Neural Network Architectures I

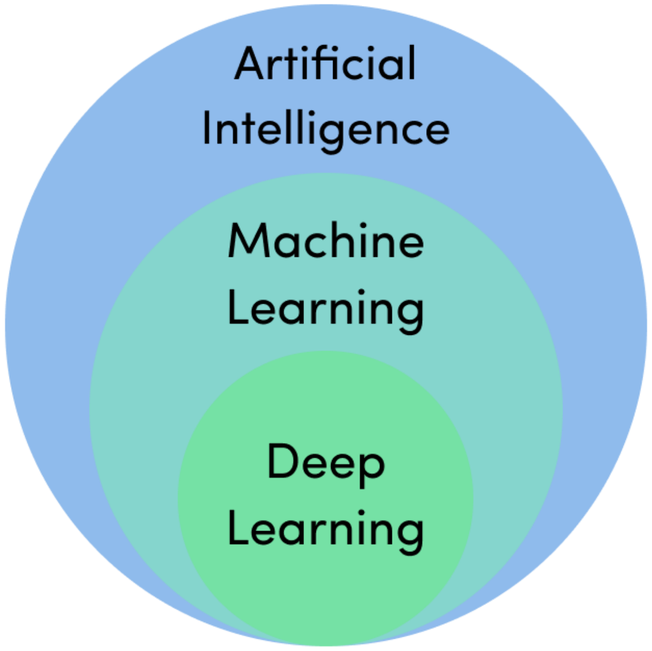
Beyond CS 181

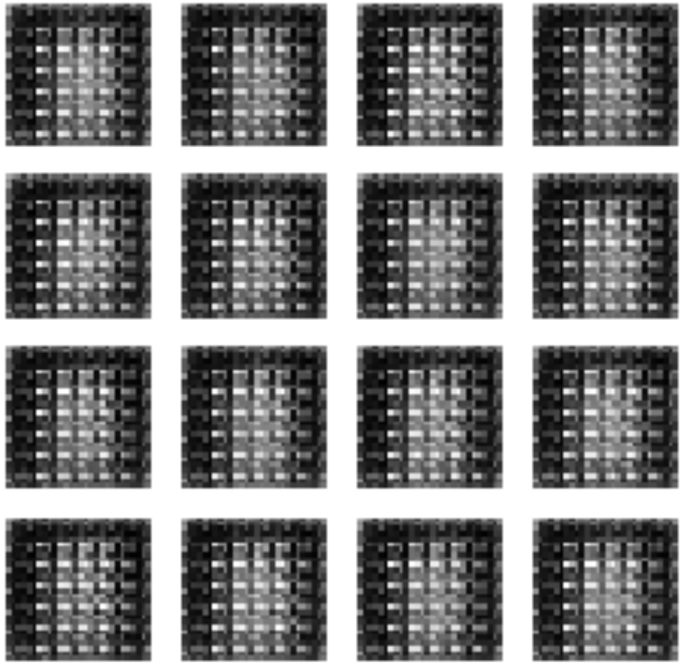
Matthew Nazari

HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Deep learning

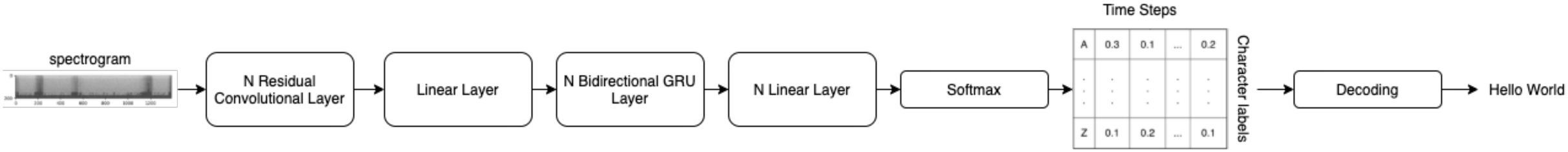
- Machine learning with artificial neural networks
- We will consider feed-forward and convolutional architectures





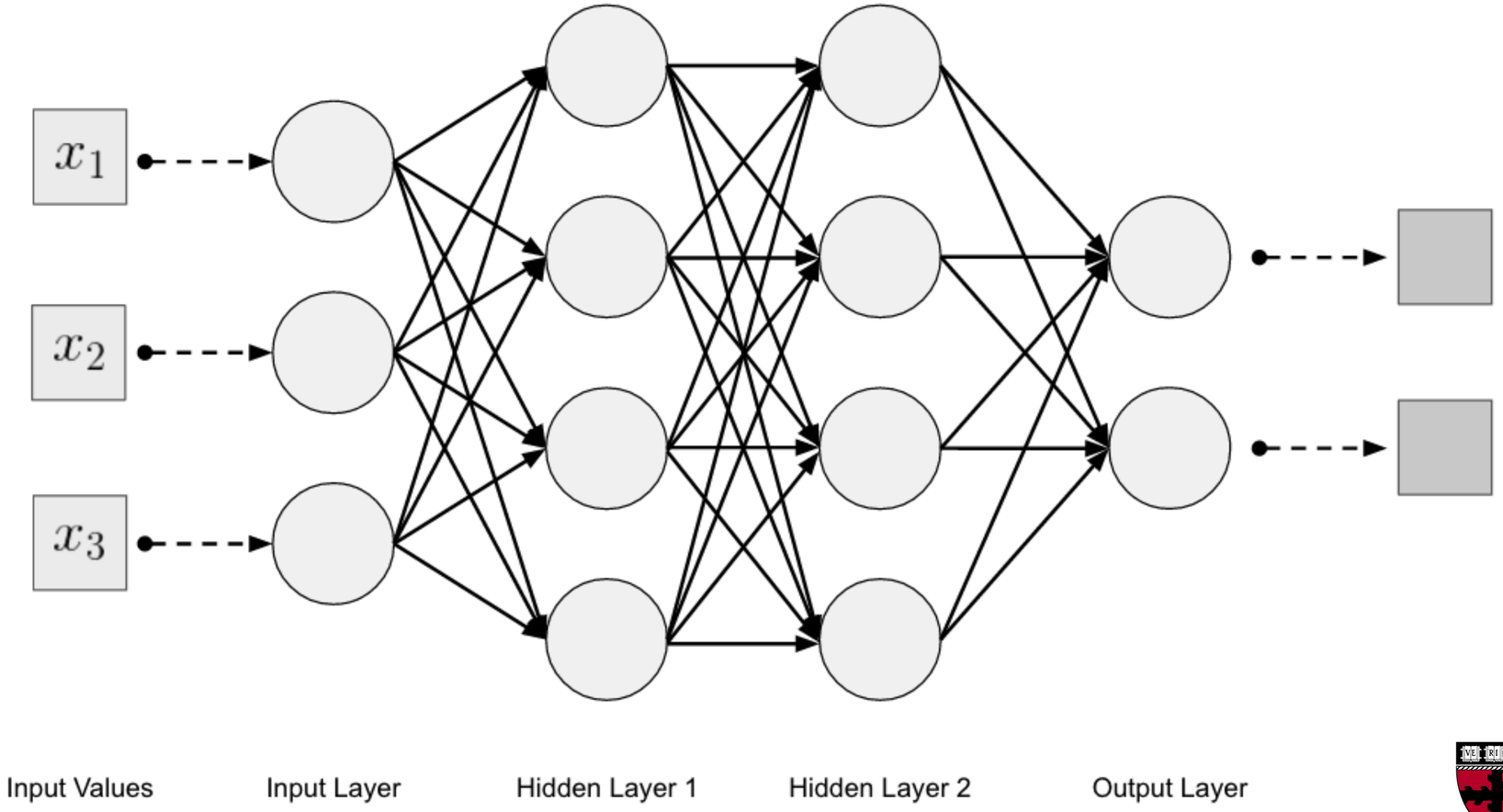
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Deep learning



HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Deep learning



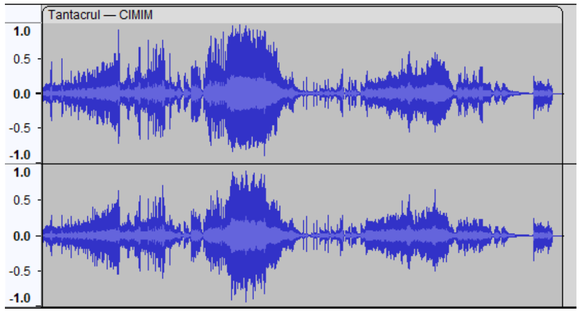
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Data

- A numerical representation of a real world ✨ thing ✨
- We pick how to represent it! We pick the features
- Hopefully these features are “useful”

The quick brown fox jumps over the lazy dog

Text



Sound

Images

HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Finding "useful representations"

- Some representations (or bases) are more helpful than others

- In linear regression, features $\mathbf{x}$ are useful if there exists a model in the family $f(\mathbf{x}; \mathbf{w}) := \mathbf{w}^\top \mathbf{x}$ that does well

Bad features: $\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$ Useful features! : $\begin{bmatrix} 1 \\ x_1 x_2 \end{bmatrix}$

Finding “useful representations”

- Domain experts can tell us which bases are useful

$$\phi(\mathbf{x}) := \begin{bmatrix} 1 \\ x_1 x_2 \end{bmatrix} \implies \phi(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{bmatrix}$$

- What if we… oh idk… use linear regression to find useful $\{\phi_i\}$!!!!!

Finding “useful representations”

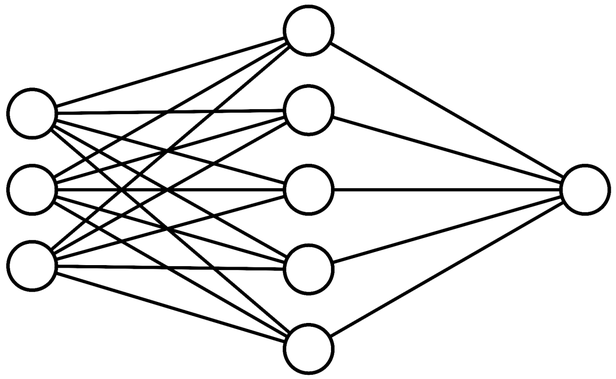
- Hence, we have a “adaptive basis”, once that we can toggle and change with by toggling or LEARNING $\{\mathbf{w}_i\}$

$$f(\mathbf{x}; \mathbf{w}, \{\mathbf{w}_i\}) := \mathbf{w}^\top \phi(\mathbf{x})$$

$$\phi_i = \mathbf{w}_i^\top \mathbf{x}$$

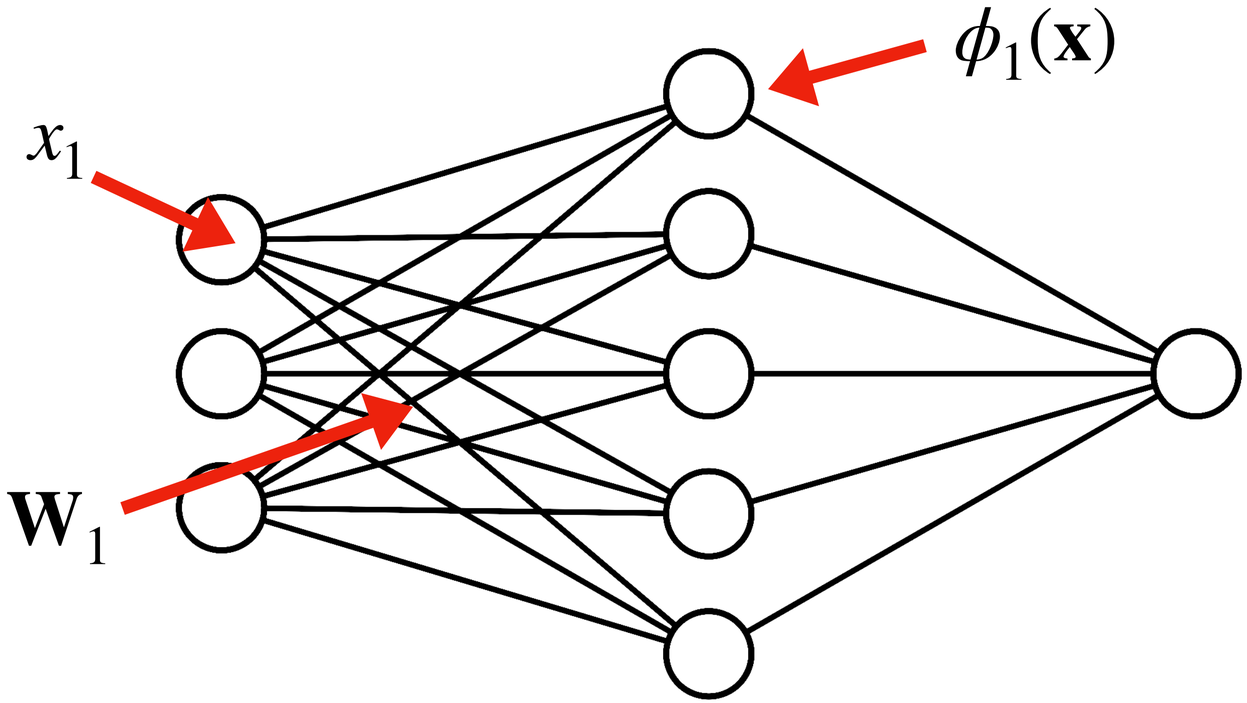
$$\mathbf{h} = \mathbf{W}_1 \mathbf{x}$$

$$y = \mathbf{w}_2 \mathbf{h}$$



HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Neural networks



$$\mathbf{h} = \mathbf{W}_1 \mathbf{x}$$

$$y = \mathbf{w}_2 \mathbf{h}$$

Input layer

$\mathbf{x} \in \mathbb{R}^D$

Hidden layer

$\mathbf{h} \in \mathbb{R}^J$

Output

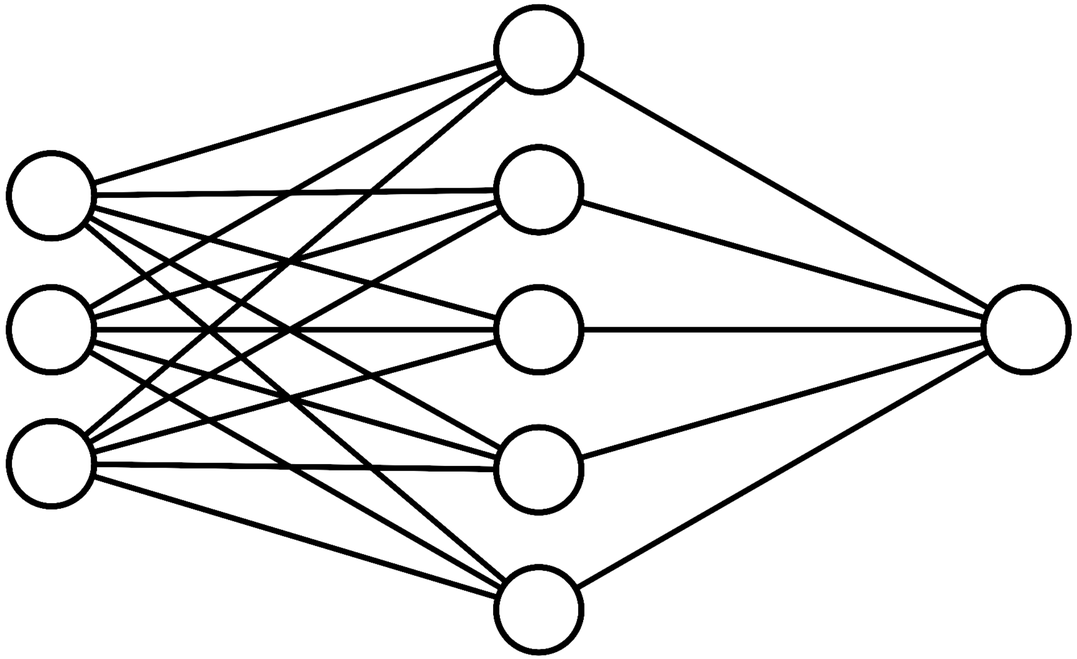
$y \in \mathbb{R}$

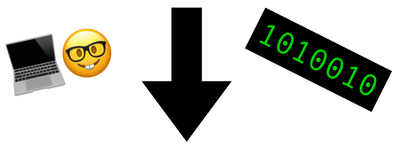
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Neural networks

$$\mathbf{h} = \mathbf{W}_1 \mathbf{x}$$

$$y = \mathbf{w}_2 \mathbf{h}$$






```python
from torch import nn  # Torch's neural network library

D, J = 3, 5  # Define our dimensions
net = nn.Sequential(
    nn.Linear(in_features=D, out_features=J),  # Hidden layer
    nn.Linear(in_features=J, out_features=1)   # Output layer
)
```

Input layer

$\mathbf{x} \in \mathbb{R}^D$

Hidden layer

$\mathbf{h} \in \mathbb{R}^J$

Output

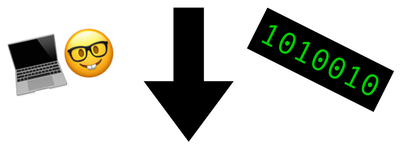
$y \in \mathbb{R}$

HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Neural networks

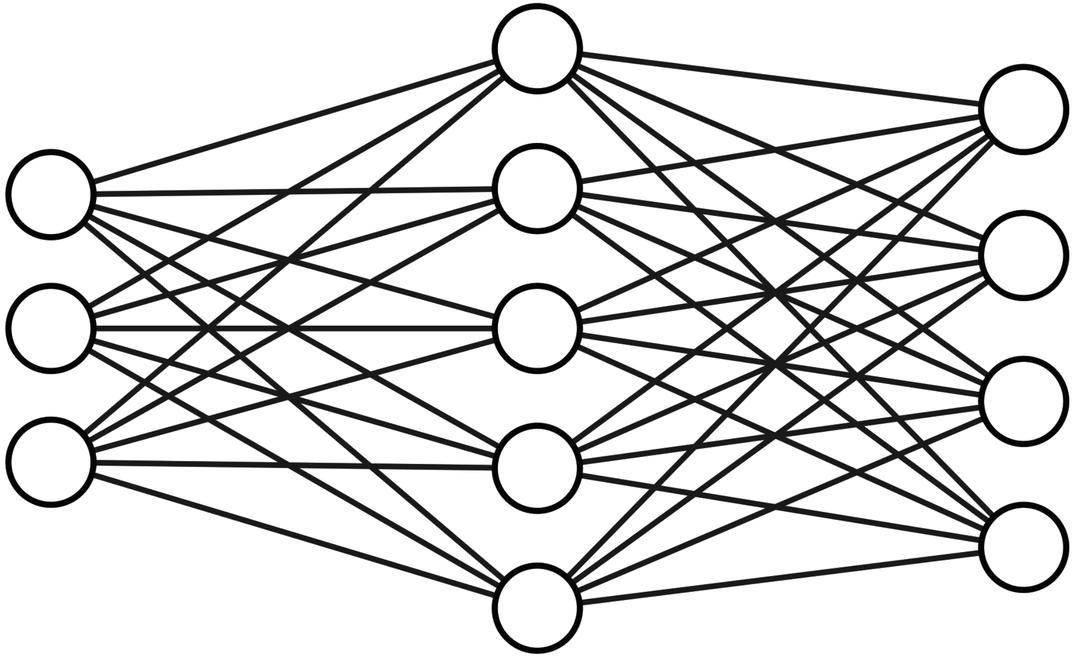
$$\mathbf{h} = \mathbf{W}_1 \mathbf{x}$$

$$\mathbf{y} = \mathbf{W}_2 \mathbf{h}$$




```
from torch import nn

D, J, K = 3, 5, 4
net = nn.Sequential(
    nn.Linear(D, J),
    nn.Linear(J, K)
)
```



Input layer

$\mathbf{x} \in \mathbb{R}^D$

Hidden layer

$\mathbf{h} \in \mathbb{R}^J$

Output

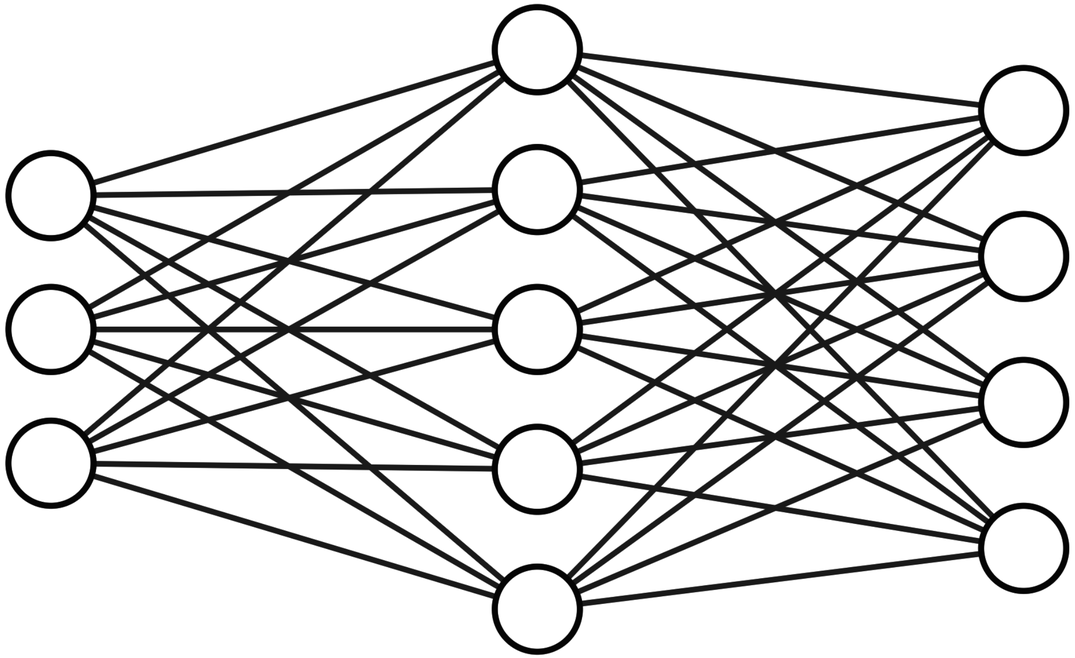
$\mathbf{y} \in \mathbb{R}^K$

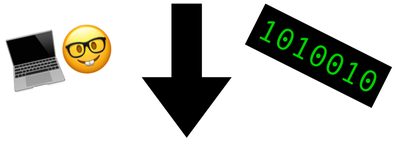
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Neural networks

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x})$$

$$\mathbf{y} = \mathbf{W}_2 \mathbf{h}$$






```python
from torch import nn

D, J, K = 3, 5, 4
net = nn.Sequential(
    nn.Linear(D, J),
    nn.ReLU(),
    nn.Linear(J, K)
)
```

Input layer

$\mathbf{x} \in \mathbb{R}^D$

Hidden layer

$\mathbf{h} \in \mathbb{R}^J$

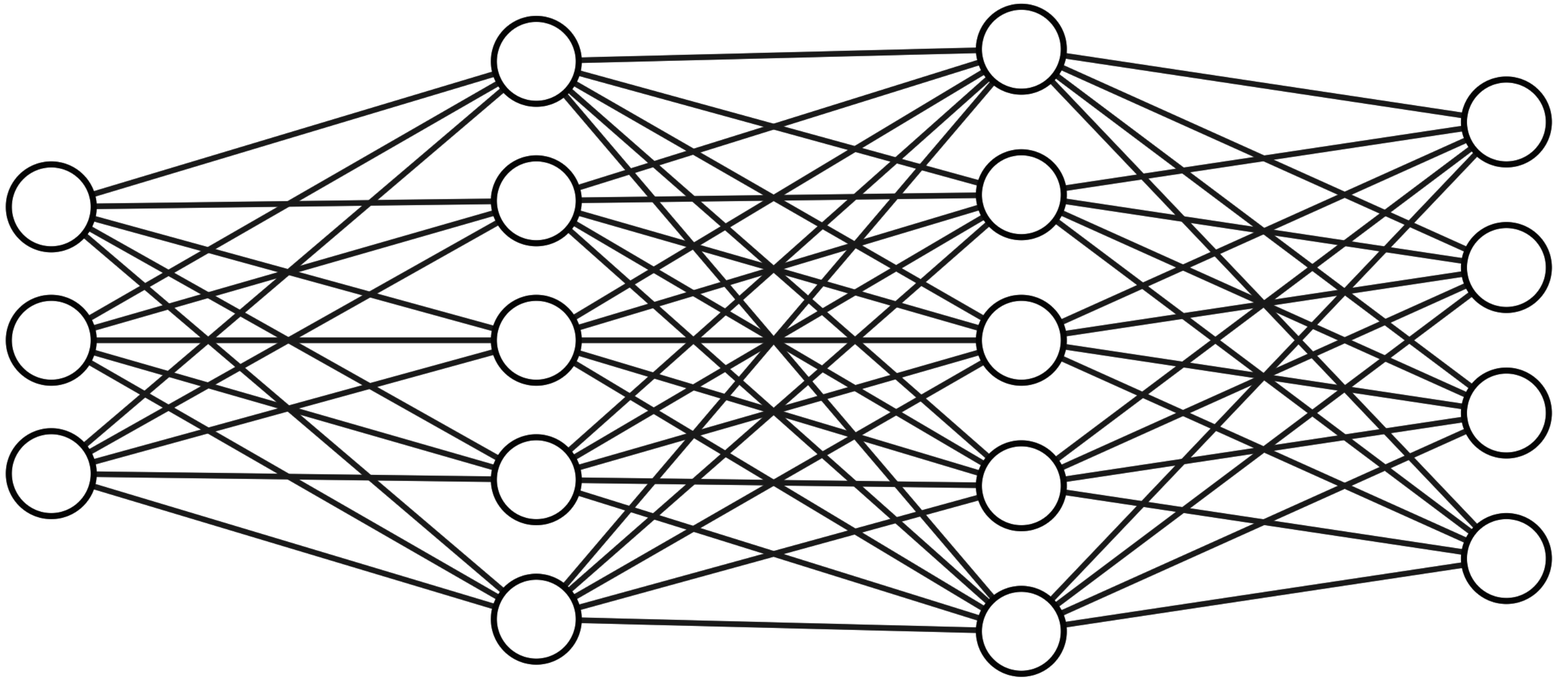
Output

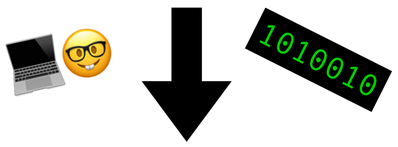
$\mathbf{y} \in \mathbb{R}^K$

HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Neural networks

$$\mathbf{h}_1 = \sigma_1(\mathbf{W}_1\,\mathbf{x})$$
$$\mathbf{h}_2 = \sigma_2(\mathbf{W}_2\,\mathbf{x})$$
$$\mathbf{y} = \mathbf{W}_3\,\mathbf{h}_2$$






```
from torch import nn

D, J, K = 3, 5, 4
net = nn.Sequential(
    nn.Linear(D, J),
    nn.ReLU(),
    nn.Linear(J, J),
    nn.ReLU(),
    nn.Linear(J, K)
)
```

Input layer	Hidden layer	Hidden layer	Output
$\mathbf{x} \in \mathbb{R}^D$	$\mathbf{h}_1 \in \mathbb{R}^J$	$\mathbf{h}_2 \in \mathbb{R}^J$	$\mathbf{y} \in \mathbb{R}^K$

HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Neural networks

- There's an issue… we can't pick up on SPATIAL CORRELATION!

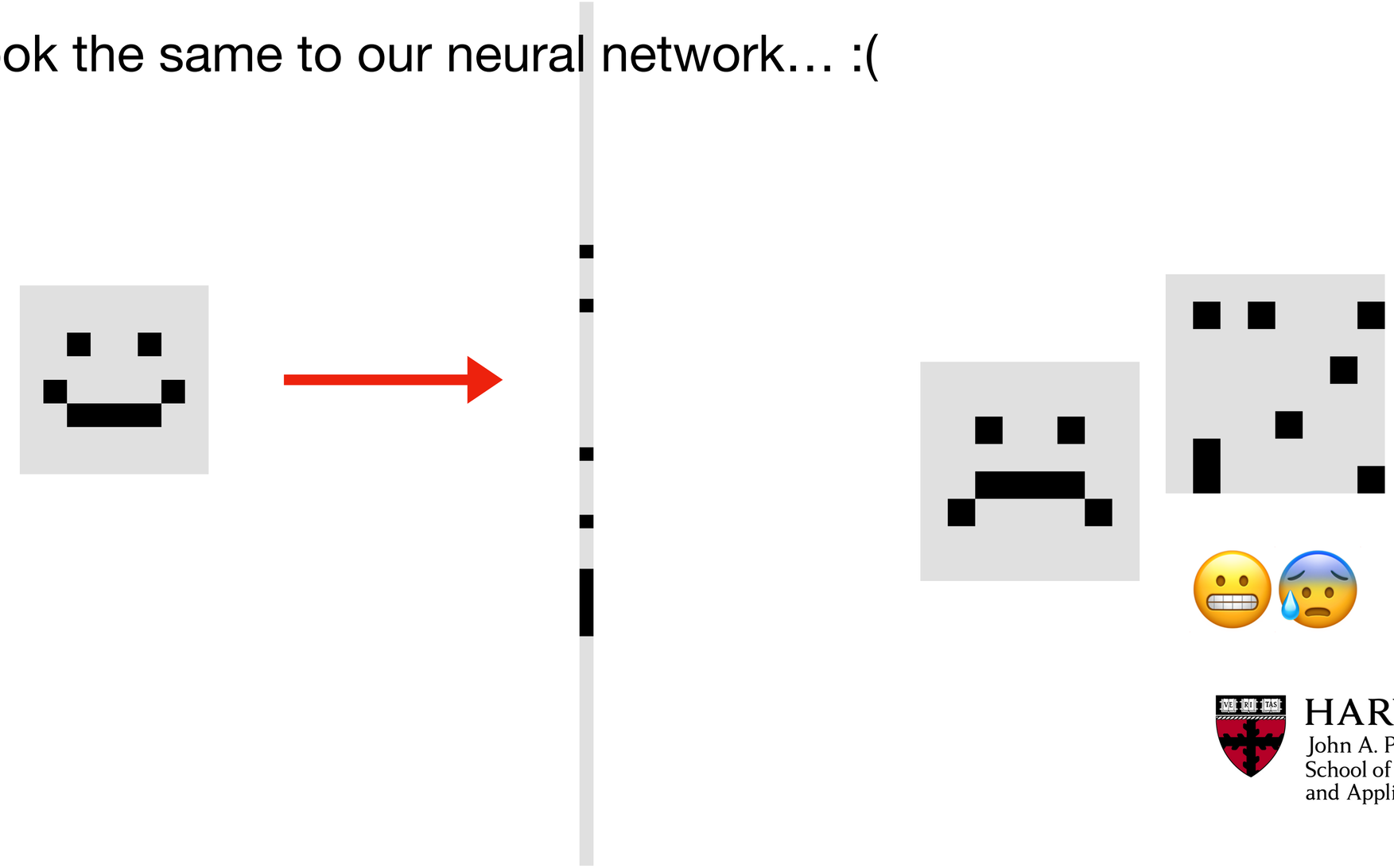
$$\phi_i(\mathbf{x}) = \sigma(\mathbf{w}_i^\top \mathbf{x})$$

- (There's also SO many parameters….)

HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Spatial correlation

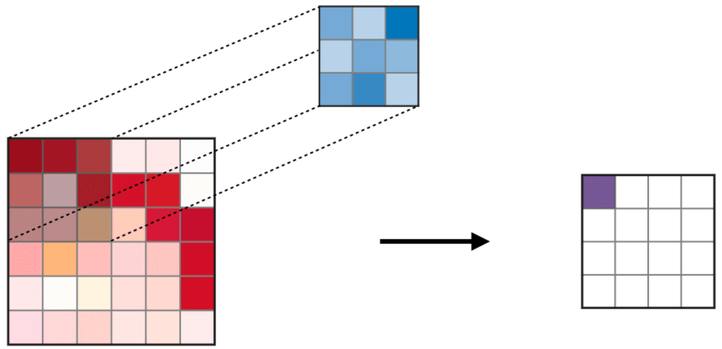
- These all look the same to our neural network… :(



HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Convolutional Neural Networks (CNNs)

- Neural networks that exploit spatial locality by use of convolutional layers
- We are still trying to find a useful representation, i.e. column vector $\phi(\mathbf{x})$
- Now, though, each feature $\phi_i(\mathbf{x})$ we discover are functions over not all of $\mathbf{x}$ but regions



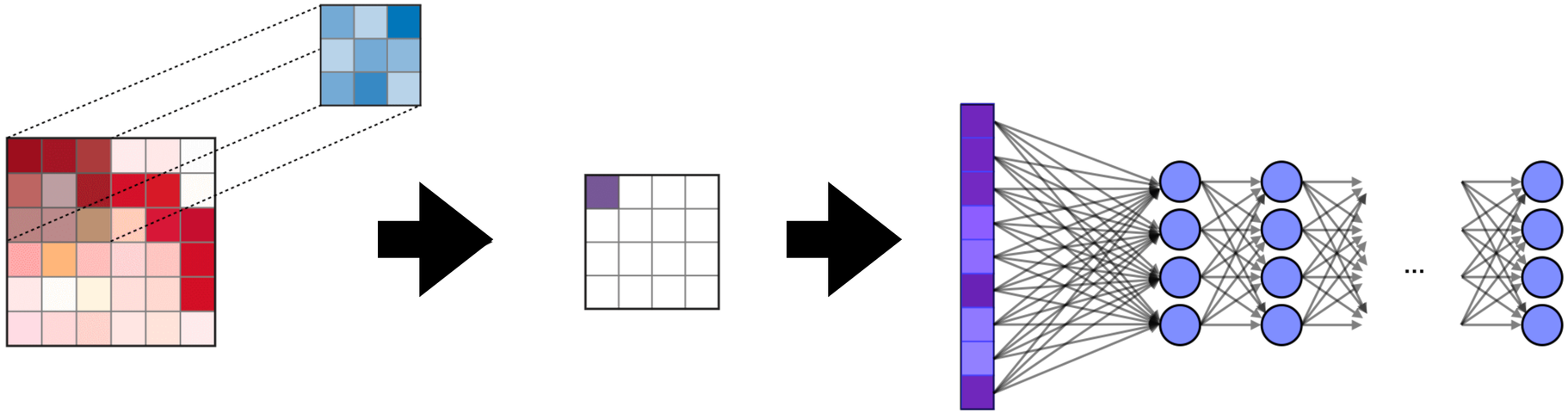
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Credit for all images from here on out:
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

Convolutional Neural Networks (CNNs)

- Each filter has the theoretical capacity to detect any feature, like “ear” or “nose” (those are too abstract, really)

- The final features in purple take into account spatial information in an image

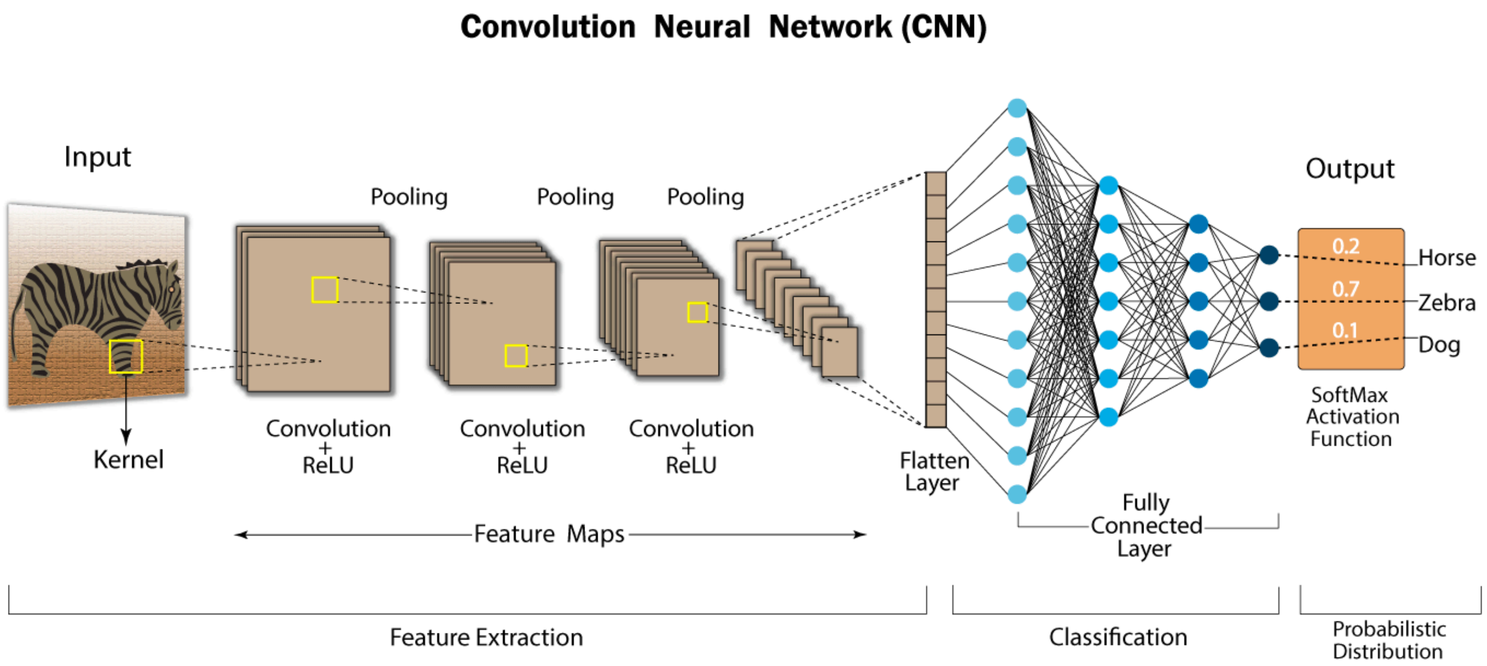


HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Credit for all images from here on out:
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

Convolutional Architecture

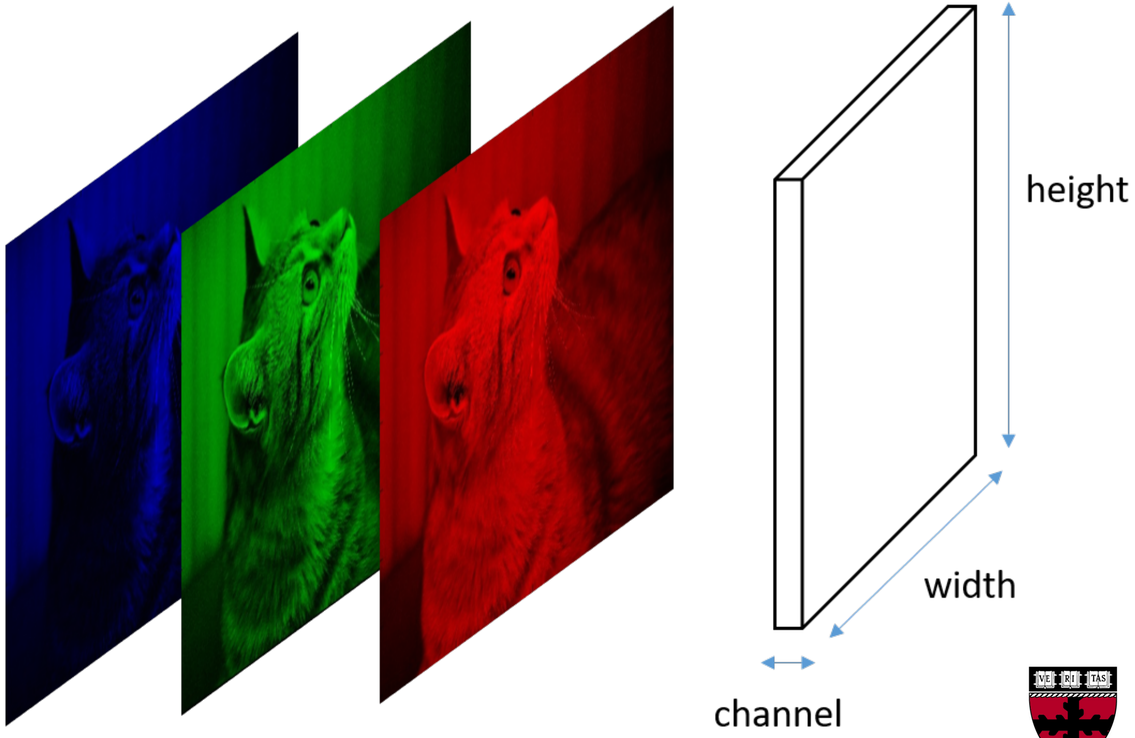
- Channels
- Filters
- Padding and strides
- Pooling
- Data augmentation



HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Channels

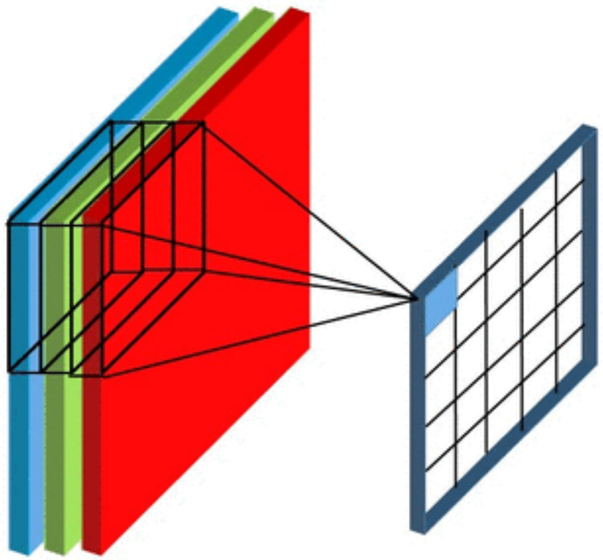
- Each raw datapoint is an image, which translates to a tensor! $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$
- $C =$ Channels (like colors)
- $H =$ Height
- $W =$ Width



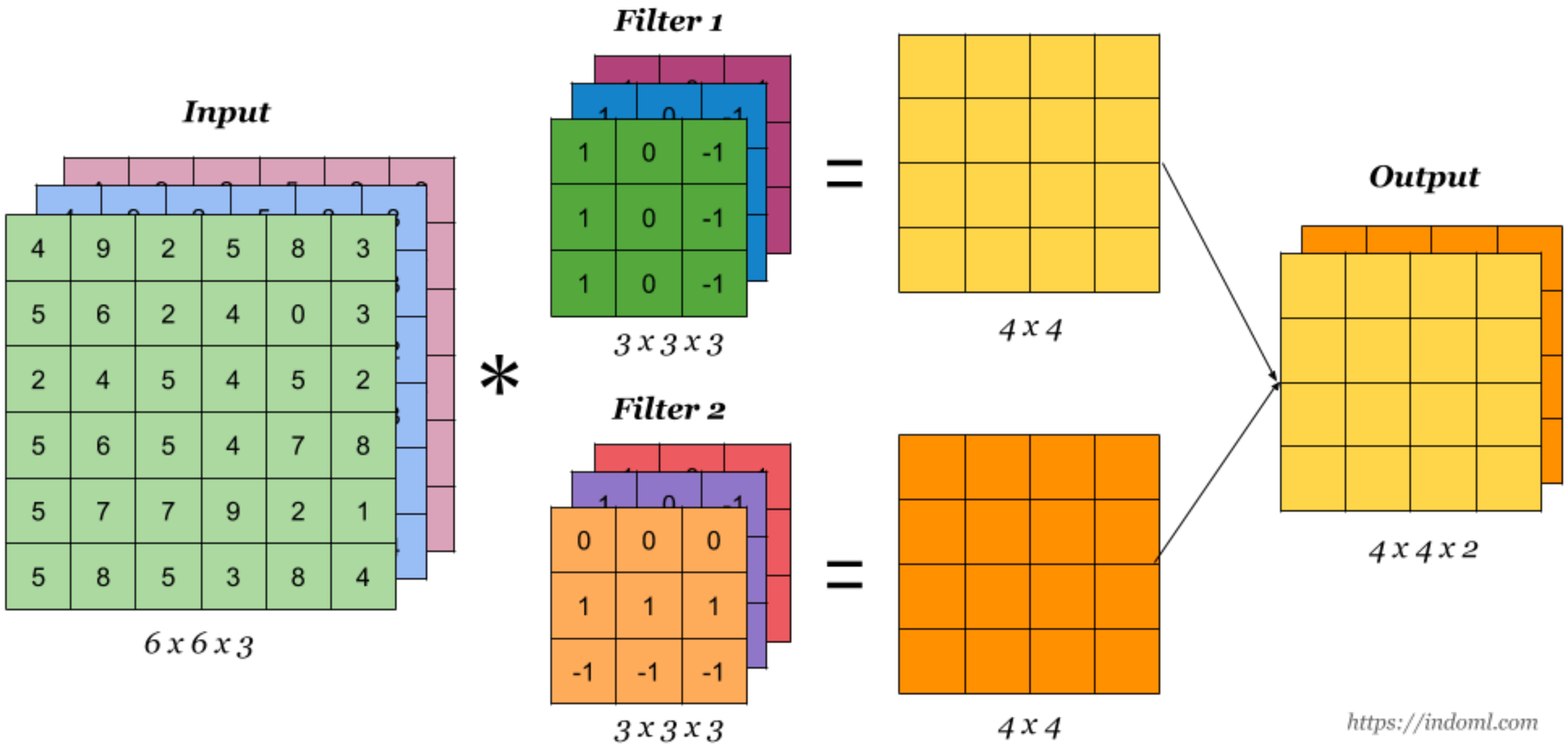
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Filters

- Usually square windows that stretch across all channels
- More filters than there are channels

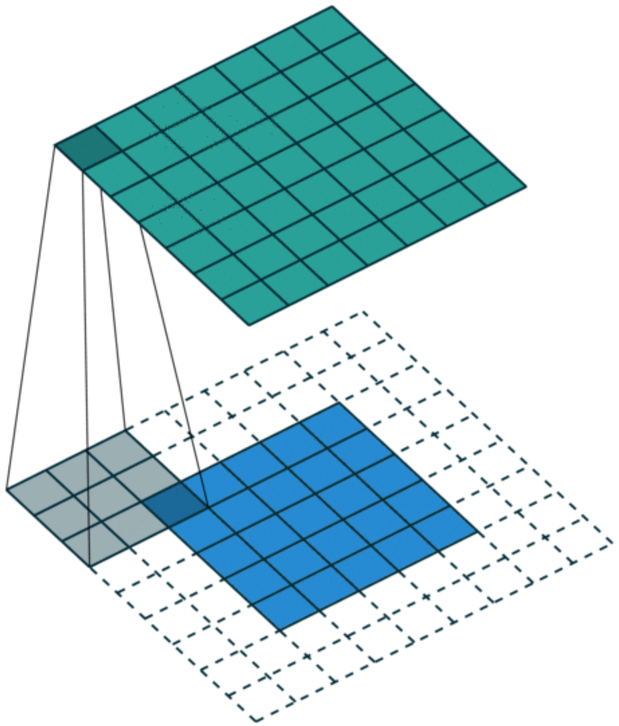


Filters


Input
4 9 2 5 8 3
5 6 2 4 0 3
2 4 5 4 5 2
5 6 5 4 7 8
5 7 7 9 2 1
5 8 5 3 8 4
6 x 6 x 3
*
Filter 1
1 0 -1
1 0 -1
1 0 -1
3 x 3 x 3
=
4 x 4
Filter 2
0 0 0
1 1 1
-1 -1 -1
3 x 3 x 3
=
4 x 4
Output
4 x 4 x 2
https://indoml.com

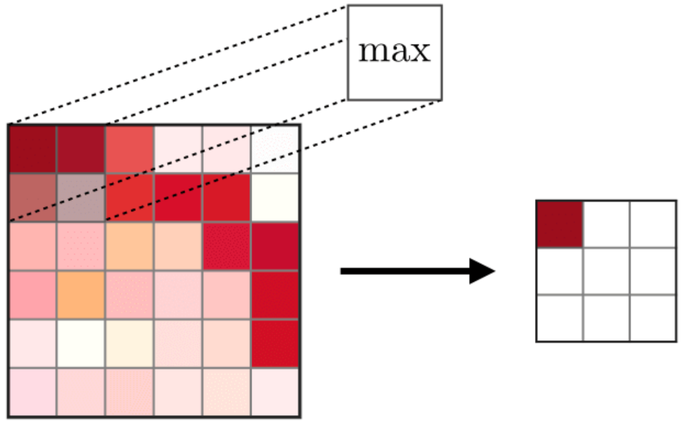
Padding and Stride

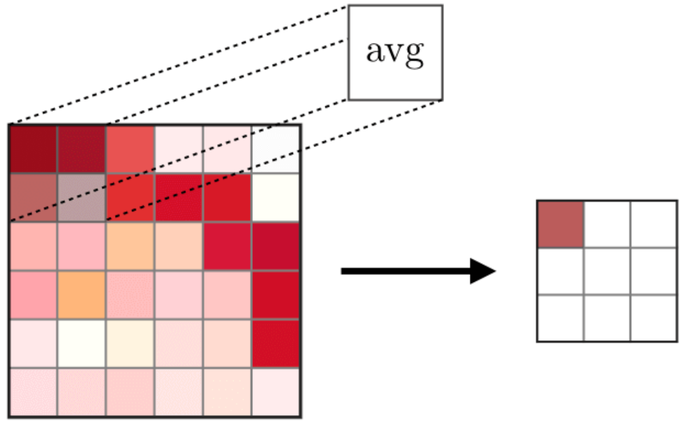
- These determine the pattern the filter slides over the image



Pooling

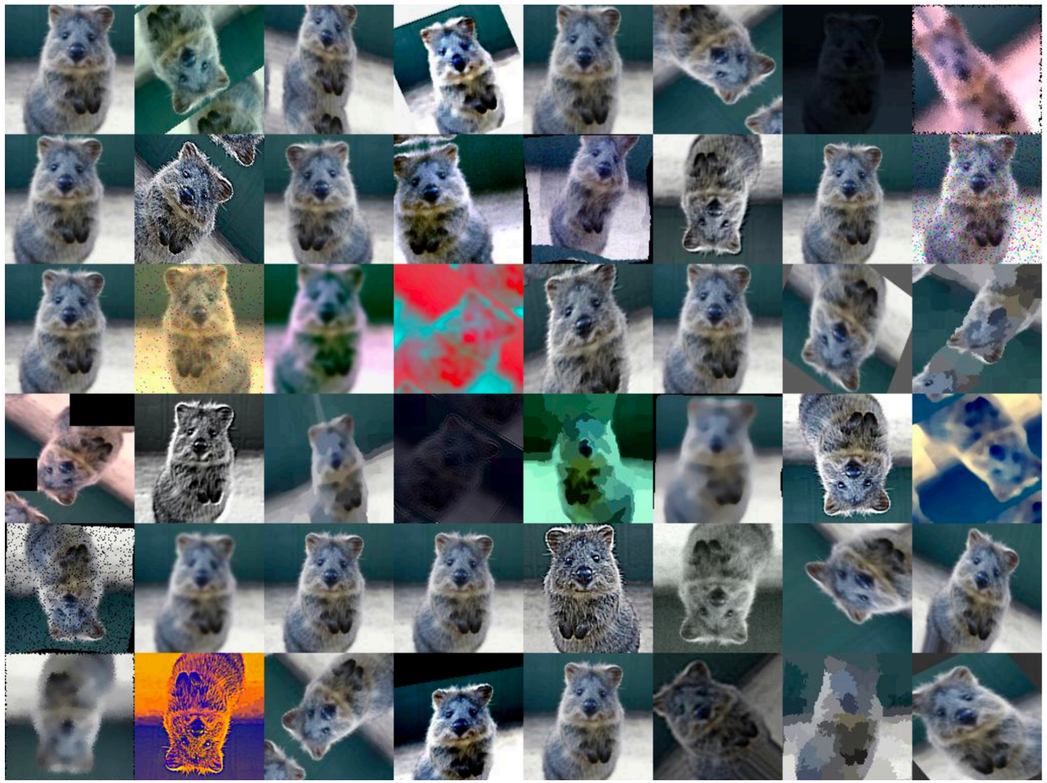
- Reduce the dimensionality
- Preserve detected features (Max) or downsize feature map (Avg)





Data Augmentation

- Literally just perturb the image
- Adds robustness!



Implementing in PyTorch