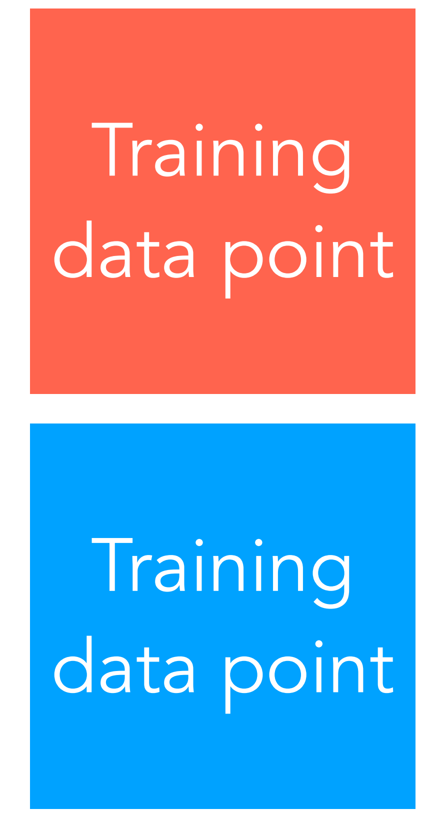
Train method on data in green

Predict on data in pink

Terminology for this class:
“**Proper training data**”
(the green box)

“**Validation data**”
(the pink box)

Predicted labels on validation data

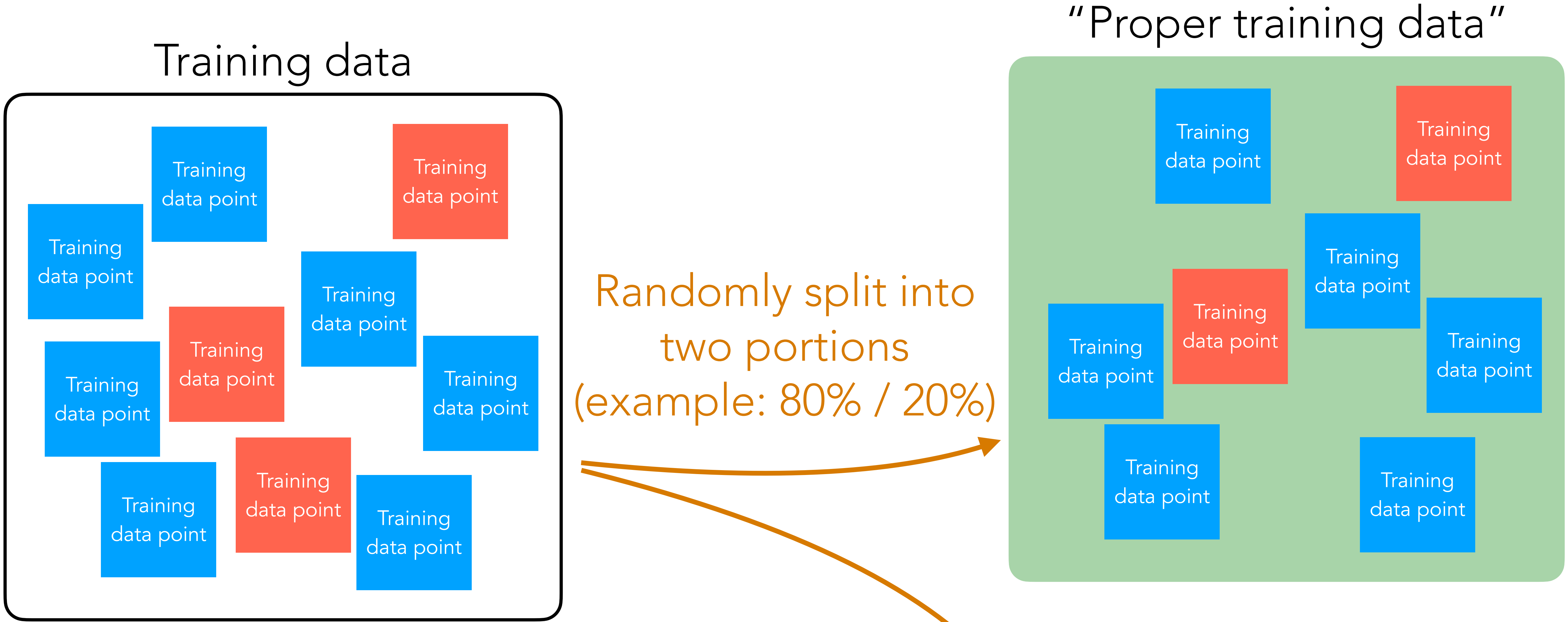


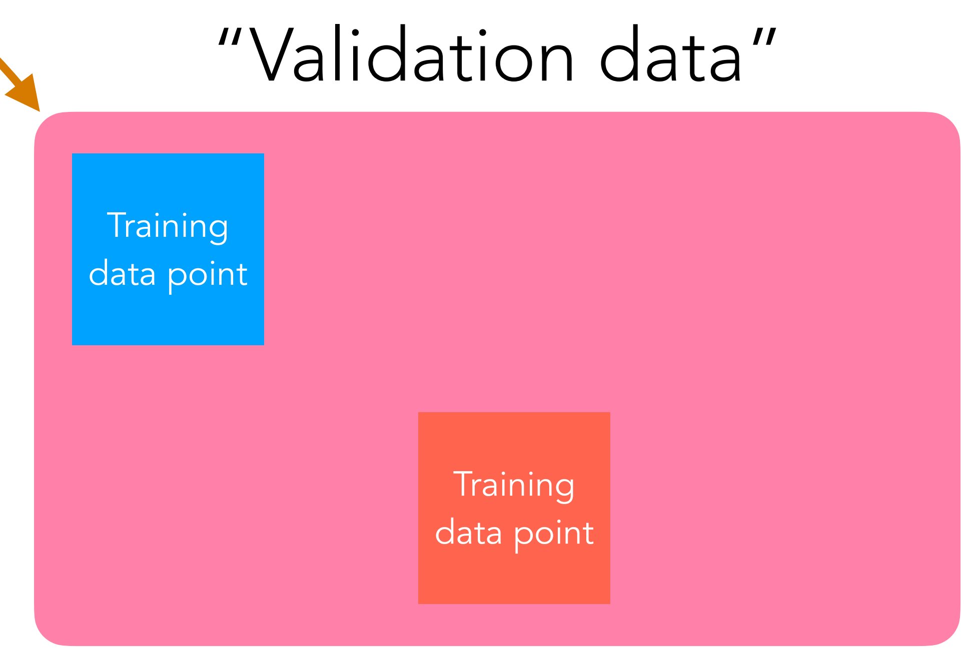
Compute prediction accuracy

50%

This is called data splitting/“train-validation split”
In this example: we did a 80%-20% split

Some people, including sklearn, call this “train-test split” but in this class, we will use “test data” to refer to true test data that the training procedure does not see





For *k* = 1, 2, …, some user-specified max:

1. Train *k*-NN classifier on proper training data
2. Use a score function to evaluate how well the trained model predicts on validation data

Use whichever value of *k* achieves the best score

There are many score functions possible

Example: fraction of validation points correctly predicted (raw accuracy)

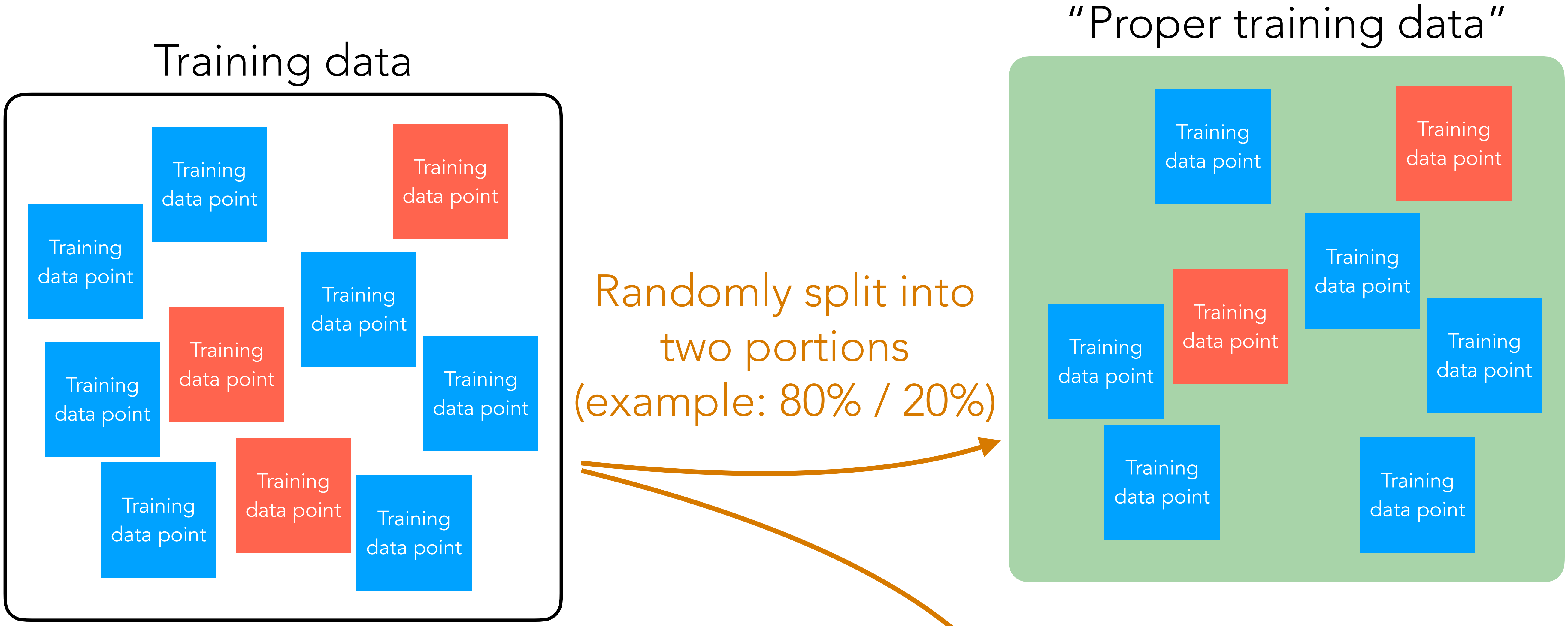
Terminology Remarks

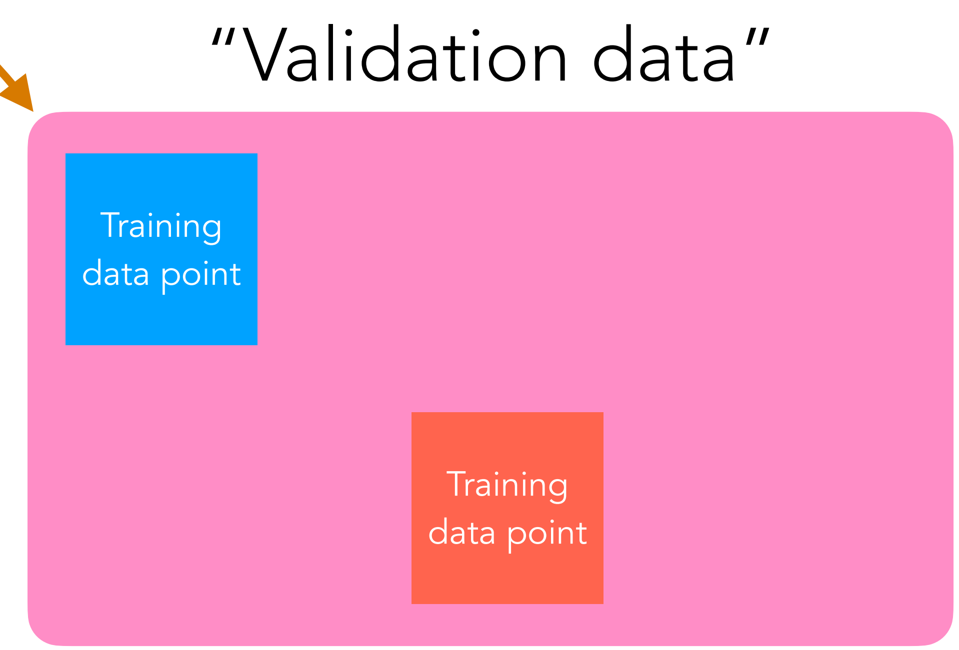
- What we're using is commonly called a **train/validation split**
  - If you also consider that there's a **test set** that's not part of train/validation data: division is called **train/validation/test split**

- **Warning:** in the machine learning community, what I'm calling the "proper training data"/"proper training set" is commonly also called the "training data"/"training set" even though it is typically a *subset* of the full training data (that we split into proper training/validation sets)
  - Put another way: what precisely the "training data" refers to can be ambiguous as it could mean the full training data or the full training data minus the validation data
  - In 95-865, to avoid confusion, we use the somewhat non-standard terminology "proper training set"/"proper training data" to refer to the the full training data minus the validation data

Hyperparameter Tuning for *k*-NN Classifier

Demo





For k = 1, 2, …, some user-specified max:

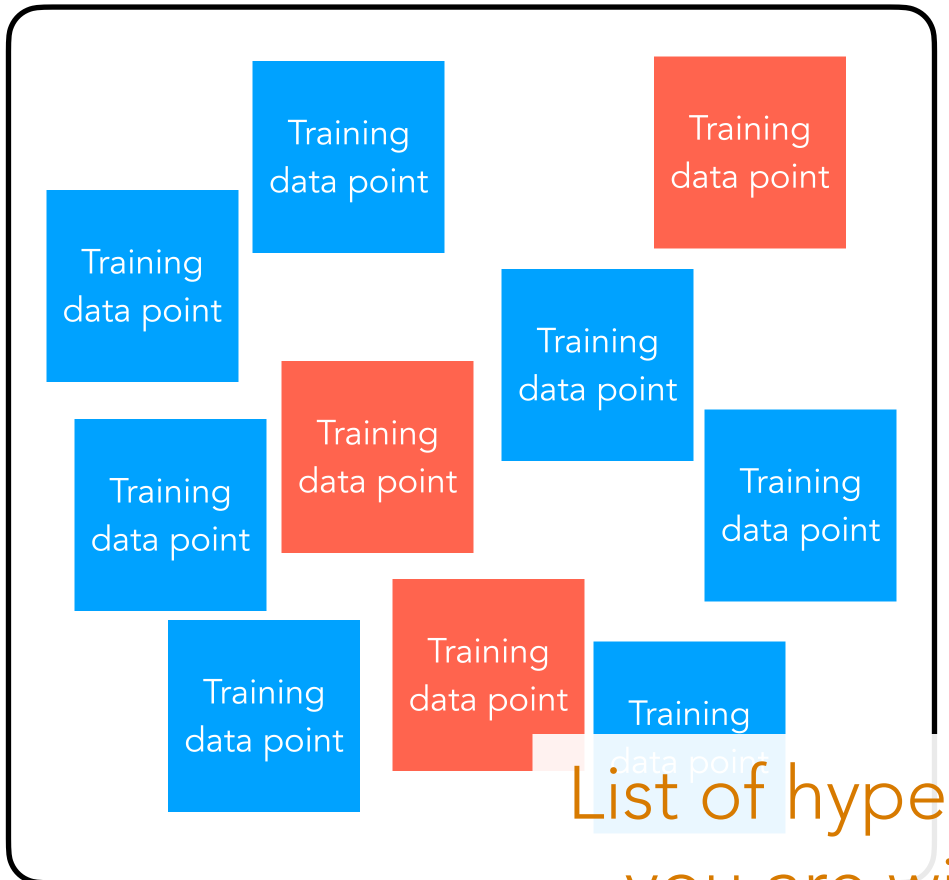
1. Train k-NN classifier on proper training data
2. Use a score function to evaluate how well the trained model predicts on validation data

Use whichever value of k achieves the best score

There are many score functions possible

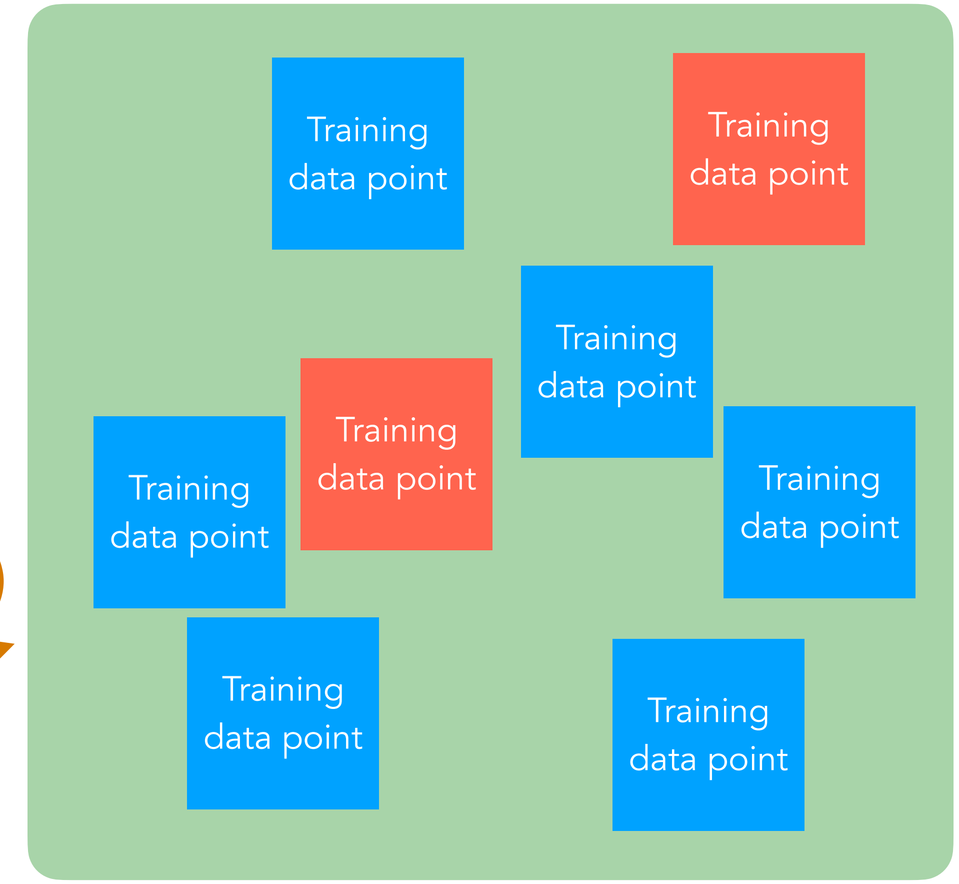
Example: fraction of validation points correctly predicted (raw accuracy)

Training data



Randomly split into two portions (example: 80% / 20%)

"Proper training data"



"Validation data"

List of hyperparameters you are willing to try

For *k*, *ρ* = (1, "Euclidean"), (1, "Cosine"), …:

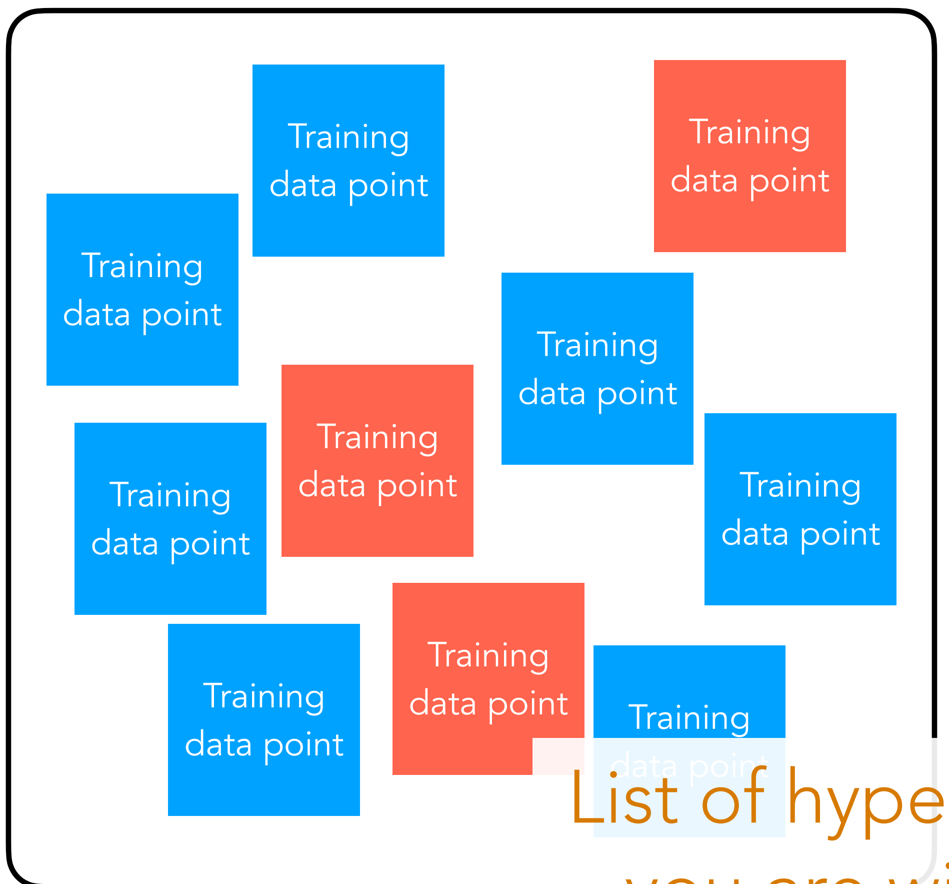
1. Train *k*-NN classifier on proper training data with distance dist *ρ*
2. Use a score function to evaluate how well the trained model predicts on validation data

Use whichever value of *k* achieves the best score

There are many score functions possible

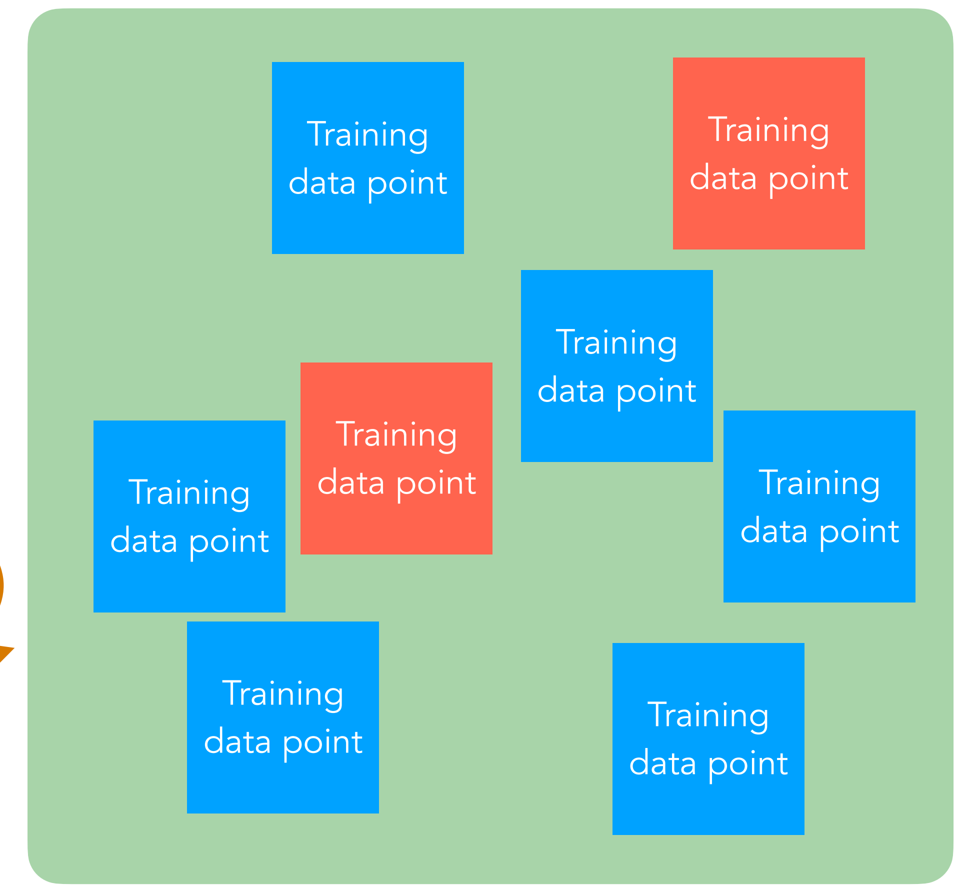
Example: fraction of validation points correctly predicted (raw accuracy)

Training data

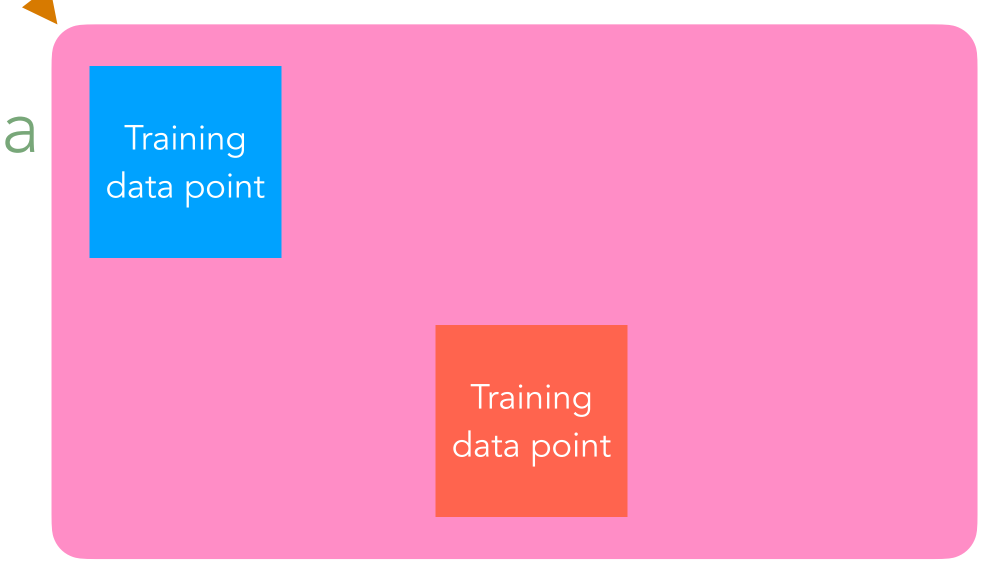


Randomly split into two portions (example: 80% / 20%)

“Proper training data”



“Validation data”



List of hyperparameters you are willing to try

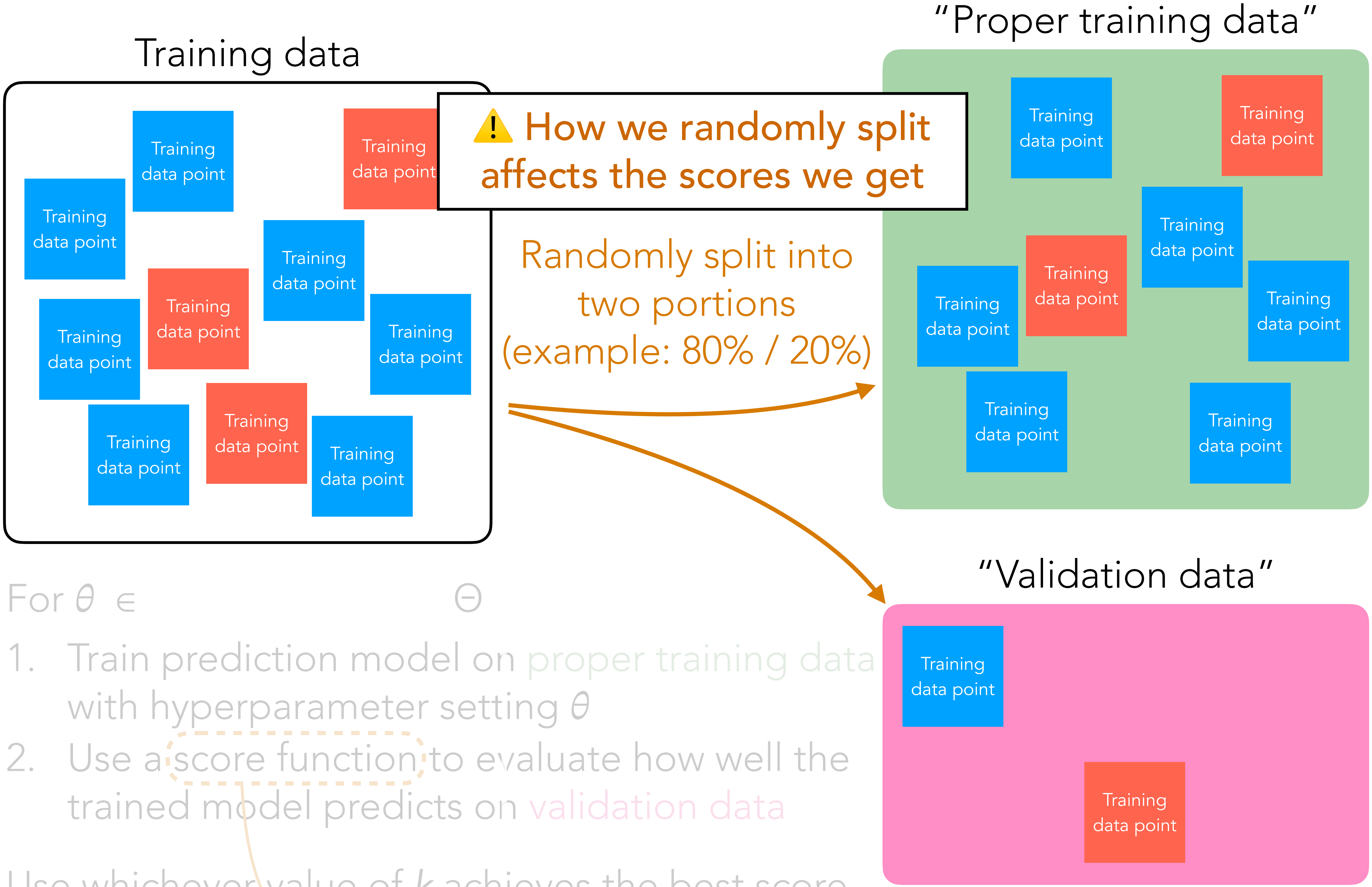
For $\theta \in \Theta$

1. Train prediction model on proper training data with hyperparameter setting θ
2. Use a score function to evaluate how well the trained model predicts on validation data

Use whichever value of k achieves the best score

There are many score functions possible

Example: fraction of validation points correctly predicted (raw accuracy)



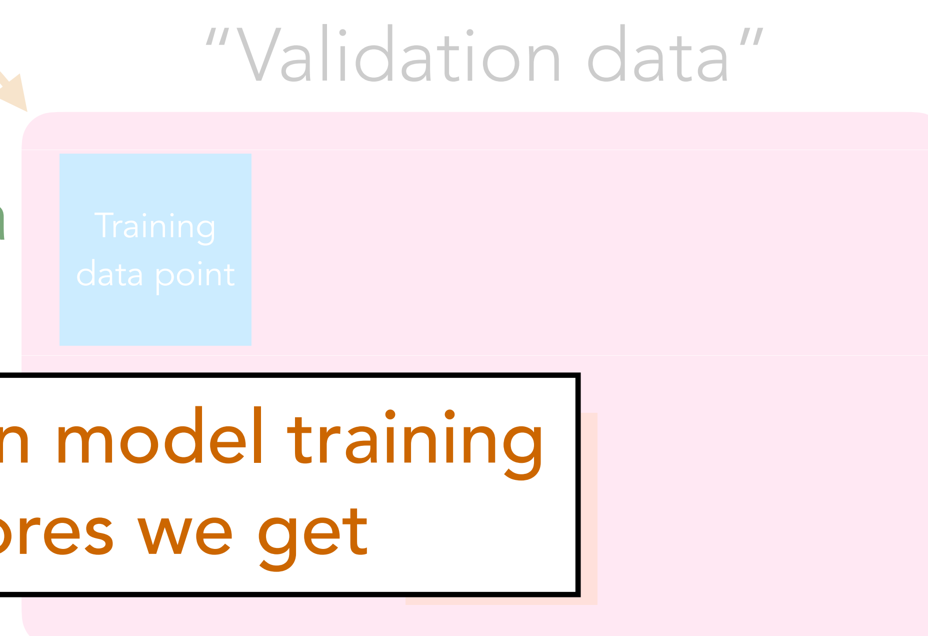
For $\theta \in \Theta$

1. Train prediction model on proper training data with hyperparameter setting θ
2. Use a score function to evaluate how well the trained model predicts on validation data

Use whichever value of k achieves the best score

There are many score functions possible

Example: fraction of validation points correctly predicted (raw accuracy)



For $\theta \in \Theta$

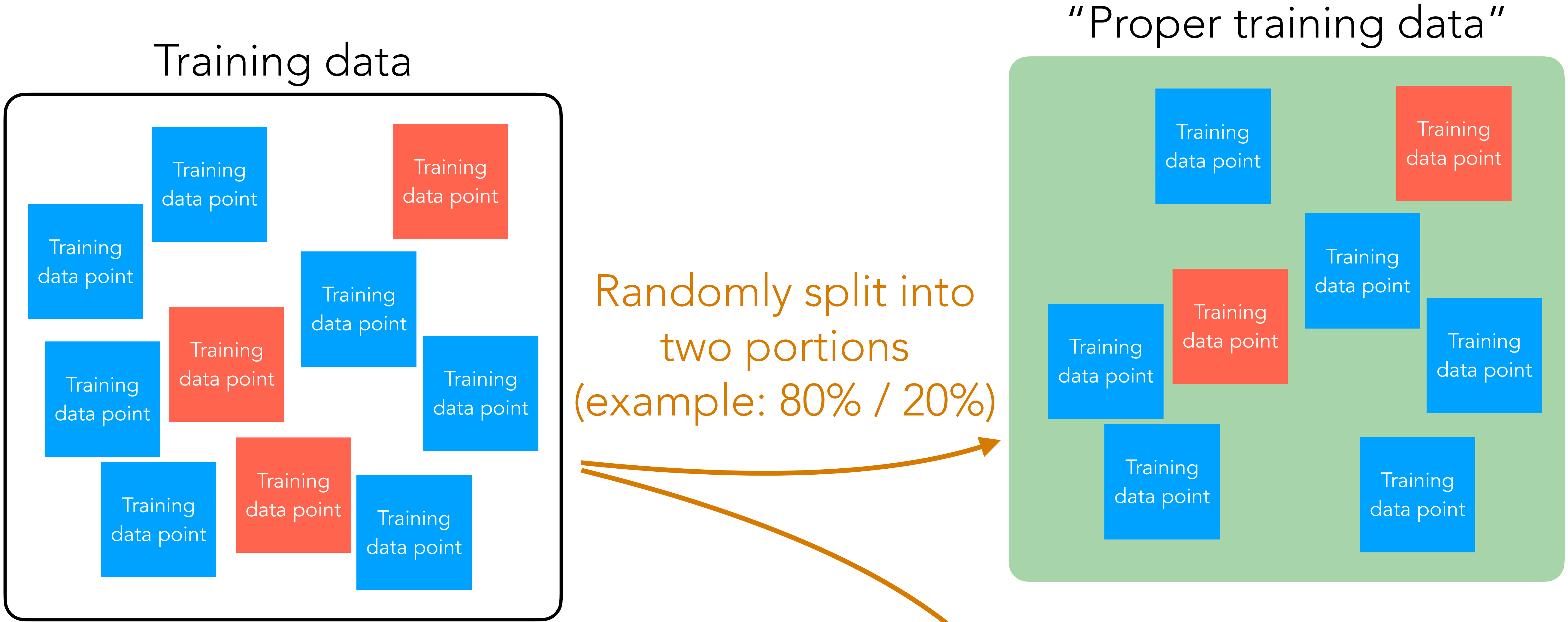
1. Train prediction model on proper training data with hyperparameter setting θ
2. Use a score function to eval[uate how well the] trained model predicts on v[alidation data]

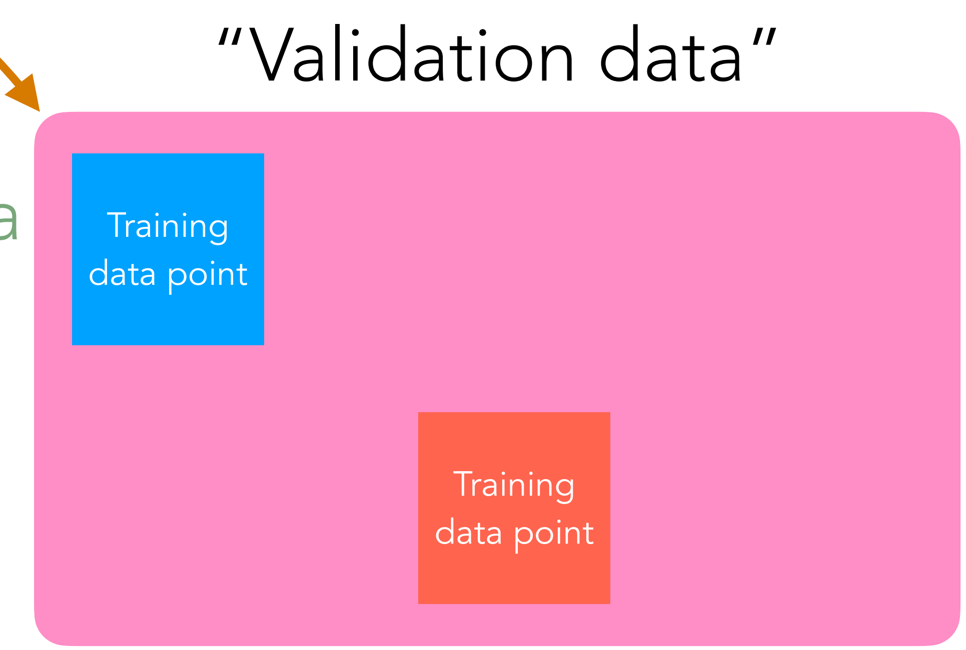
Use whichever value of k achieves the best score

There are many score functions possible

Example: fraction of validation points correctly predicted (raw accuracy)

⚠️ **Randomness in model training affects the scores we get**





For $\theta \in \Theta$

1. Train prediction model on proper training data with hyperparameter setting θ
2. Use a score function to evaluate how well the trained model predicts on validation data

Use whichever value of *k* achieves the best score

There are many score functions possible

Example: fraction of validation points correctly predicted (raw accuracy)

The rest of the prediction models we consider in UDA will be based on *neural nets* (which commonly have hyperparameters!)

Neural net models can be tuned in the same manner we just saw for k-NN classification

Important: some of you may have seen *cross-validation* before

- If you don't know what this is, don't worry about it
- Cross-validation is commonly too expensive for neural net training so we stick to the train/val split strategy

Neural Nets & Deep Learning

Extremely useful in practice:

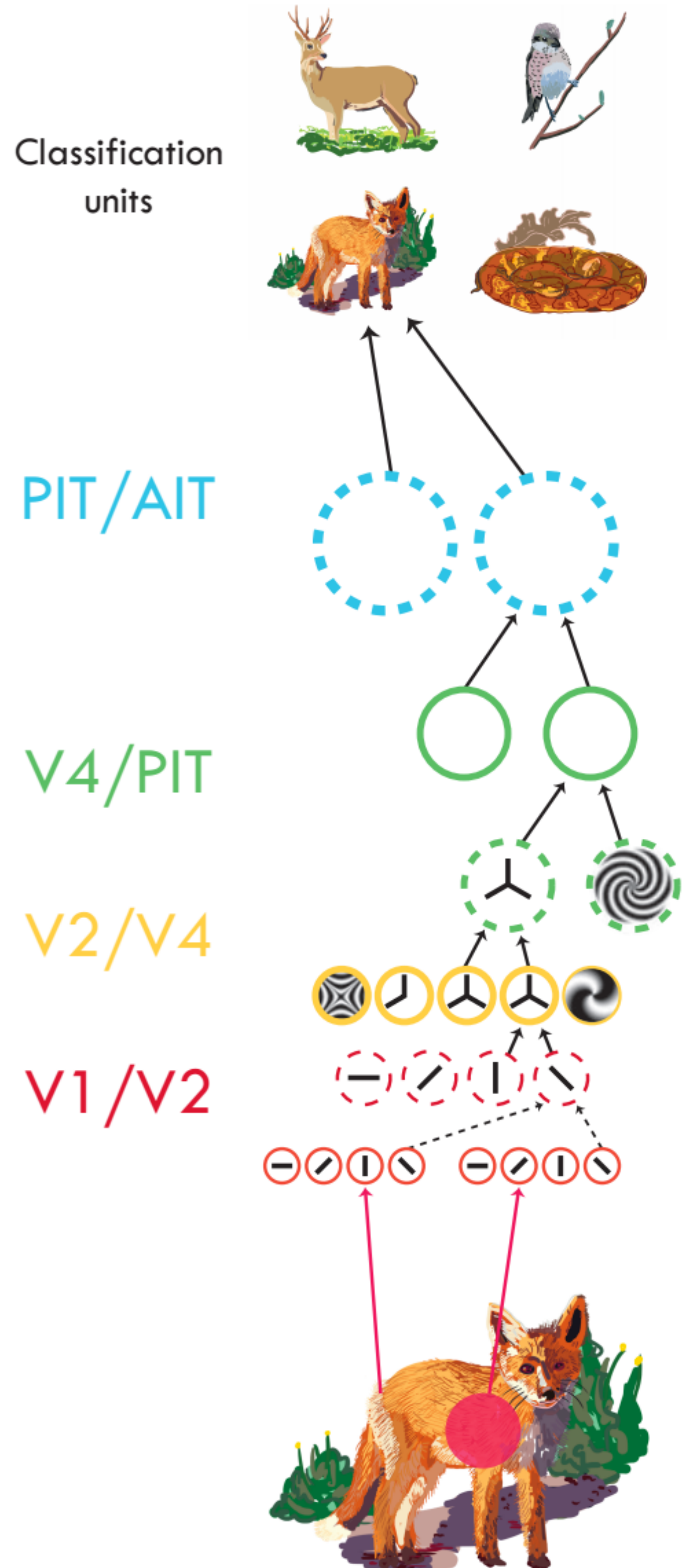
- Human-level image classification
- Human-level speech recognition
- Human-level in machine translation, text-to-speech
- Self-driving cars
- Better than humans at playing Go and many other games
- Capable of generating fake images, video, and audio that look real
- Human-level chatbots (ChatGPT, GPT4.0, Gemini, Claude, …)

⚠️ We don't fully understand when many of these technologies fail or how best to prevent their misuse

⚠️ All of this technology will get better over time

🤔 Sometimes, I think the question isn't whether you ask chatbots for help, the question is whether in a few years they'll bother asking you for help

What is deep learning?



Slide by Phillip Isola

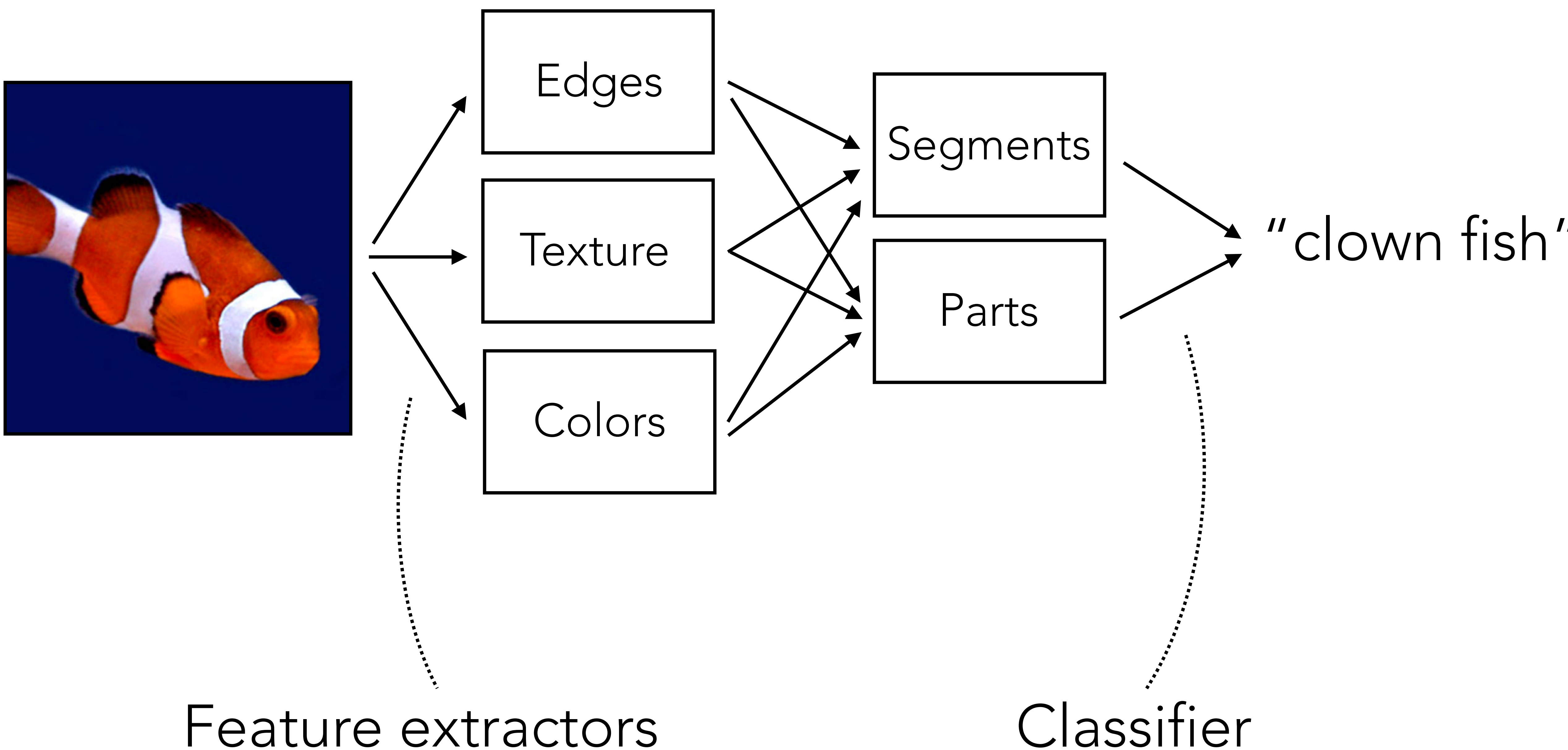
Serre, 2014

Basic Idea

 → Brain/Machine → “clown fish”

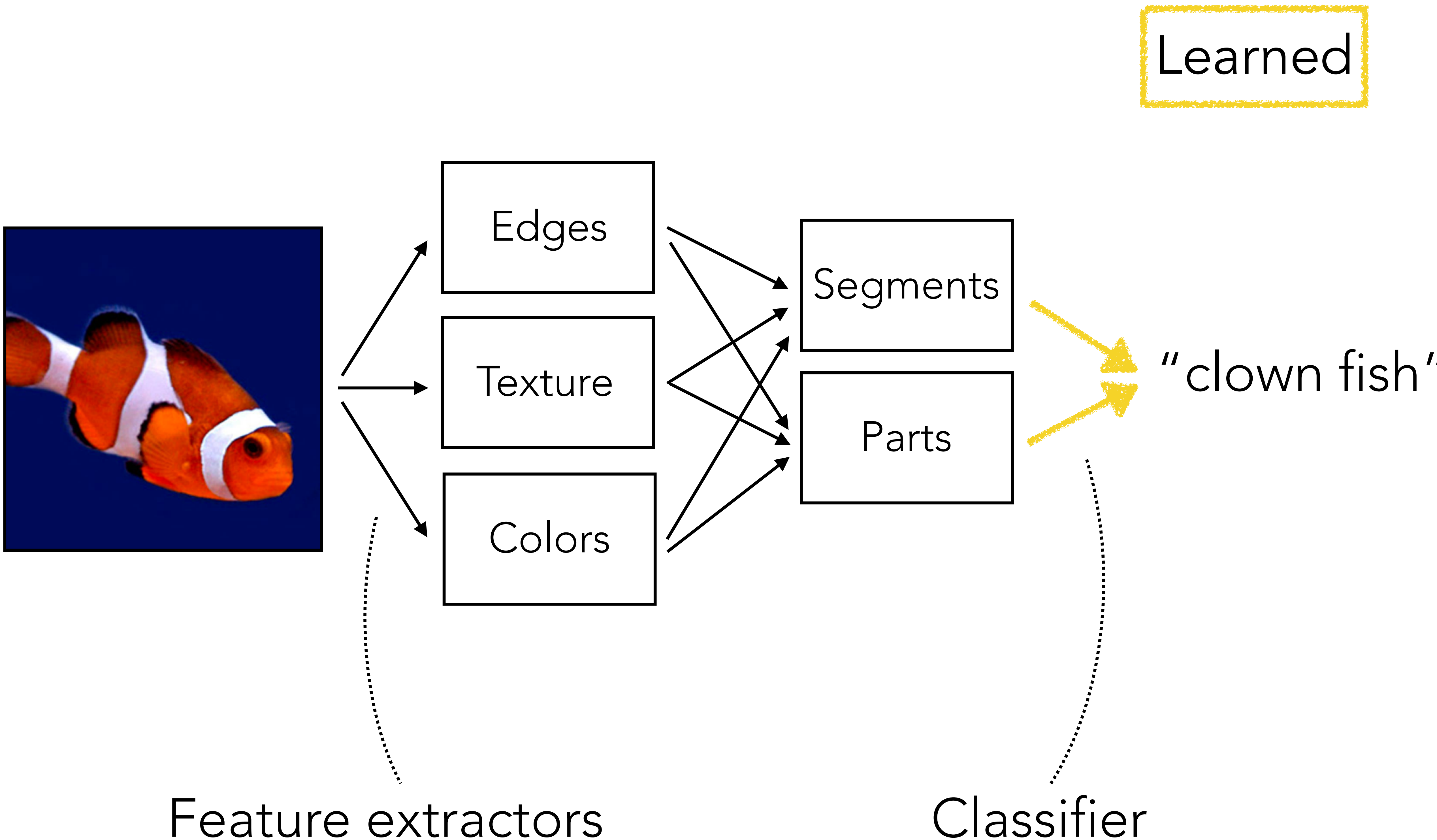
Slide by Phillip Isola

Object Recognition



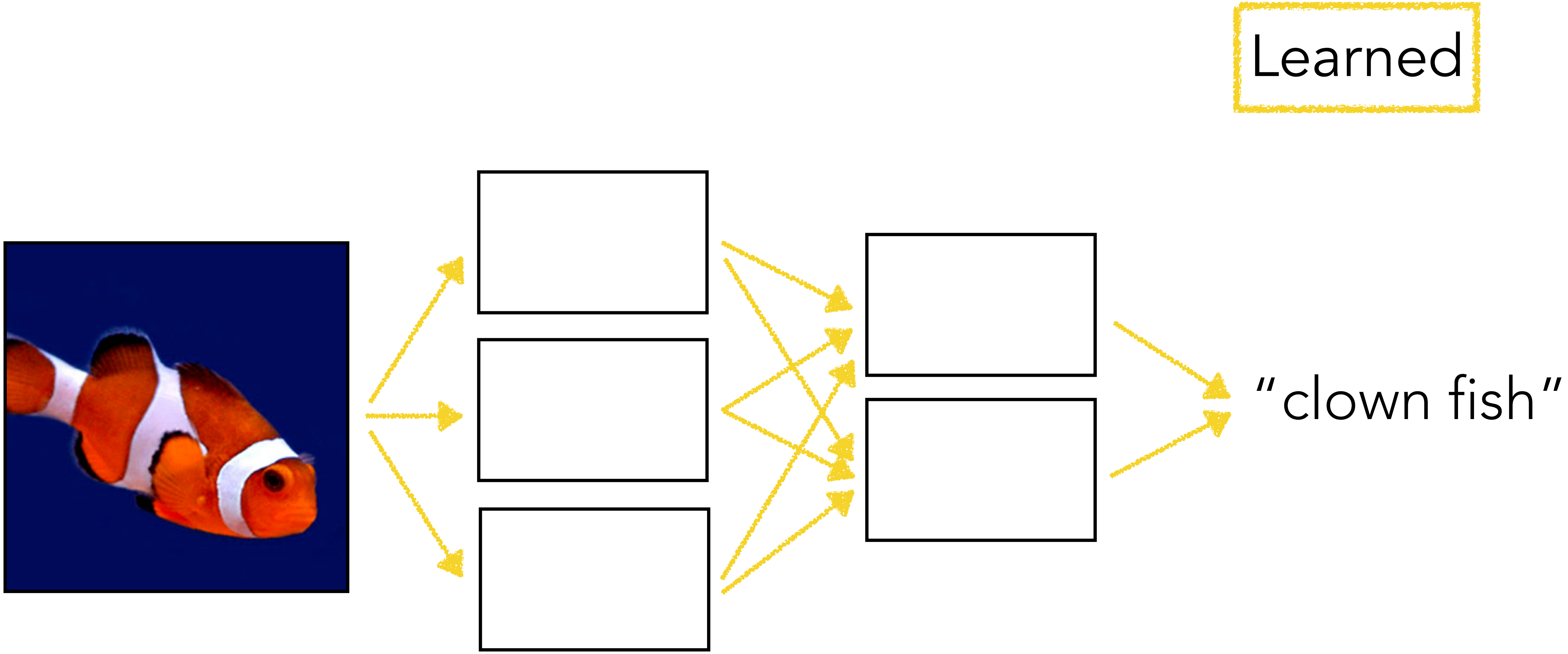
Slide by Phillip Isola

Object Recognition



Slide by Phillip Isola

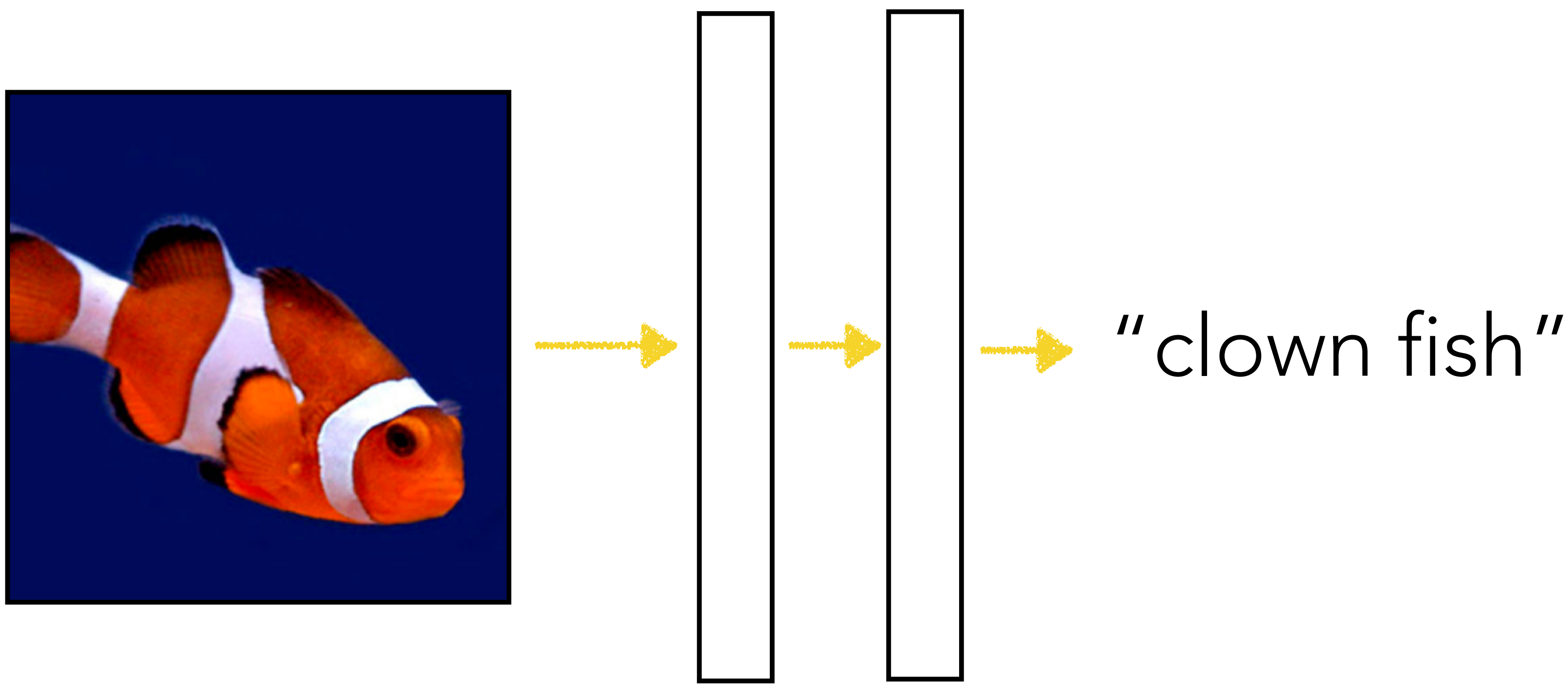
Neural Network



Slide by Phillip Isola

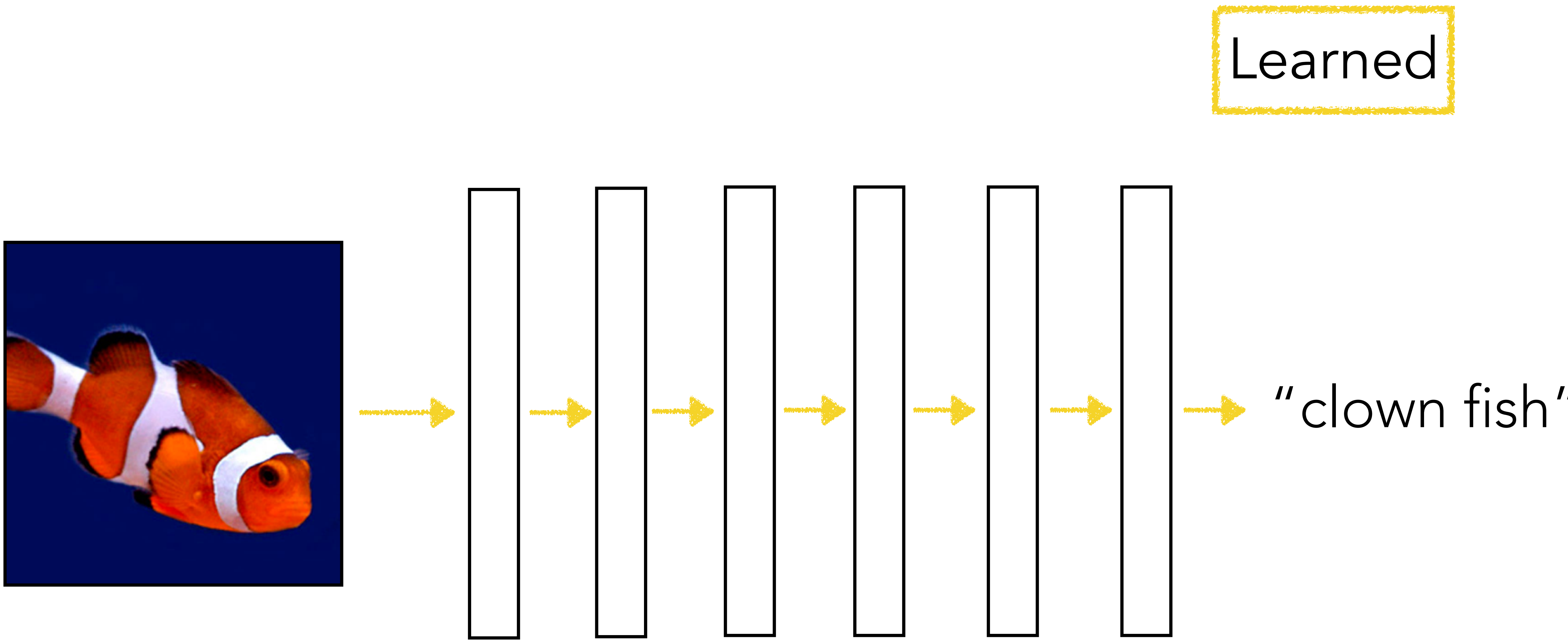
Neural Network

Learned



Slide by Phillip Isola

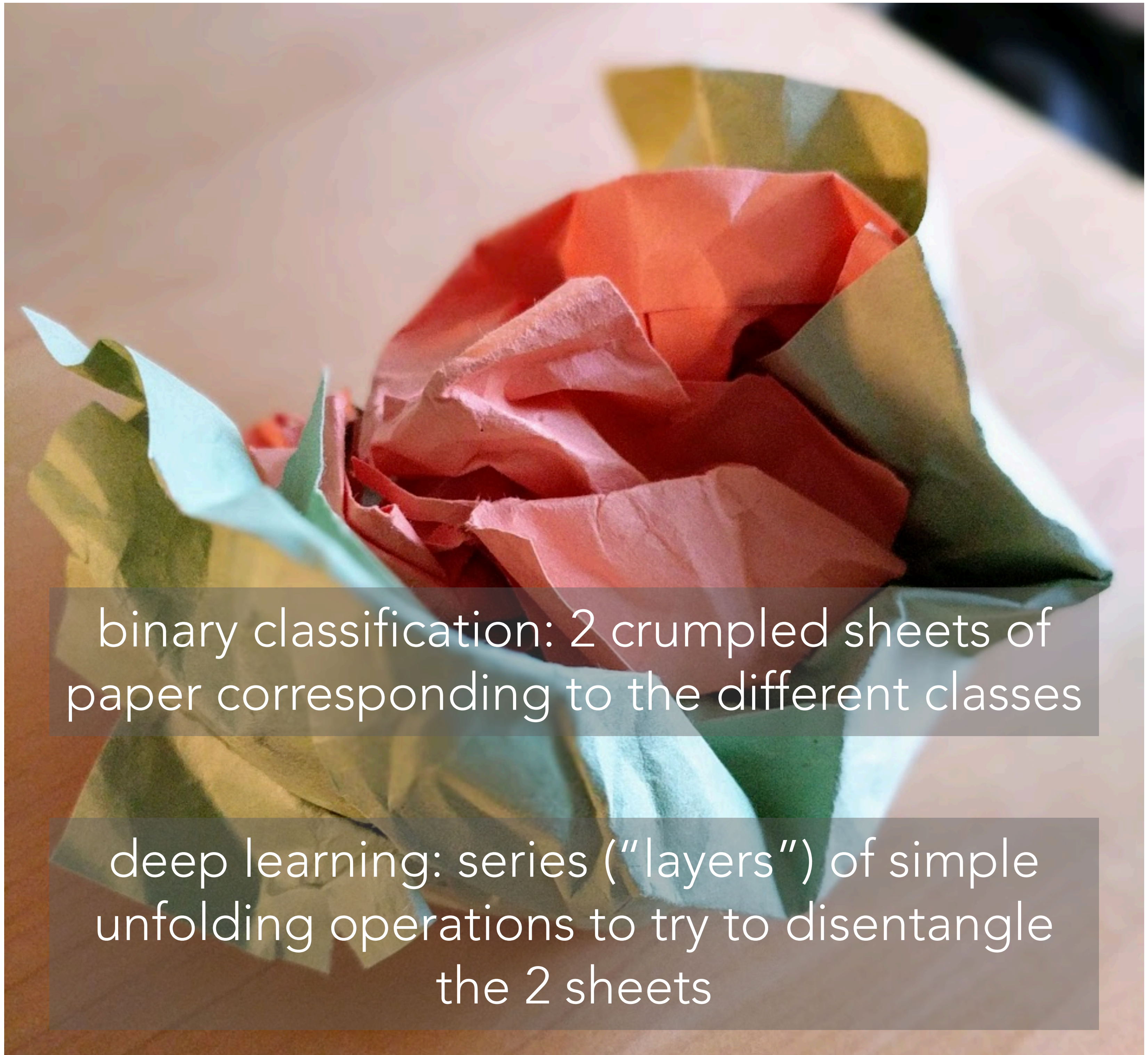
Deep Neural Network



Deep learning just refers to learning deep neural nets

Slide by Phillip Isola

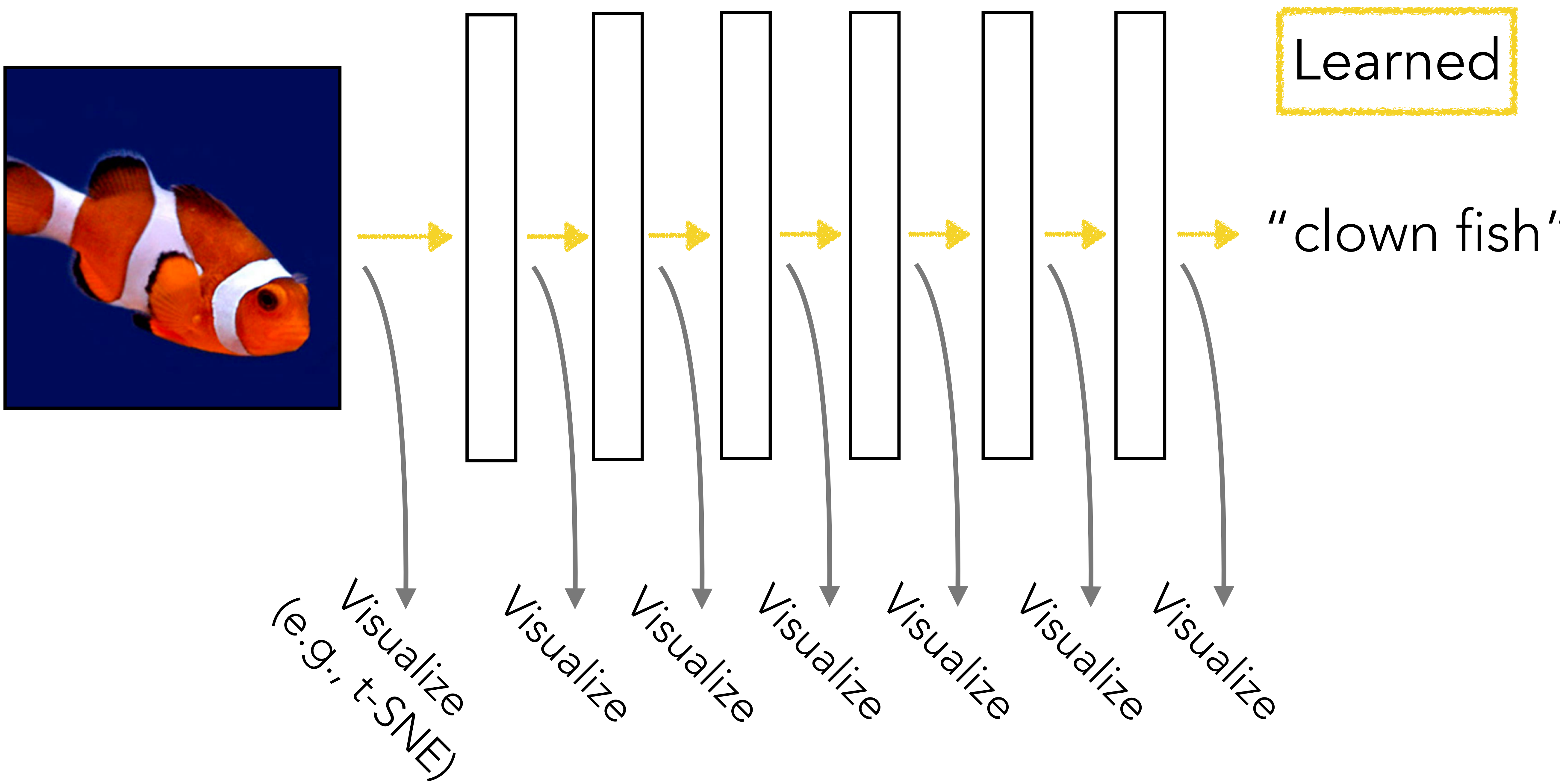
Crumpled Paper Analogy



Analogy: Francois Chollet, photo: George Chen

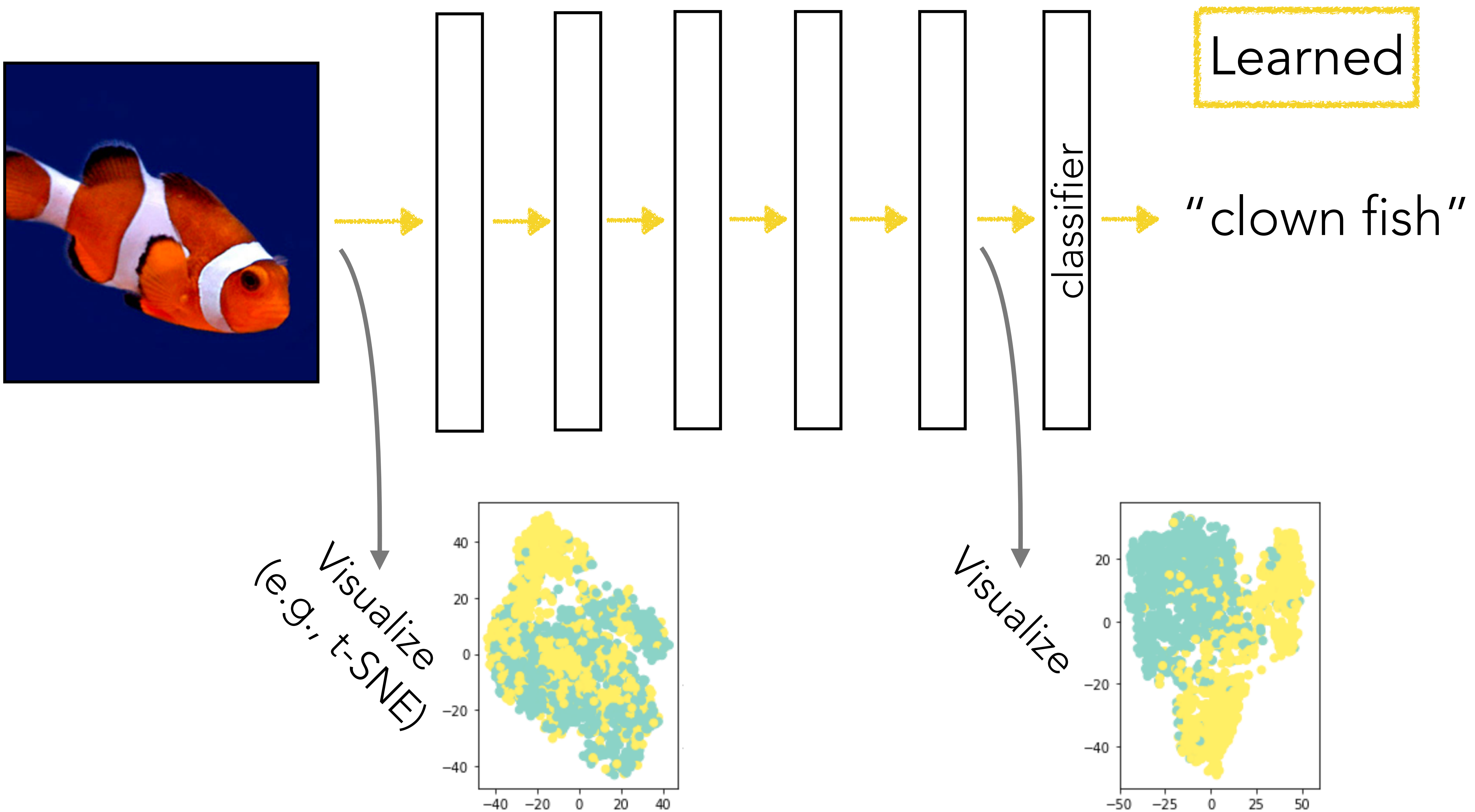
Representation Learning

Each layer's output is *another way we could represent the input data*



Representation Learning

Each layer's output is *another way we could represent the input data*



Why Does Deep Learning Work?

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

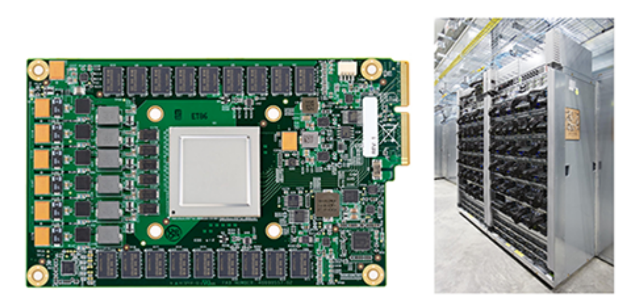
- Big data

There's even a patent from 1961 that basically amounts to a convolutional neural net for OCR

- Better hardware

CPU's
& Moore's law

GPU's



TPU's

Many companies now make dedicated hardware for deep nets (e.g., Google, Apple, Tesla)

- 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 of tokens
    - Note: video is a time series of images

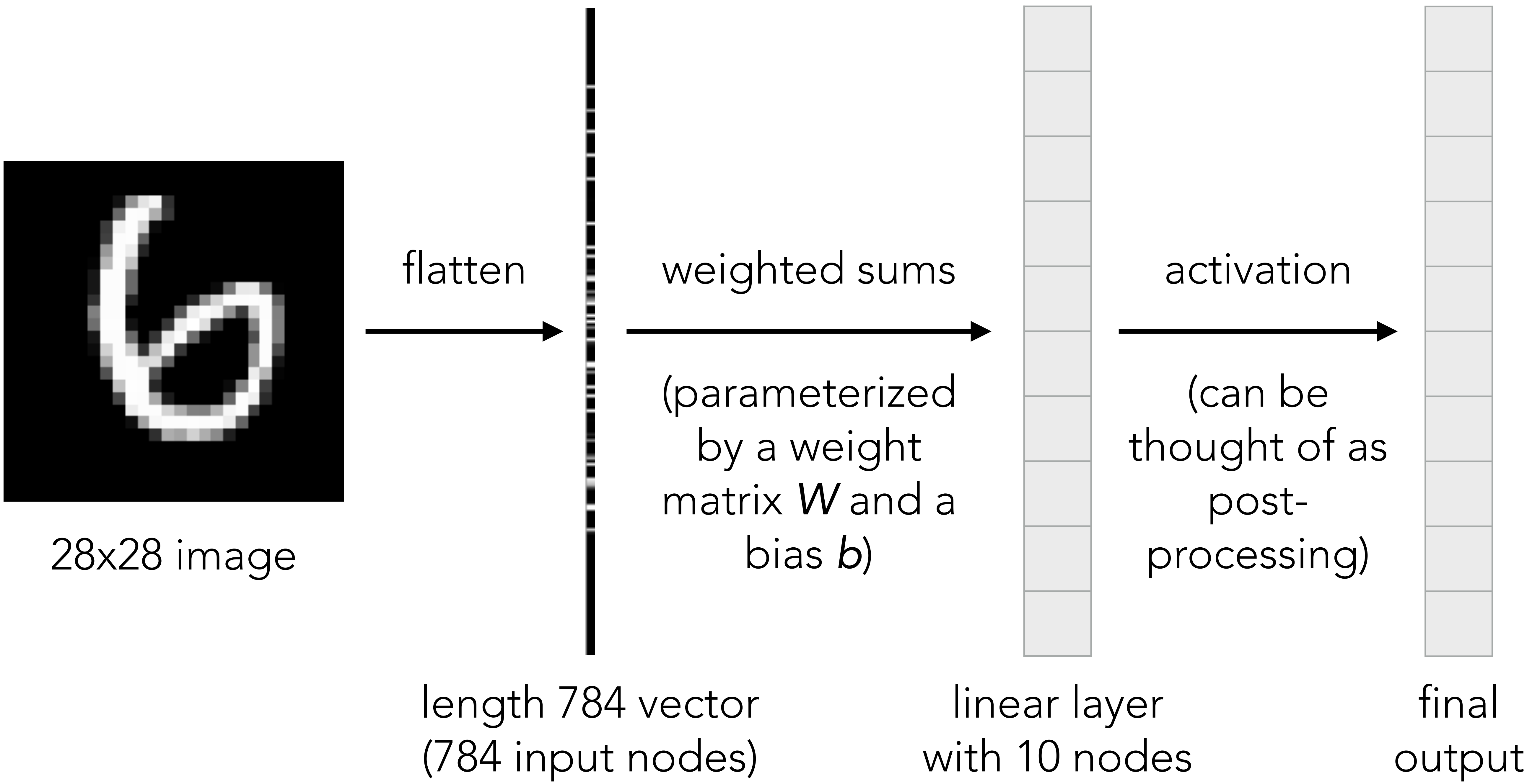
Historical note: RNNs were all the rage some years back (especially starting around the late 90s) but they've been getting replaced by **transformers** (Vaswani et al, 2017), which in turn *might* be getting replaced by **state space models** like Mamba (Gu & Dao, 2024)…

- Transformers are also for time series analysis and incorporate the concept of <u>learning how to weight previous time steps' contributions to a prediction at the current time step</u>

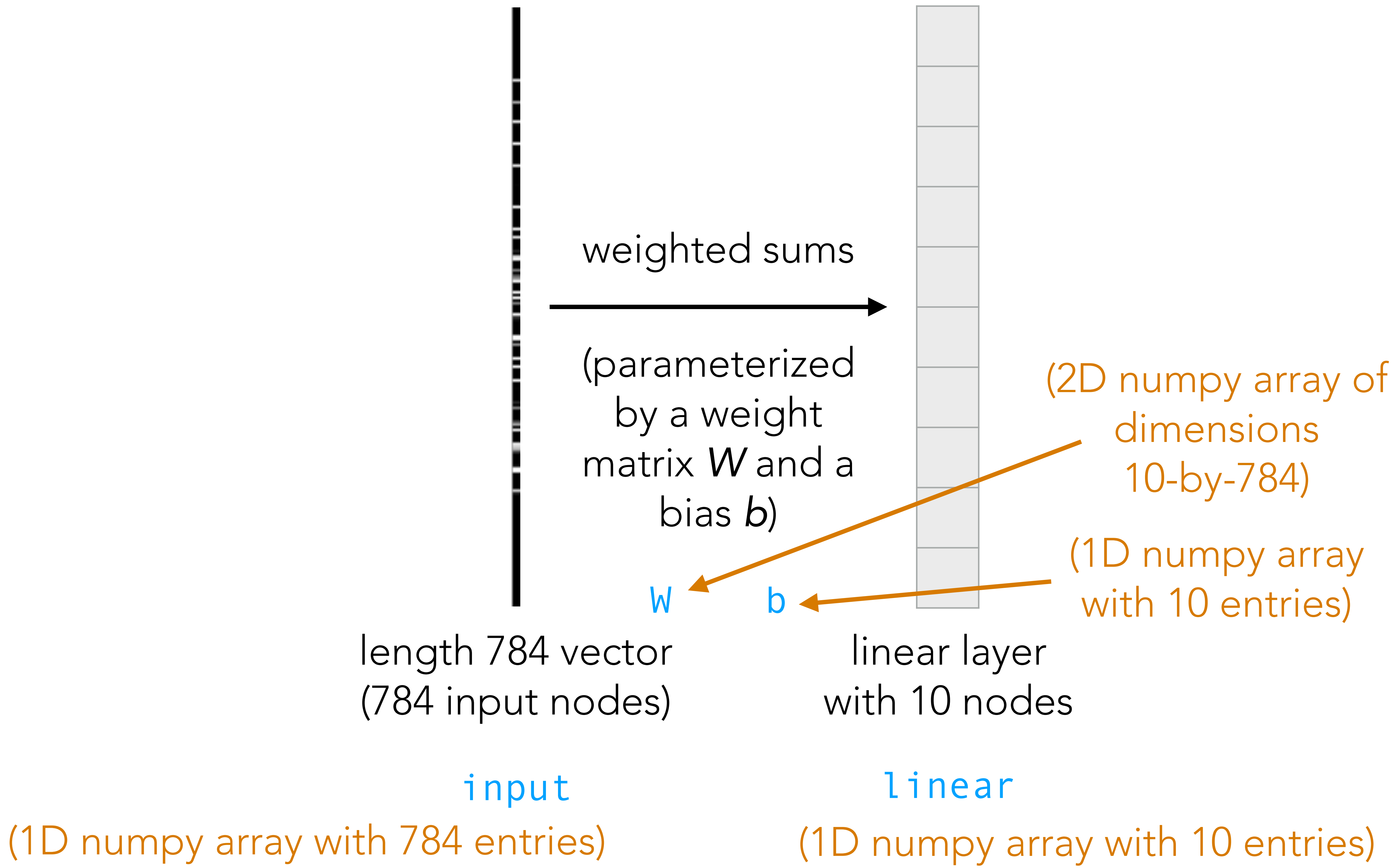
Handwritten Digit Recognition Example

Walkthrough of 2 extremely simple neural nets

Handwritten Digit Recognition

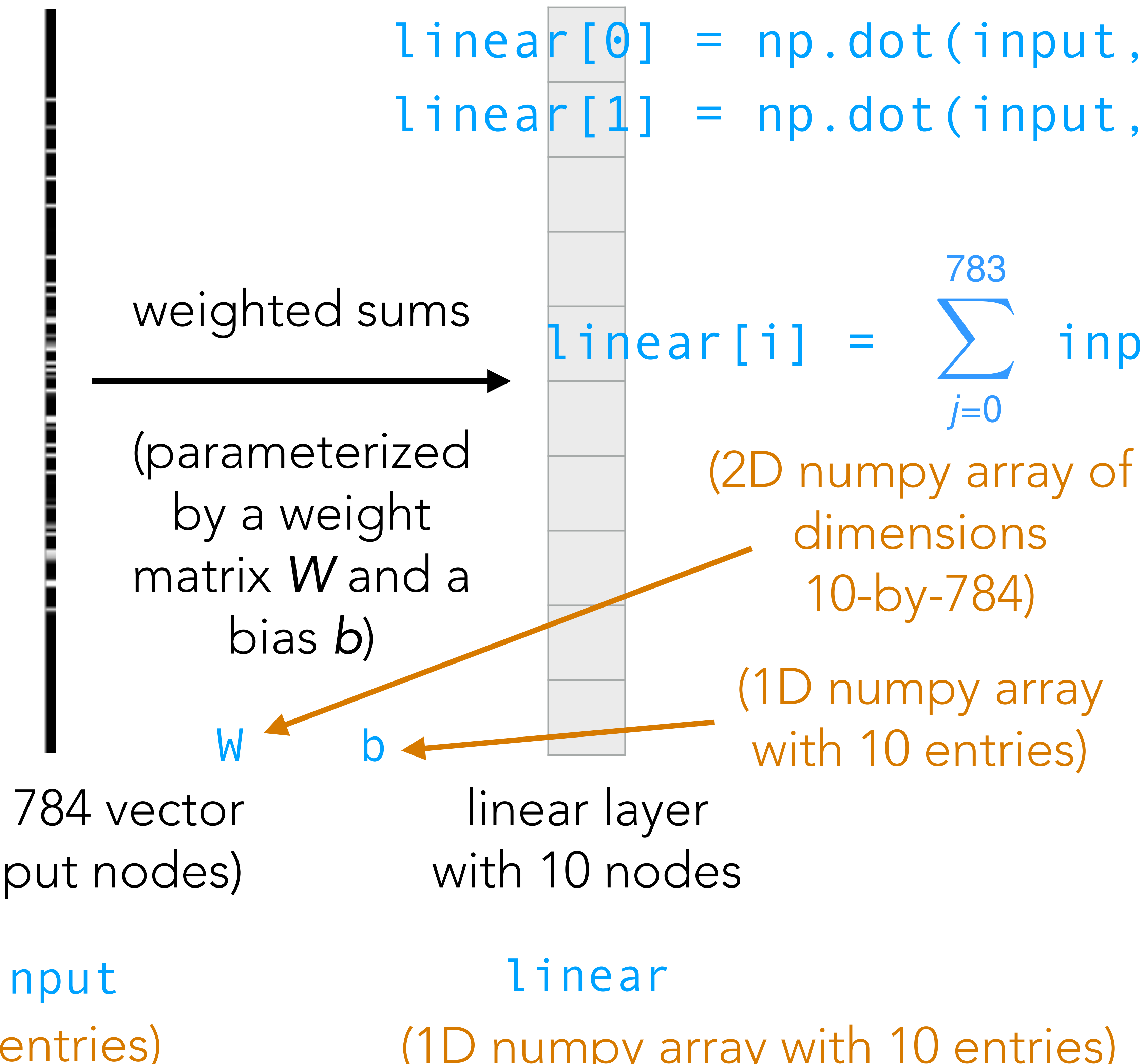


Handwritten Digit Recognition

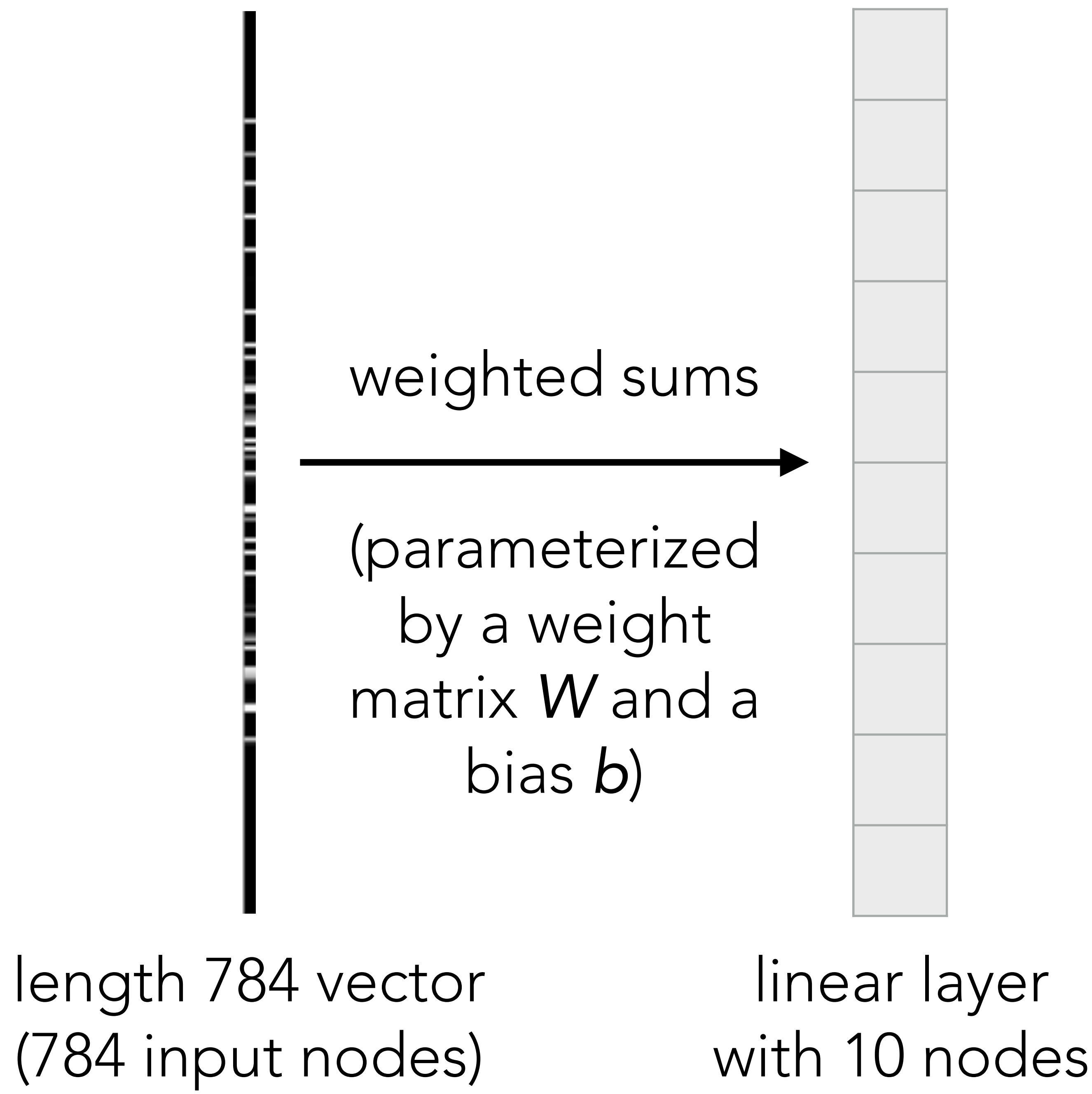


Handwritten Digit Recognition

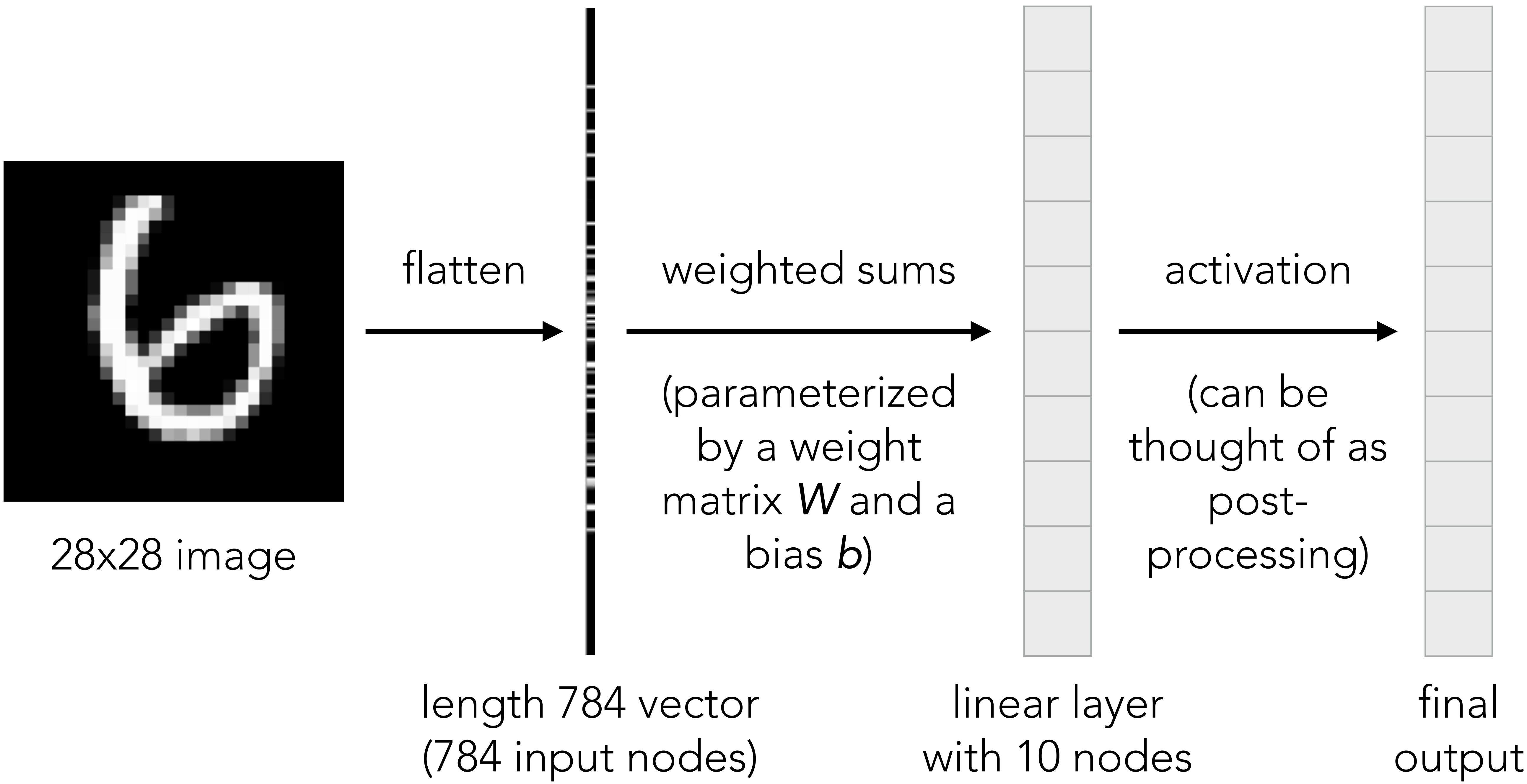
$$\texttt{linear[0] = np.dot(input, W[0, :]) + b[0]}$$
$$\texttt{linear[1] = np.dot(input, W[1, :]) + b[1]}$$
$$\vdots$$
$$\texttt{linear[i]} = \sum_{j=0}^{783} \texttt{input[j]} \times \texttt{W[i,j]} + \texttt{b[i]}$$



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

```
final = np.maximum(0, linear)
```

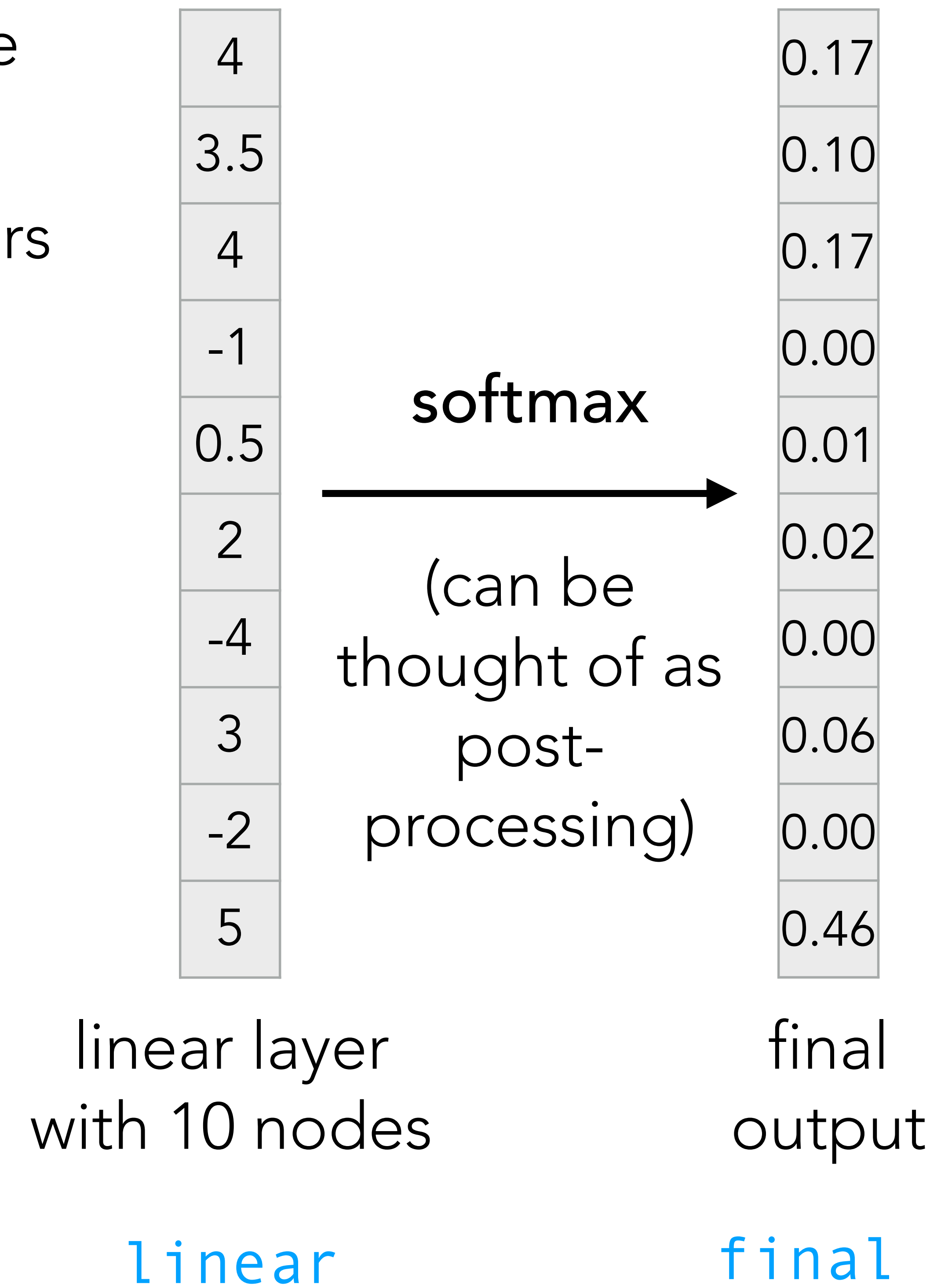
linear layer with 10 nodes		final output
4		4
3.5		3.5
4		4
-1		0
0.5	**ReLU** →	0.5
2	(can be thought of as post-processing)	2
-4		0
3		3
-2		0
5		5
`linear`		`final`

Handwritten Digit Recognition

Many different activation functions possible

Example: **softmax** converts a table of numbers into a probability distribution

```
exp = np.exp(linear)
final = exp / exp.sum()
```



Handwritten Digit Recognition

Many different activation functions possible

Example: **linear** activation does nothing

This is equivalent to there being
no activation function

```
final = linear
```