Supervised Learning: Neural networks and Deep learning — Part II

Jean Feng & Ali Shojaie

July 28-30, 2025
Summer Institute in Statistics for Big Data
University of Washington

Outline

1. What is deep learning?
2. What can DL do and not do
3. Introduction to:
    1. Dense neural networks
    2. **Convolutional neural networks**
4. How to get started using neural networks

Let's talk about image data

- Neural networks can be drastically improved by taking advantage of prior knowledge.

- **Image data has important spatial structure**. When performing visual tasks like object detection, the outcome…

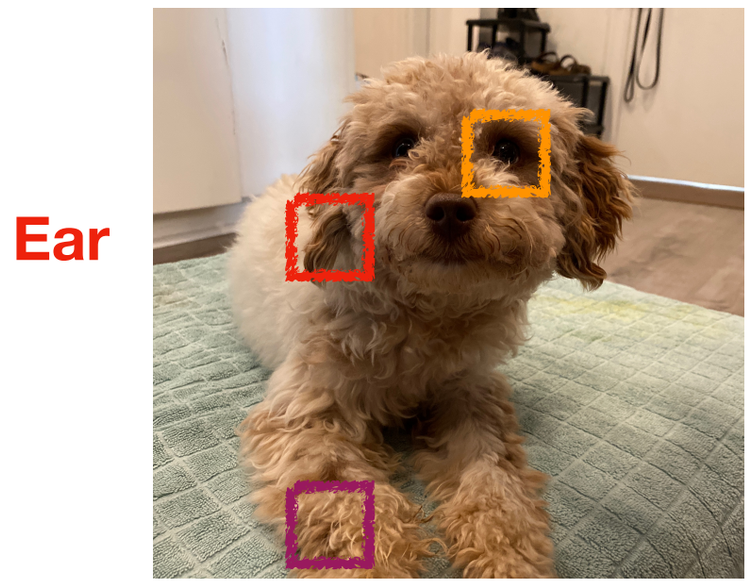
1. **Is invariant to translational shifts in the image**: the absolute location of the object in the image does not matter.

2. Is a complex function of **local image regions**

Translational Invariance

Dog Still a dog

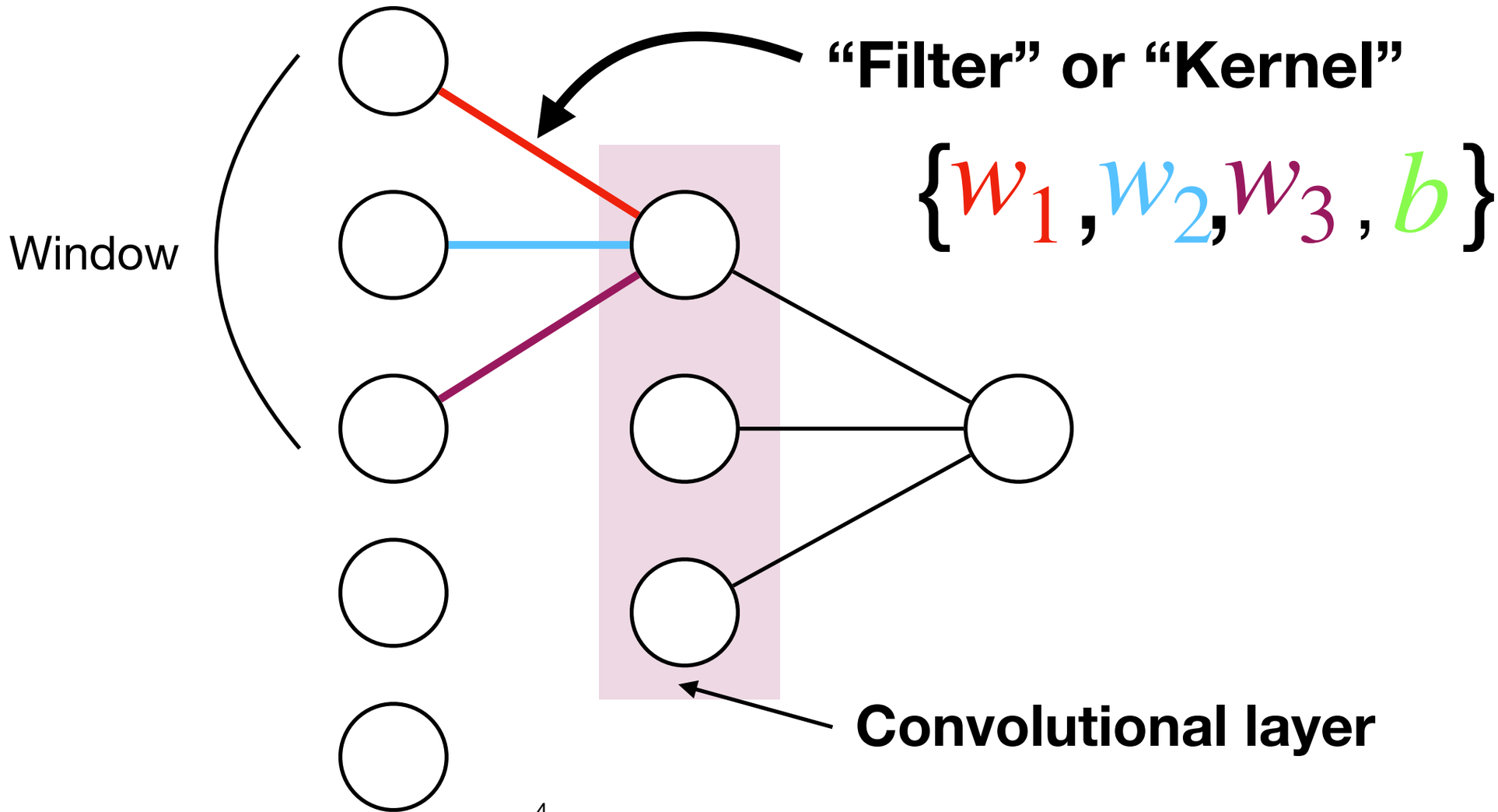
Locality



Convolutional neural networks (CNNs)

- Convolutional layers constrain the network weights based on the ideas of *translational invariance* and *locality*.

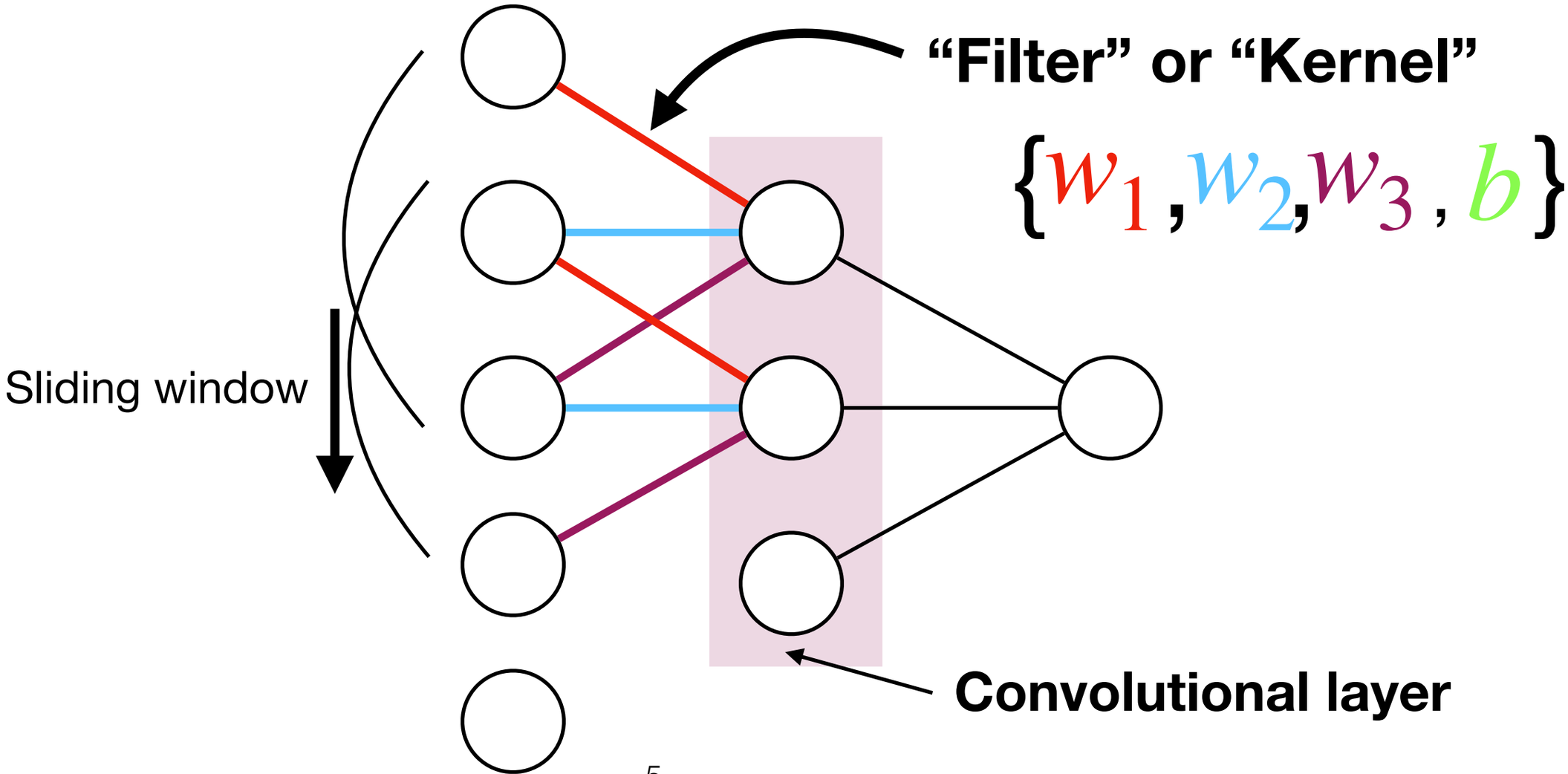
- 1D example: We use the same set of weights



Convolutional neural networks (CNNs)

- Convolutional layers constrain the network weights based on the ideas of *translational invariance* and *locality*.

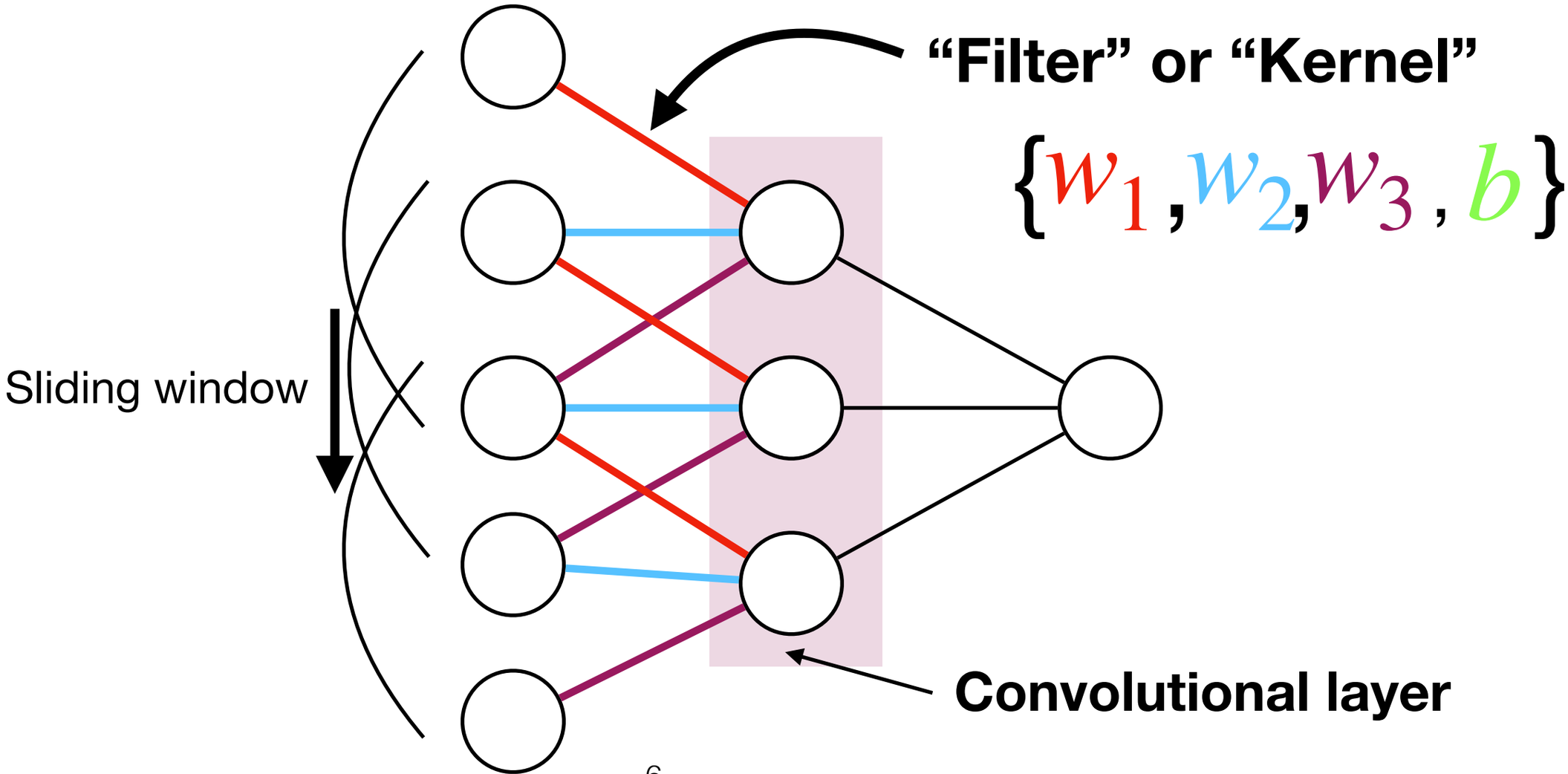
- 1D example: We use the same set of weights



Convolutional neural networks (CNNs)

- Convolutional layers constrain the network weights based on the ideas of *translational invariance* and *locality*.

- 1D example: We use the same set of weights



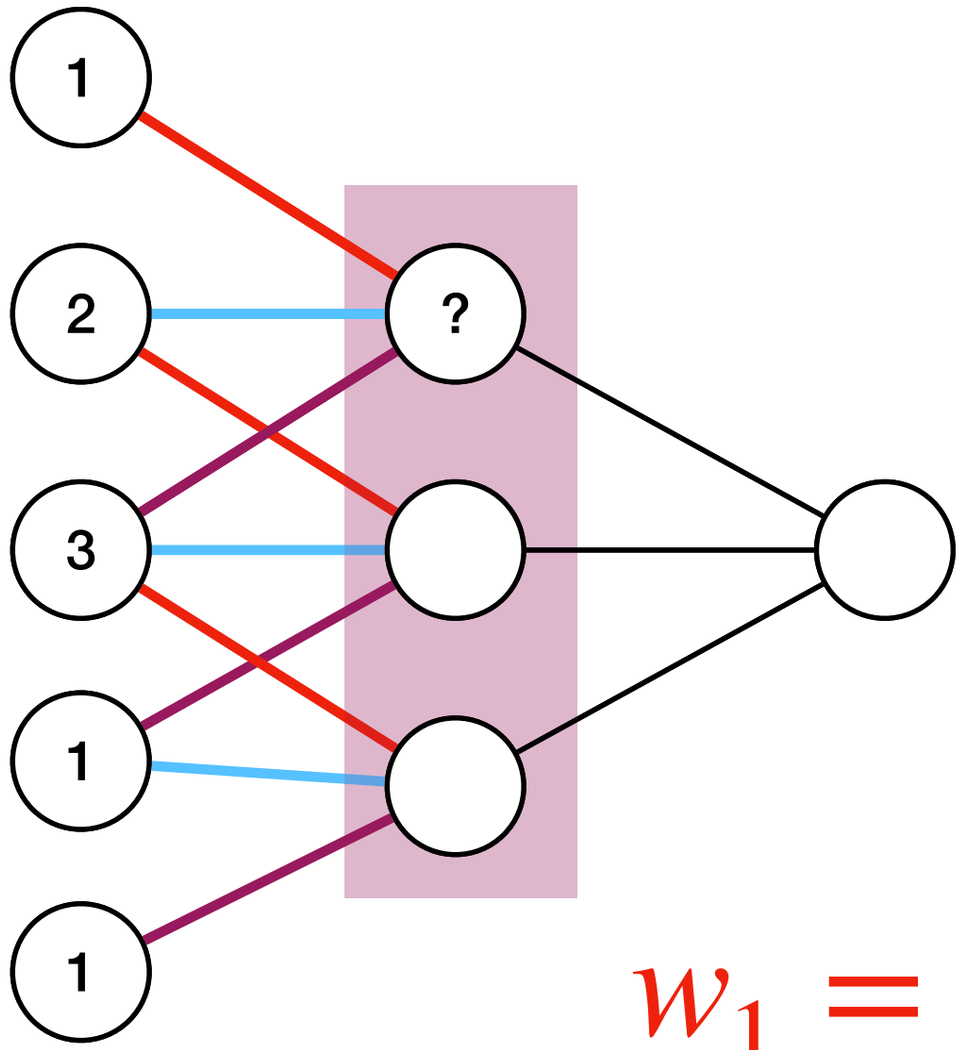
Quick Quiz

Q: What is the value in the 1st hidden node?

A: 1 * 1 + 2 * 2 + 3 * 1 + 0 = 8

B: 2 * 1 + 3 * 2 + 1 * 1 + 0 = 9

C: 1 * 1 + 2 * 1 + 3 * 1 + 0 = 6



$w_1 = 1$

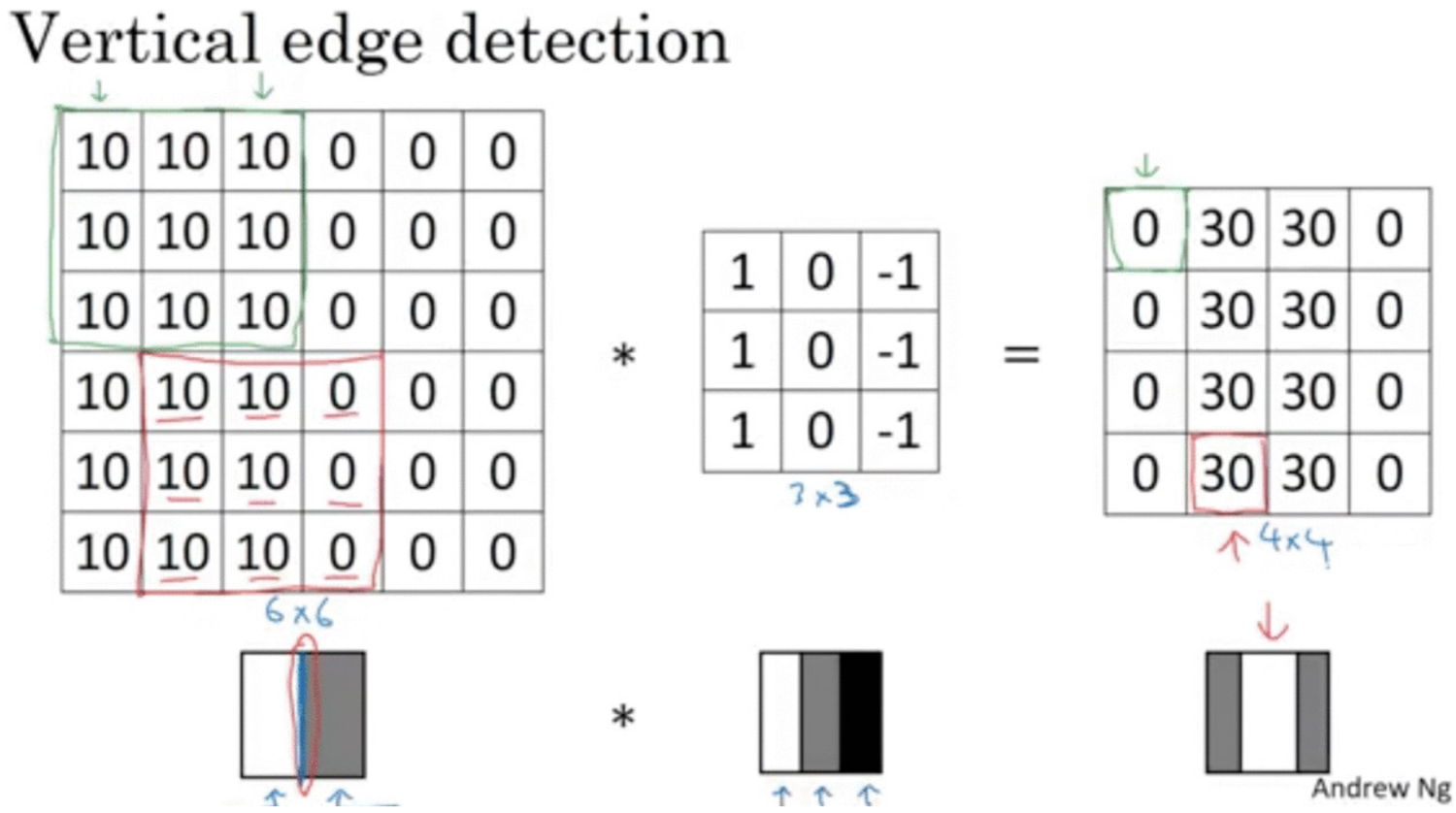
$w_2 = 2$

$w_3 = 1$

$b = 0$

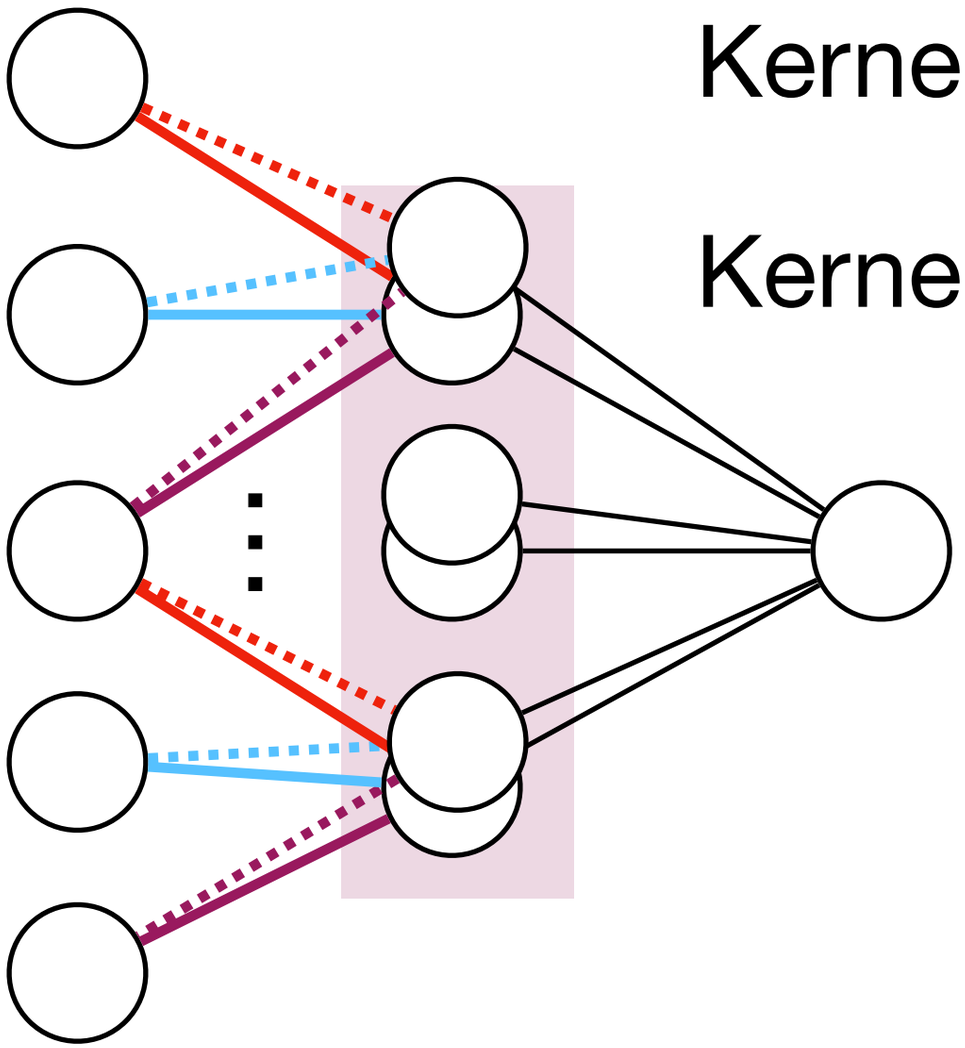
Convolutional neural networks (CNNs)

- Extending this to images, we have something like this:



CNNs: Multiple output channels

- One kernel is usually not enough to summarize a region.
- If we want to capture more information from each region, add more kernels!
- Each kernel corresponds to a different output **channel**.



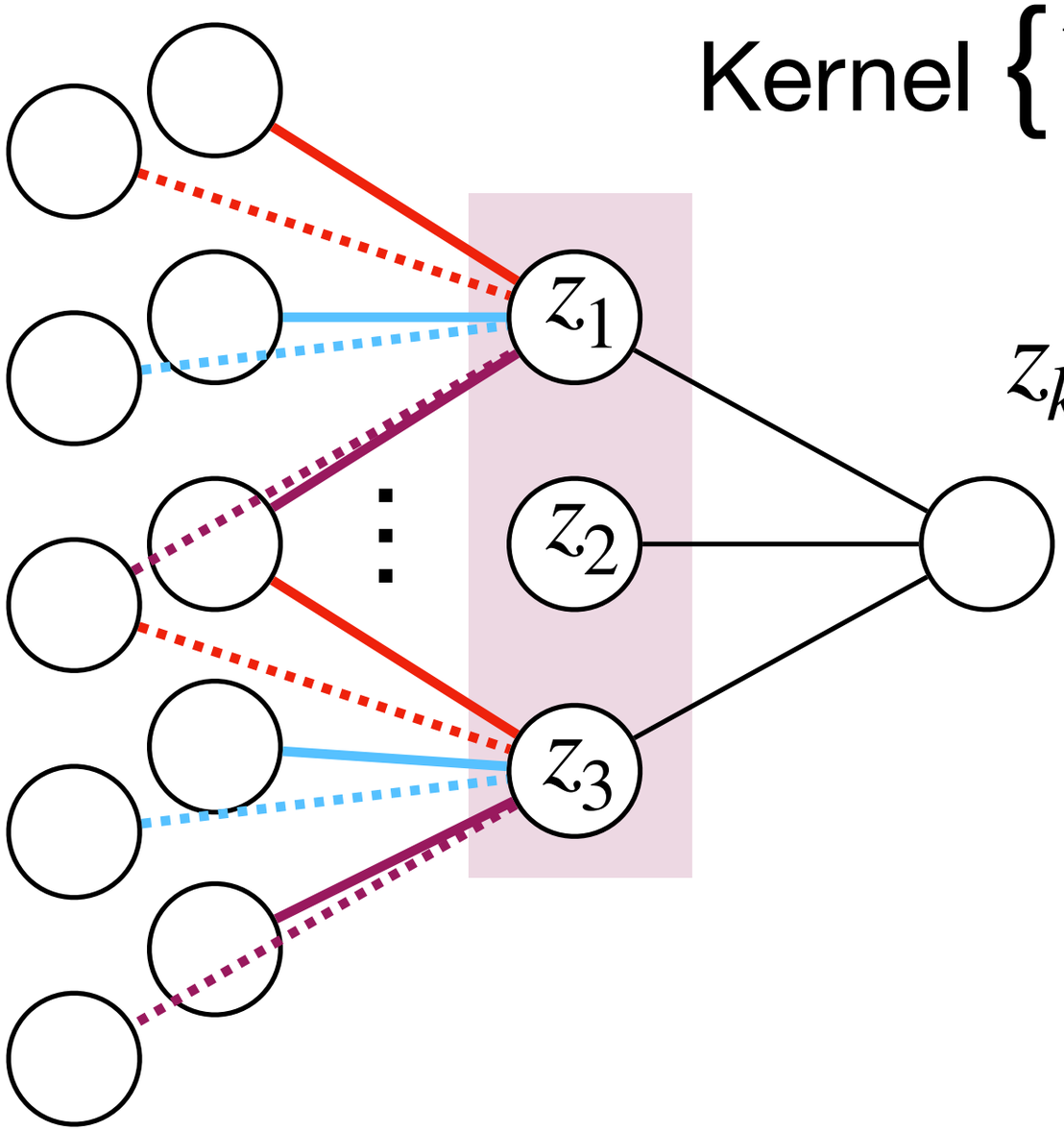
Kernel 1 $\{W^{(1)} \in \mathbb{R}^d \quad , b^{(1)} \in \mathbb{R}\}$

Kernel 2 $\{W^{(2)} \in \mathbb{R}^d \quad , b^{(2)} \in \mathbb{R}\}$

$$z_k^{(l)} = \sum_{i=1}^{d} W_i^{(l)} x_{k+i} + b^{(l)}$$

CNNs: Multiple input channels

- An input can have c channels. We can then express the kernel as a matrix.



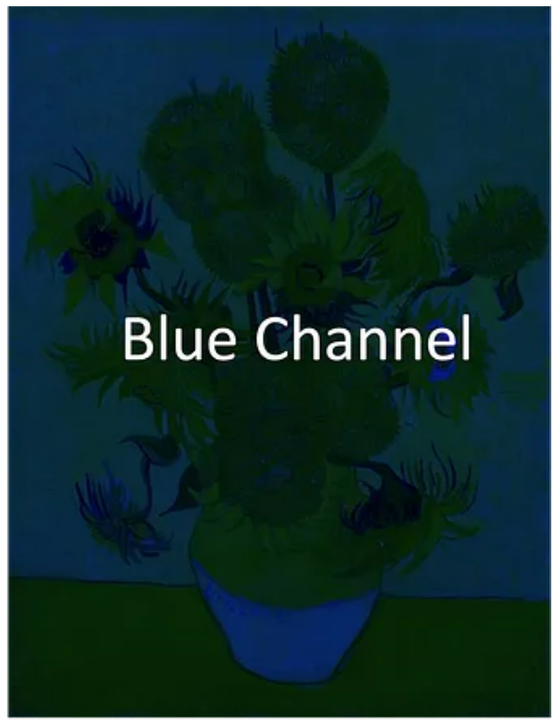
Kernel $\{W \in \mathbb{R}^{c \times d}, b \in \mathbb{R}\}$

$$z_k = \sum_{j=1}^{c} \sum_{i=1}^{d} W_{i,j} x_{k+i,j} + b$$

Channels

The term “channel” comes from images.

Color images are composed of 3 channels: RGB.



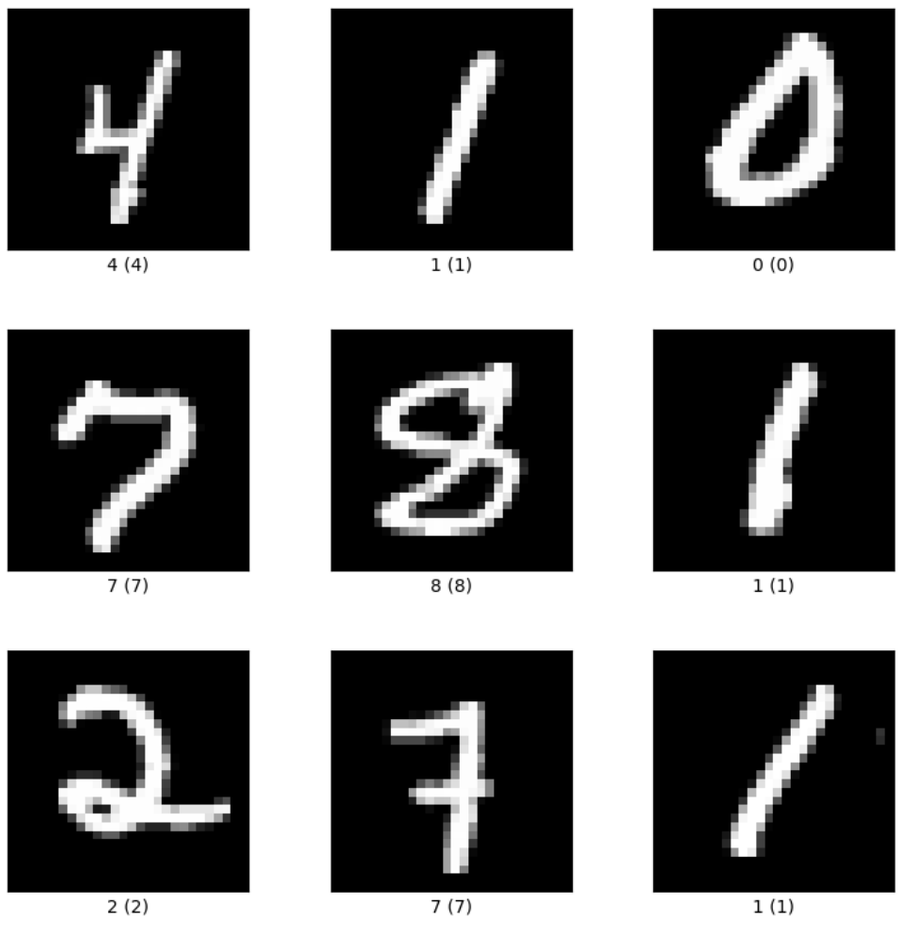
Quiz

- How many channels are needed to represent a black and white image (or grayscale image)?

- A: 1 channel

- B: 2 channels

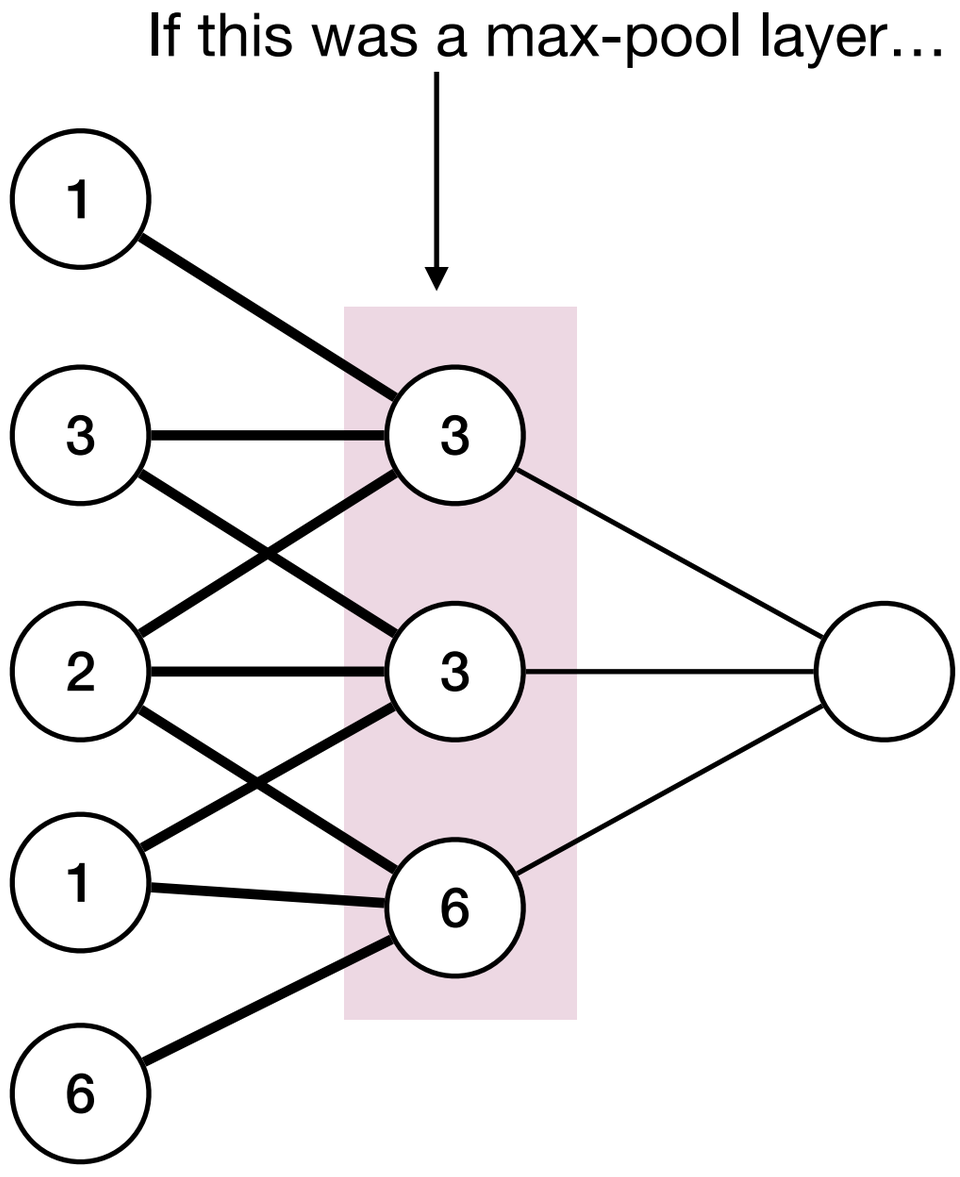
- C: 3 channels



Pooling layers

- Pooling operators similar to convolutional layers but contain no parameters. They usually just take the maximum or average over a window.

- Purpose is to decrease the number of nodes, on top of translational invariance and locality.

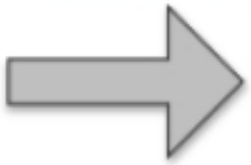


Pooling layers for images

Single depth slice of input

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

Max pool with
2×2 filters and stride 2

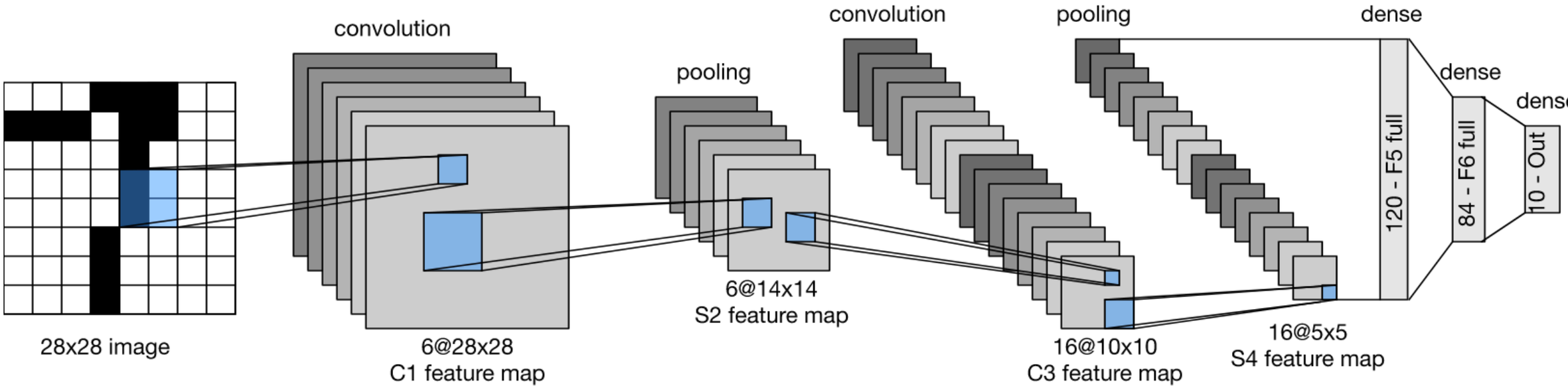


'Pooled' output

6	9
3	4

Convolutional neural networks

Putting this all together…



LeNet-5 (Lecun 1998)

AlexNet

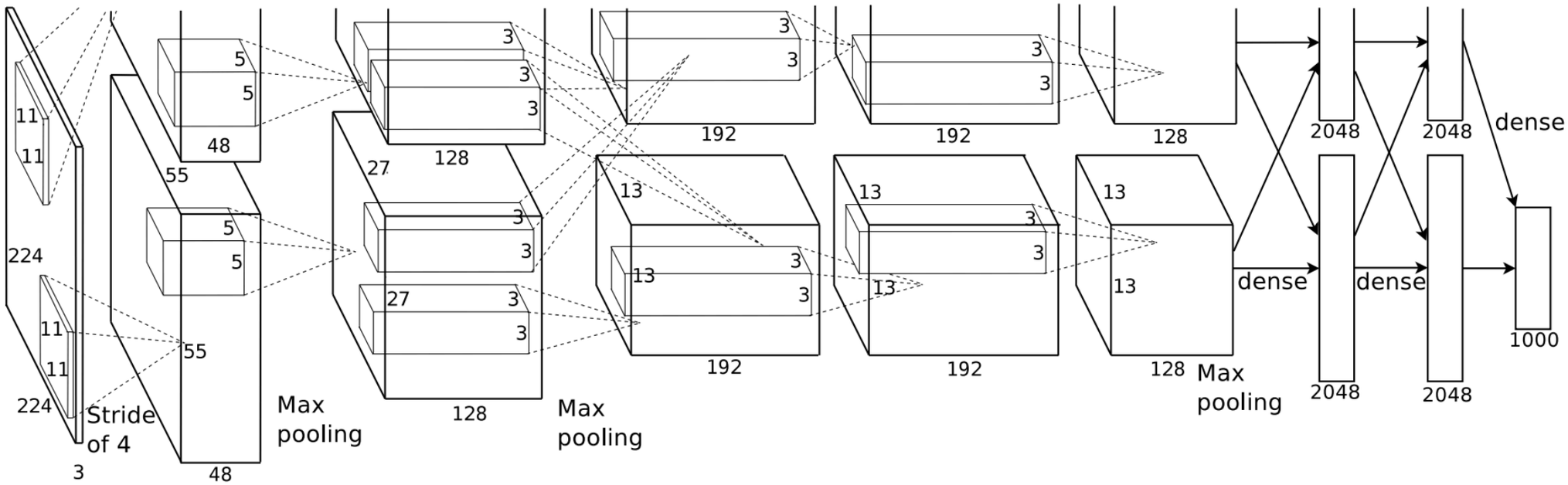
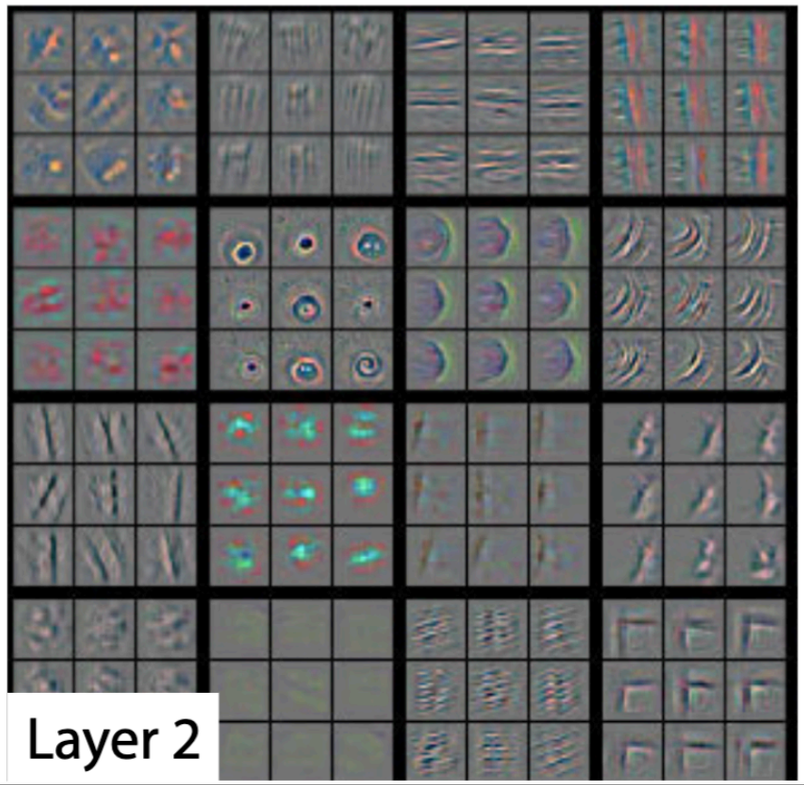


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

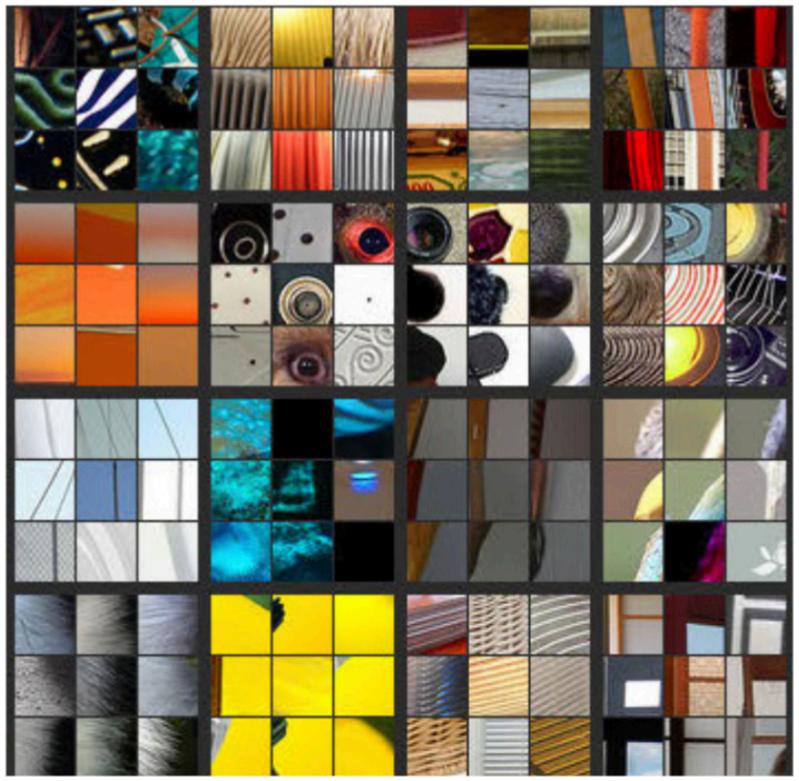
AlexNet

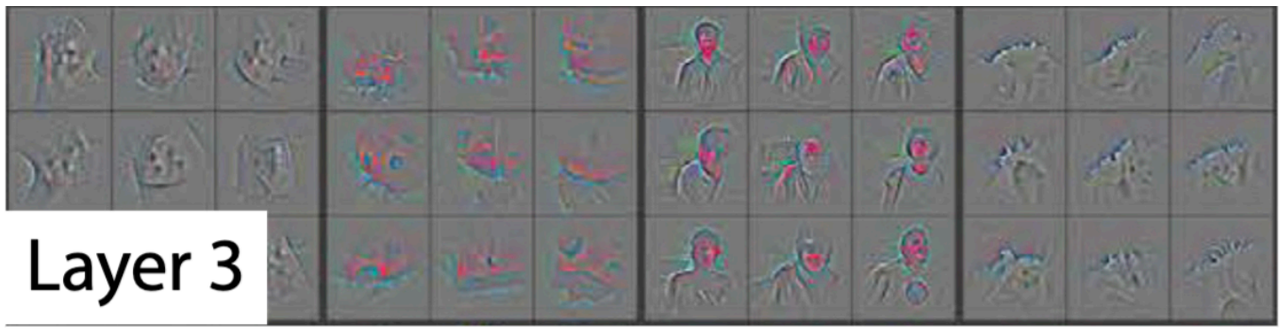
"Representation learning"

Layer 1

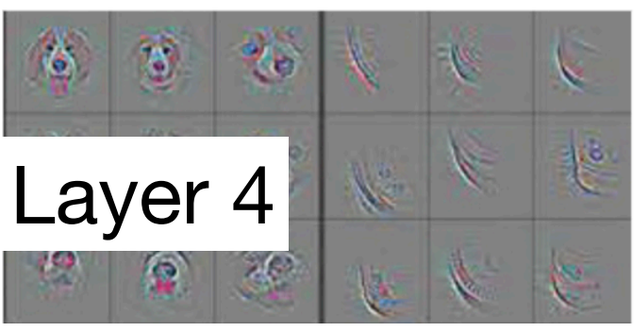


Layer 2

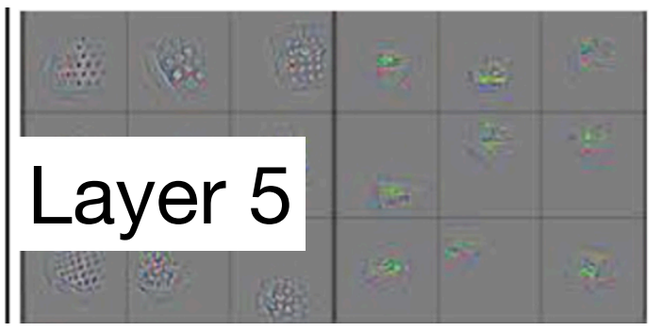




Layer 3



Layer 4



Layer 5

Zeiler, Matthew D., and Rob Fergus. 2014. "Visualizing and Understanding Convolutional Networks" In *Computer Vision ECCV 2014*

Summary: CNNs

- CNNs are neural networks that are specially designed to handle data where the translational invariance and locality properties hold. This prior knowledge is encoded in the form of convolutional and pooling layers.

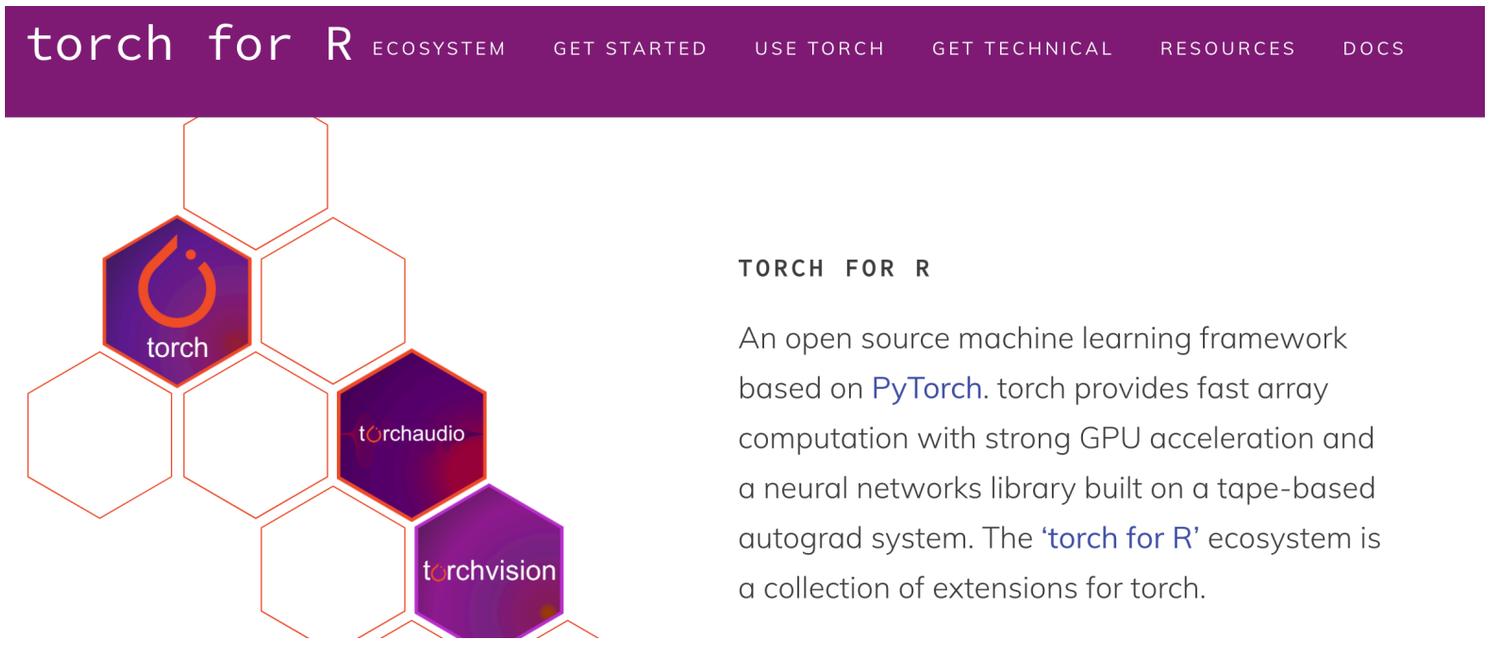
- They are useful for analyzing image data, molecular sequences, biological networks, and more.

Outline

1. What is deep learning?
2. What can DL do and not do
3. Introduction to:
    1. Dense neural networks
    2. Convolutional neural networks
4. **How to get started using neural networks**

Software for deep learning

- There are a lot of good software tools and tutorials online:
  - Python: PyTorch, TensorFlow, Keras
  - R: TensorFlow for R, Torch for R (https://torch.mlverse.org/)



Before you get too excited…

Before you use a neural network, always try something simpler!

- Neural networks take extra time and care. The amount of effort is not necessarily worth it. Generally, we find that…

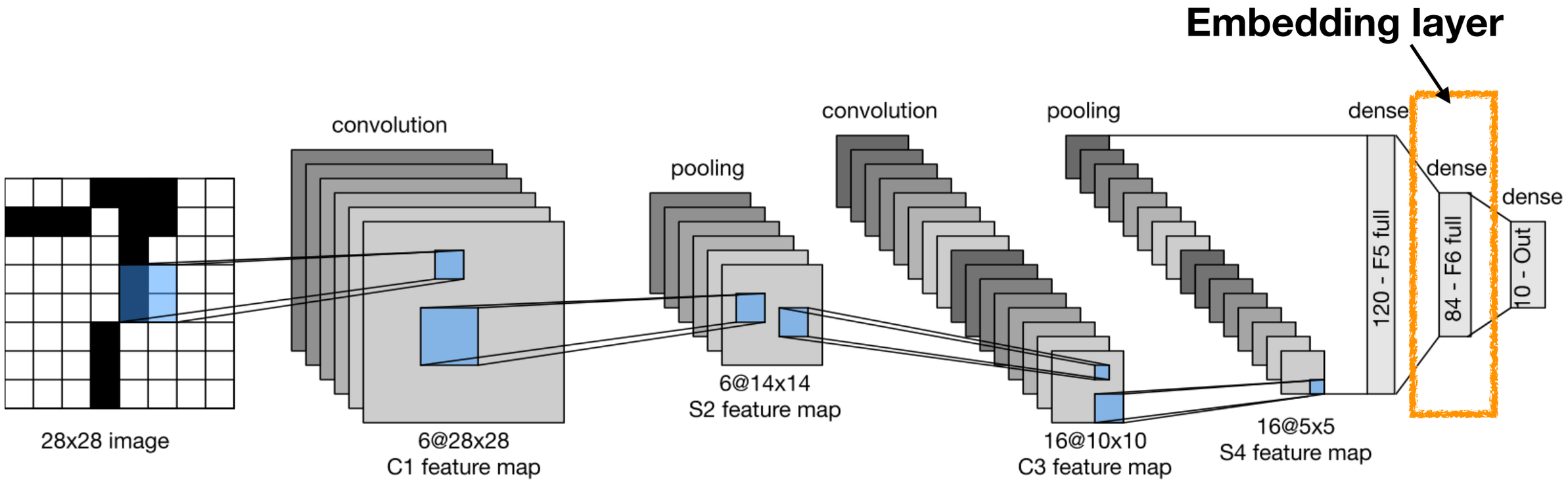
    - For datasets with a small number of samples, simple methods like logistic regression often do as well as (and usually better than) a neural network.

    - For high-dimensional datasets, neural networks often struggle unless you properly employ regularization/feature selection.

- However, if there is a pre-trained neural network for a similar task, you can try to fine-tune this pre-trained model for the task of interest.

Easiest way to get started

- Download pretrained networks and use them for feature extraction.



Useful References

- Textbook: “Dive into Deep Learning” https://d2l.ai/
- Online courses: Coursera’s Deep learning courses
- A sequence of posts on torch on R:
    - https://blogs.rstudio.com/ai/posts/2020-10-01-torch-network-from-scratch/
    - https://blogs.rstudio.com/ai/posts/2020-10-07-torch-modules/
    - https://blogs.rstudio.com/ai/posts/2020-10-09-torch-optim/
    - https://blogs.rstudio.com/ai/posts/2020-10-19-torch-image-classification/

Try it out!

https://docs.pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Introduction to torch in R

- **Tensor**: a multi-dimensional array, e.g.
  - 1-D tensor for tabular data
  - 3-D tensor for images
- **Module**: A neural network layer, e.g.
  - nn_conv1d(): A convolutional layer over 1 dimension
- **Optimizers**: Optimization algorithms
- **Dataloaders**: Objects that load data

Torch tensors

```
> library(torch)
> # a 1d vector of length 2
> t <- torch_tensor(c(1, 2))
> t
torch_tensor
 1
 2
[ CPUFloatType{2} ]
```

```
> # a 3x3 tensor (matrix)
> t <- torch_tensor(rbind(c(1,2,0), c(3,0,0), c(4,5,6)))
> t
torch_tensor
 1  2  0
 3  0  0
 4  5  6
[ CPUFloatType{3,3} ]
```

```
> t <- torch_zeros(3, 5, 5)
> t
torch_tensor
(1,.,.) =
  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

(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

(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
[ CPUFloatType{3,5,5} ]
```

Torch tensors

Converting torch tensor back to R

```
> t <- torch_tensor(matrix(1:9, ncol = 3, byrow = TRUE))
> as_array(t)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Indexing

```
> t <- torch_tensor(rbind(c(1,2,3), c(4,5,6)))
> t
torch_tensor
 1  2  3
 4  5  6
[ CPUFloatType{2,3} ]
>
> # a single value
> t[1, 1]
torch_tensor
1
[ CPUFloatType{} ]
>
> # first row, all columns
> t[1, ]
torch_tensor
 1
 2
 3
[ CPUFloatType{3} ]
```

Modules

Creating a linear module

```
> l <- nn_linear(3, 1)
> l
An `nn_module` containing 4 parameters.

── Parameters ──────────────────────────────────────────────────────
• weight: Float [1:1, 1:3]
• bias: Float [1:1]
```

```
> l$parameters
$weight
torch_tensor
 0.1939  0.0619  0.5187
[ CPUFloatType{1,3} ]

$bias
torch_tensor
 0.1350
[ CPUFloatType{1} ]
```

Calling a module given input data

```
> data  <- torch_randn(10, 3)
> out <- l(data)
> out
torch_tensor
 0.1826
 0.3797
 0.1410
-0.3324
 0.4497
 1.2186
-0.2790
 0.0586
 0.3796
-0.4895
[ CPUFloatType{10,1} ]
```

Modules

Defining a model as a sequence of modules

```
> model <- nn_sequential(
+     nn_linear(3, 16),
+     nn_relu(),
+     nn_linear(16, 1)
+ )
>
> model
An `nn_module` containing 81 parameters.

── Modules ─────────────────────────────────────
• 0: <nn_linear> #64 parameters
• 1: <nn_relu> #0 parameters
• 2: <nn_linear> #17 parameters
```

Calling a model given input data

```
> out <- model(data)
> out
torch_tensor
-0.3374
-0.2348
-0.5673
-0.2602
-0.1857
-0.1011
-0.3001
-0.1868
-0.2397
-0.3328
[ CPUFloatType{10,1} ]
```

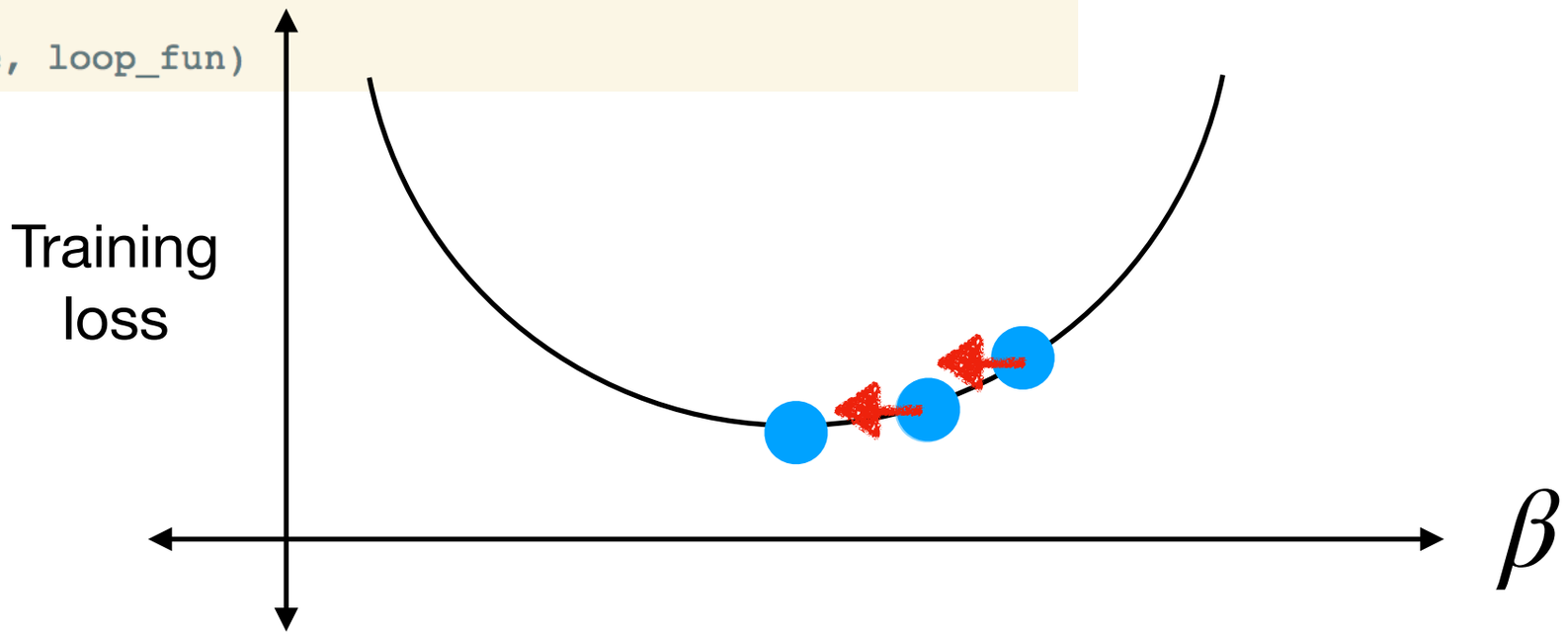
Indexing into the model

```
> model[[1]]
An `nn_module` containing 64 parameters.

── Parameters ──────────────────────────────────
• weight: Float [1:16, 1:3]
• bias: Float [1:16]
```

Optimizers

```
> optimizer <- optim_adam(model$parameters, lr = 0.01)
> optimizer
<optim_adam>
  Inherits from: <torch_Optimizer>
  Public:
    add_param_group: function (param_group)
    clone: function (deep = FALSE)
    defaults: list
    initialize: function (params, lr = 0.001, betas = c(0.9, 0.999), eps = 1e-08,
    param_groups: list
    state: list
    step: function (closure = NULL)
    zero_grad: function ()
  Private:
    step_helper: function (closure, loop_fun)
```



Dataloader

```
> train_small
<dataset>
  Public:
    .getitem: function (idx)
    .length: function ()
    clone: function (deep = FALSE)
    dataset: tiny_imagenet, image_folder, folder, dataset, R6
    indices: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ...
    initialize: function (dataset, indices)
```

```
> train_dl <- dataloader(train_small, batch_size=ntrain, shuffle = F)
> train_dl
<dataloader>
  Public:
    .auto_collation: active binding
    .dataset_kind: map
    .index_sampler: active binding
    .iter: function ()
    .length: function ()
    batch_sampler: utils_sampler_batch, utils_sampler, R6
    batch_size: 1500
    clone: function (deep = FALSE)
    collate_fn: function (batch)
    dataset: dataset, R6
    drop_last: FALSE
    generator: NULL
    initialize: function (dataset, batch_size = 1, shuffle = FALSE, sampler = NULL,
    multiprocessing_context: NULL
    num_workers: 0
    pin_memory: FALSE
    sampler: utils_sampler_sequential, utils_sampler, R6
    timeout: -1
    worker_init_fn: NULL
```