

A decorative network diagram in the top-left corner of the slide. It consists of a complex web of interconnected nodes and edges. The nodes are represented by small circles, some of which are solid blue, some are solid grey, and some are hollow with a blue outline. The edges are thin grey lines connecting the nodes. The overall shape is roughly triangular, pointing towards the top-left corner.

BST 261: Data Science II

Lecture 4

Introduction to Convolutional Neural Networks (CNNs)

Santiago Romero-Brufau
Harvard T.H. Chan School of Public Health
Spring 2

A decorative network diagram in the bottom-right corner of the slide. It is similar to the one in the top-left, featuring a web of interconnected nodes and edges. The nodes are small circles, some solid blue, some solid grey, and some hollow with a blue outline. The edges are thin grey lines. The shape is roughly triangular, pointing towards the bottom-right corner.

Leftovers

- When we use regularization, we need to standardize inputs
- On PSet 1, question 2, we ask about “test set accuracy”, we use accuracy to mean “a measure of how good the model is”.
- Make sure you have signed up to:
 - Present a paper (make sure the signup sheet says 2023)
 - What final assignment you want (Kaggle competition or project proposal)
- Video of Lab1 (including Q2) is available on Canvas

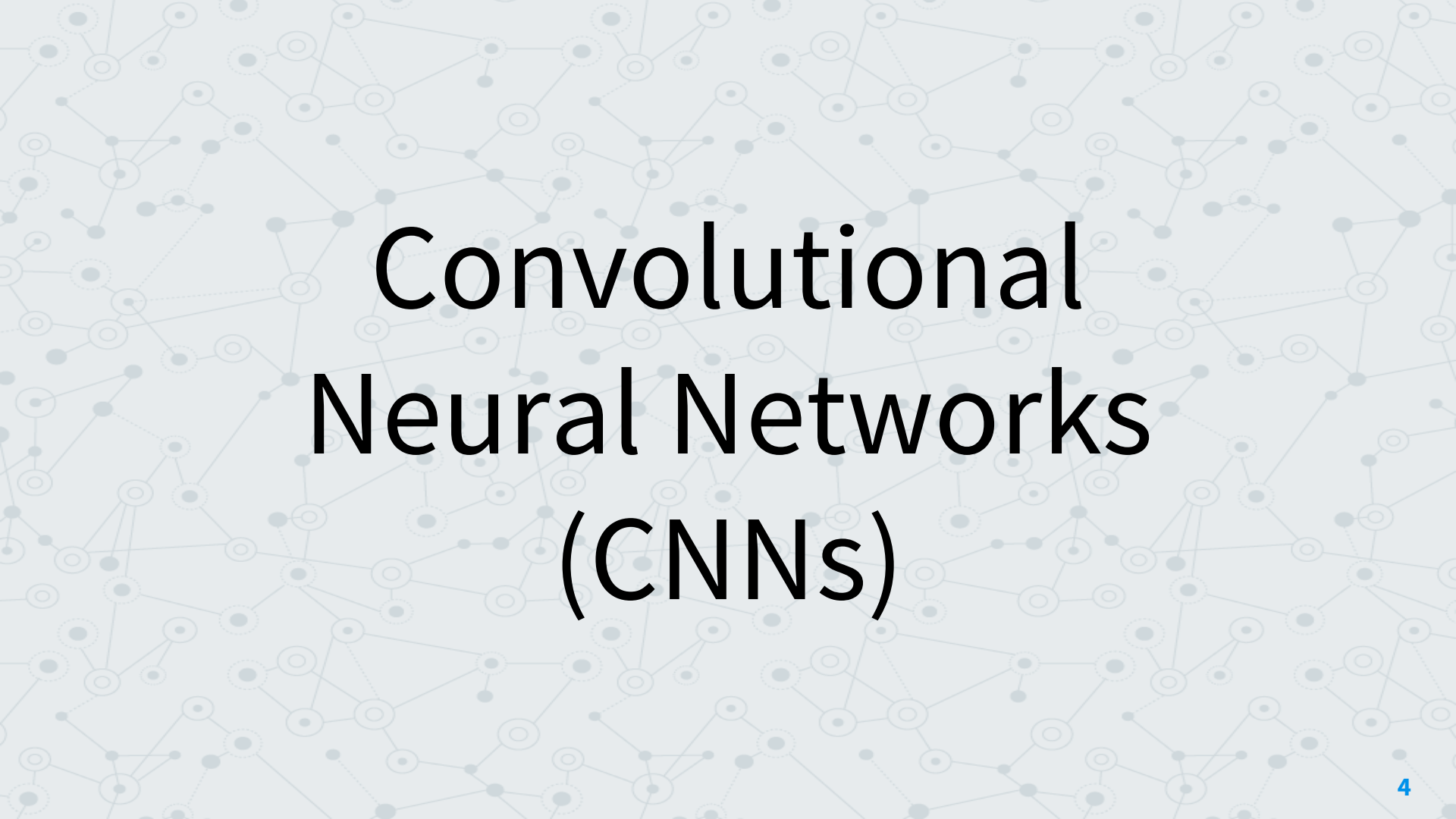


“

*“Never, ever underestimate the
importance of having fun.”*

Randy Pausch

<https://www.youtube.com/watch?v=ncoSRKoU6GQ>

The background of the slide is a light blue-grey color with a complex, repeating pattern of interconnected nodes and lines, resembling a neural network or a molecular structure. The nodes are small circles, some solid and some hollow, connected by thin lines.

Convolutional Neural Networks (CNNs)

CNNs

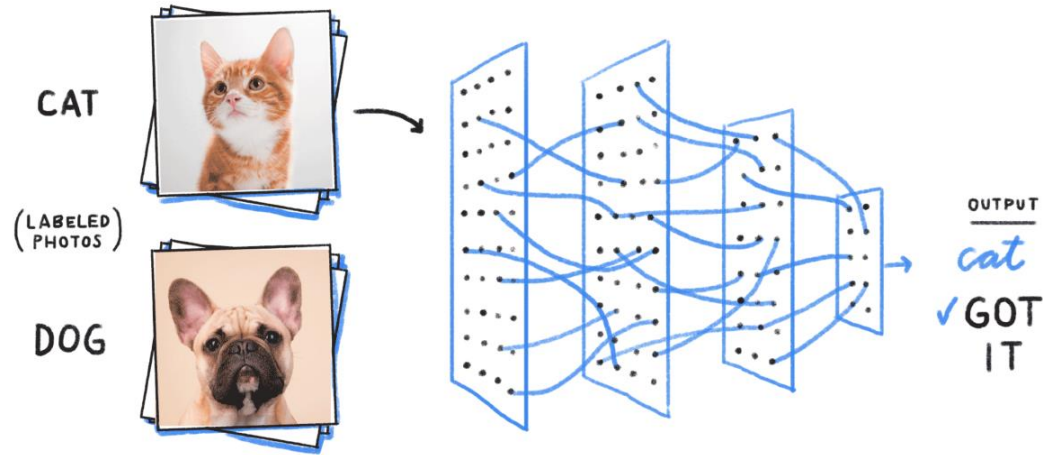
Convolutional neural networks = CNNs = convnets = (many times) computer vision

CNNs are at the heart of deep learning, emerging in recent years as the most prominent strain of neural networks in research. They have revolutionized computer vision, achieving state-of-the-art results in many fundamental tasks, as well as making strong progress in natural language processing, reinforcement learning, and many other areas. CNNs have been widely deployed by tech companies for many of the new services and features we see today. They have numerous and diverse applications, including:

- ⦿ Detecting and labeling objects, locations, and people in images
- ⦿ Converting speech into text and synthesizing audio of natural sounds
- ⦿ Describing images and videos with natural language
- ⦿ Tracking roads and navigating around obstacles in autonomous vehicles
- ⦿ Analyzing video game screens to guide autonomous agents playing them

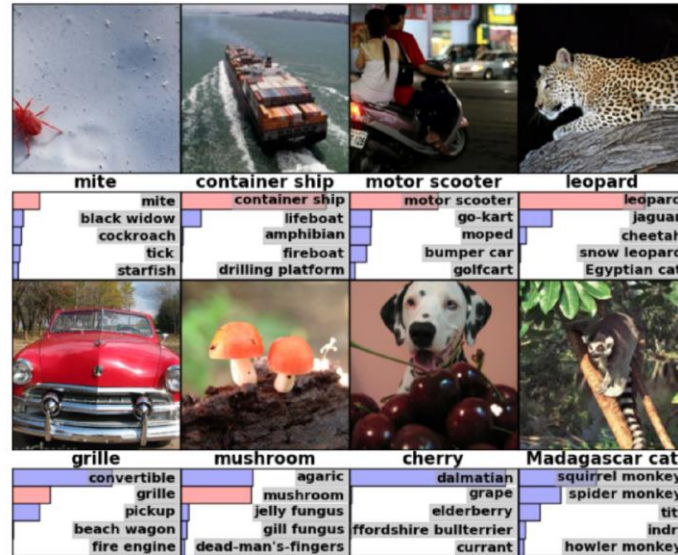
CNN Applications

Image classification



CNN Applications

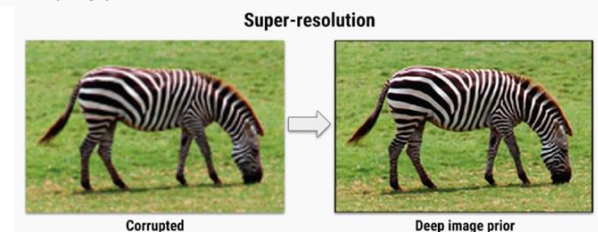
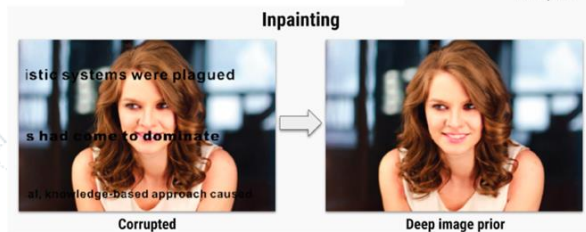
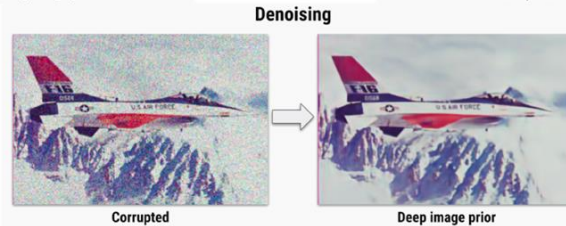
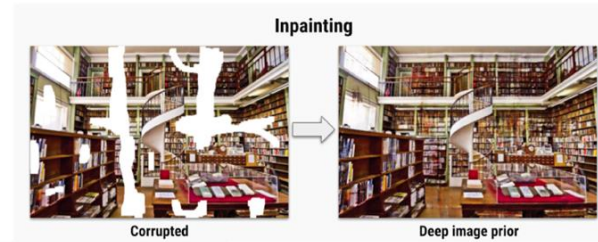
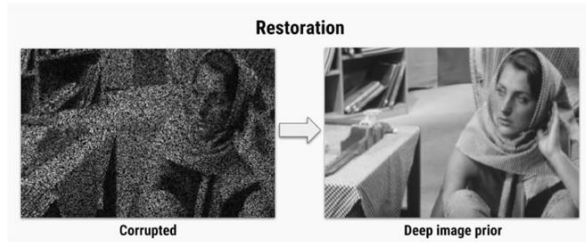
Computer vision



Top: 4 correctly classified examples. Bottom: 4 incorrectly classified examples. Each example has an image, followed by its label, followed by the top 5 guesses with probabilities. From Krizhevsky *et al.* (2012).

CNN Applications

Restoration, inpainting, denoising and super-resolution

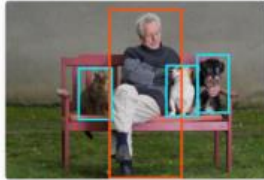


CNN Applications

PERSON, CAT, DOG



(A) Classification



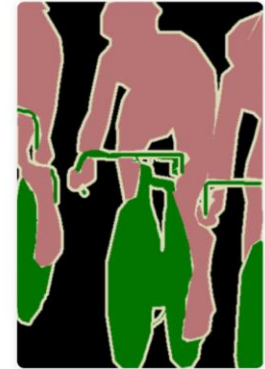
(B) Detection



(C) Segmentation



PREDICT
→



● Person ● Bicycle ● Background

<https://missinglink.ai/guides/computer-vision/image-segmentation-deep-learning-methods-applications/>

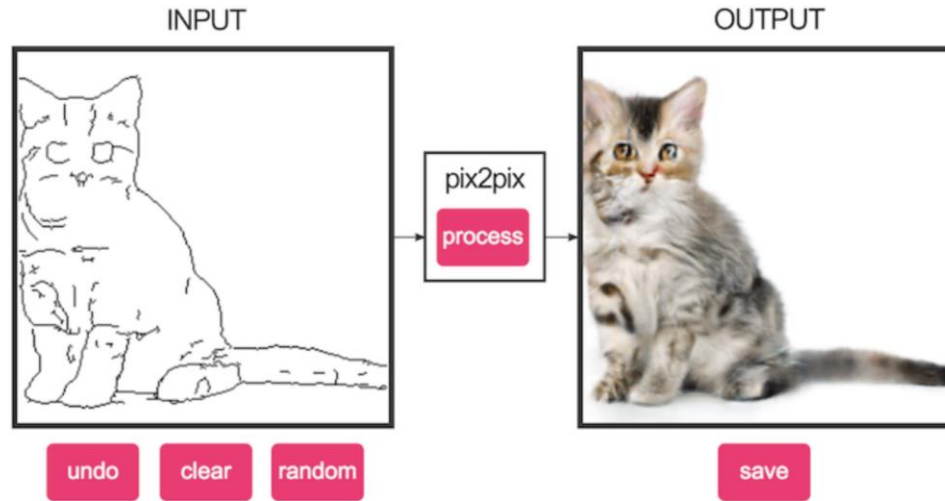
CNN Applications

Self-driving cars



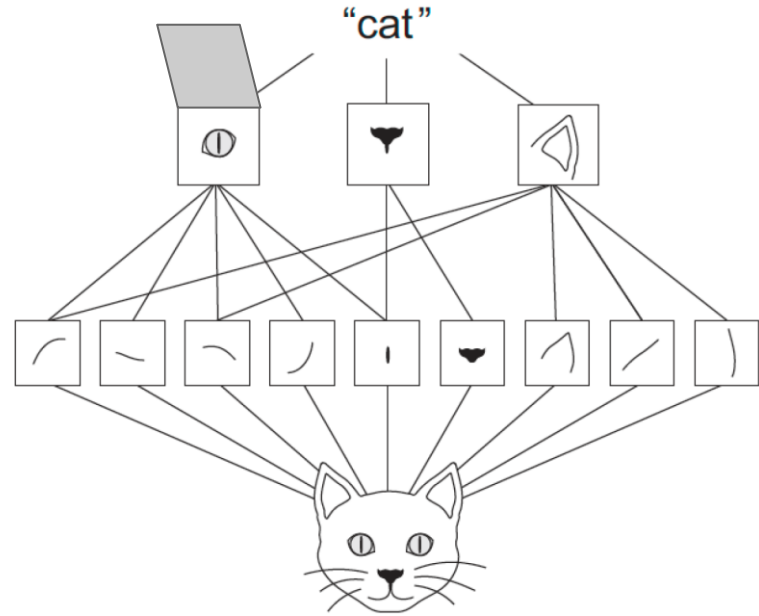
CNN Applications

Generating realistic pictures from drawn edges



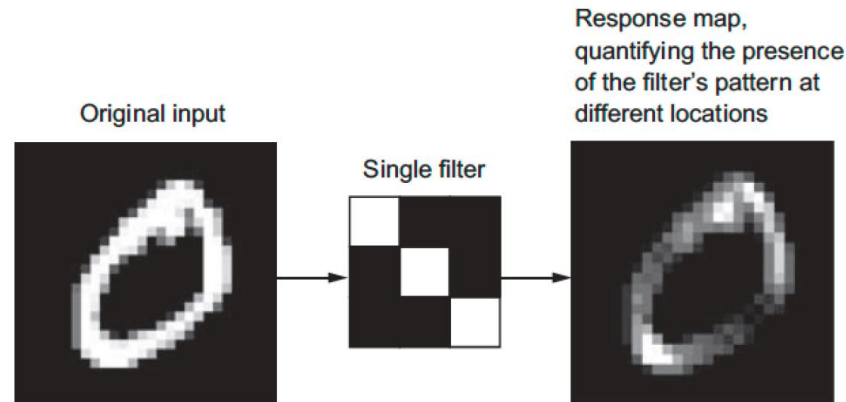
Dense vs Convolutional Layers

- ◎ Dense layers learn **global** patterns
- ◎ Convolutional layers learn **local** patterns
- ◎ CNNs learn **spatial hierarchies of patterns**: one convolutional layer will learn small patterns and the next larger patterns made of the features of the layer before, and so on

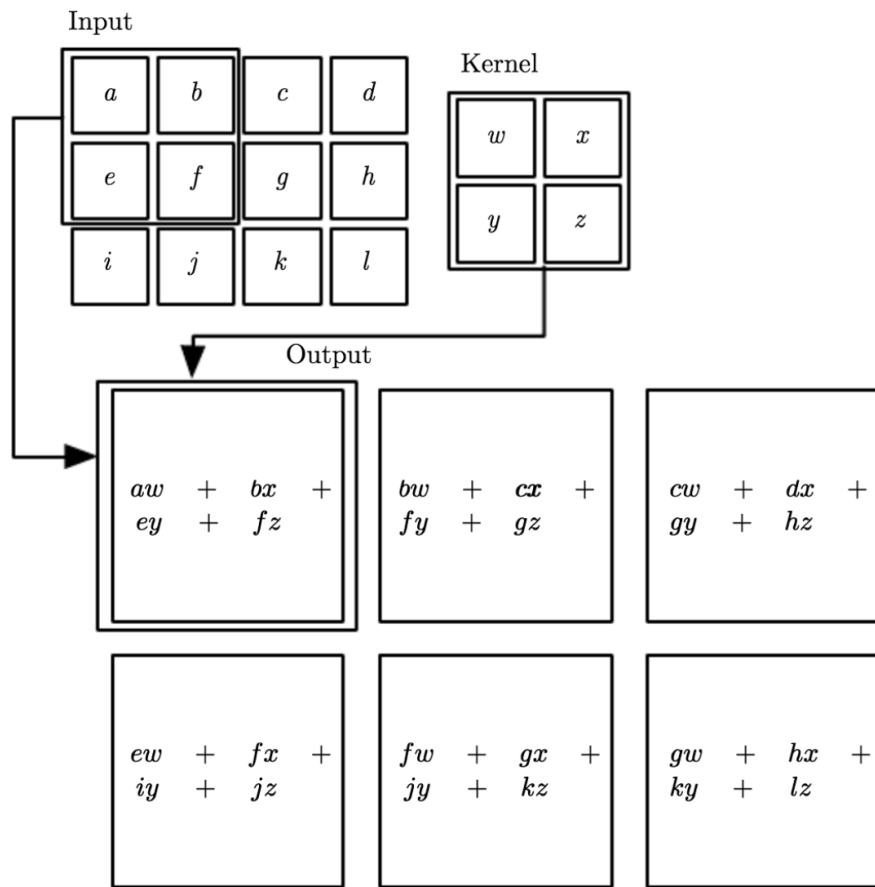


Convolution Layers

- ◎ The primary purpose of the convolution operation is to extract features from the input
- ◎ To do this, the input is split into several different areas, and the convolution operation applied to each area
- ◎ A summary value is then calculated and kept as part of the output
- ◎ We need the input image (sometimes called **feature map**), and a two-dimensional **kernel**, or more commonly, **filter**, which is a weighting function
- ◎ The output resulting from this operation is called the **response map**



The Convolution Operation



The Convolution Operation

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

The Convolution Operation

The diagram illustrates the convolution operation. On the left, a 5x5 input image is shown with a 3x3 green kernel highlighted. The kernel is applied to the input image, and the resulting 3x3 convolved feature map is shown on the right. The input image is a 5x5 grid of numbers. The kernel is a 3x3 grid of numbers. The convolved feature map is a 3x3 grid of numbers. The input image is labeled 'Image' and the convolved feature map is labeled 'Convolved Feature'.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	1	1
0	1	1
0	1	1

4	3	4
2	4	3
2	3	4

Image

Convolved Feature

Filters

- ◎ The most common filter sizes are 3×3 and 5×5
- ◎ Sometimes 7×7 filters are also used
- ◎ A 1×1 filter is a special case that we will see later in the course when talking about RNNs
- ◎ Odd dimension filters are preferred for computer vision
 - Provide natural padding (we'll see this in the following slides)
 - Ensures a central position or pixel of the filter

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

6x6

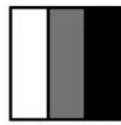


*

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

3x3

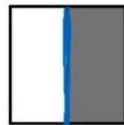
*



Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	<u>10</u>	<u>10</u>	<u>0</u>	0	0
10	<u>10</u>	<u>10</u>	<u>0</u>	0	0
10	<u>10</u>	<u>10</u>	<u>0</u>	0	0

6x6



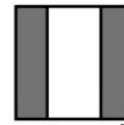
*

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

3x3

=

0	30	30	0
0	30	30	0
0	30	30	0
0	<u>30</u>	<u>30</u>	0

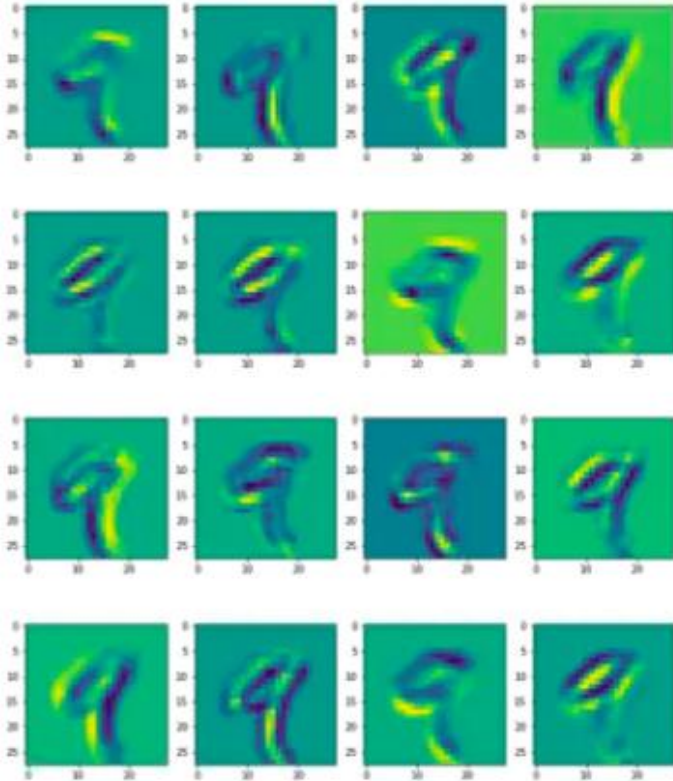


*

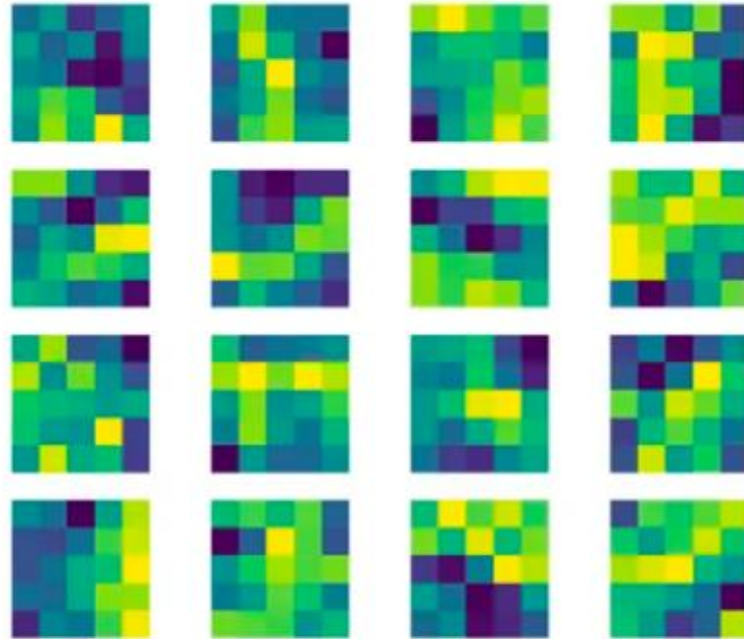


Andrew Ng

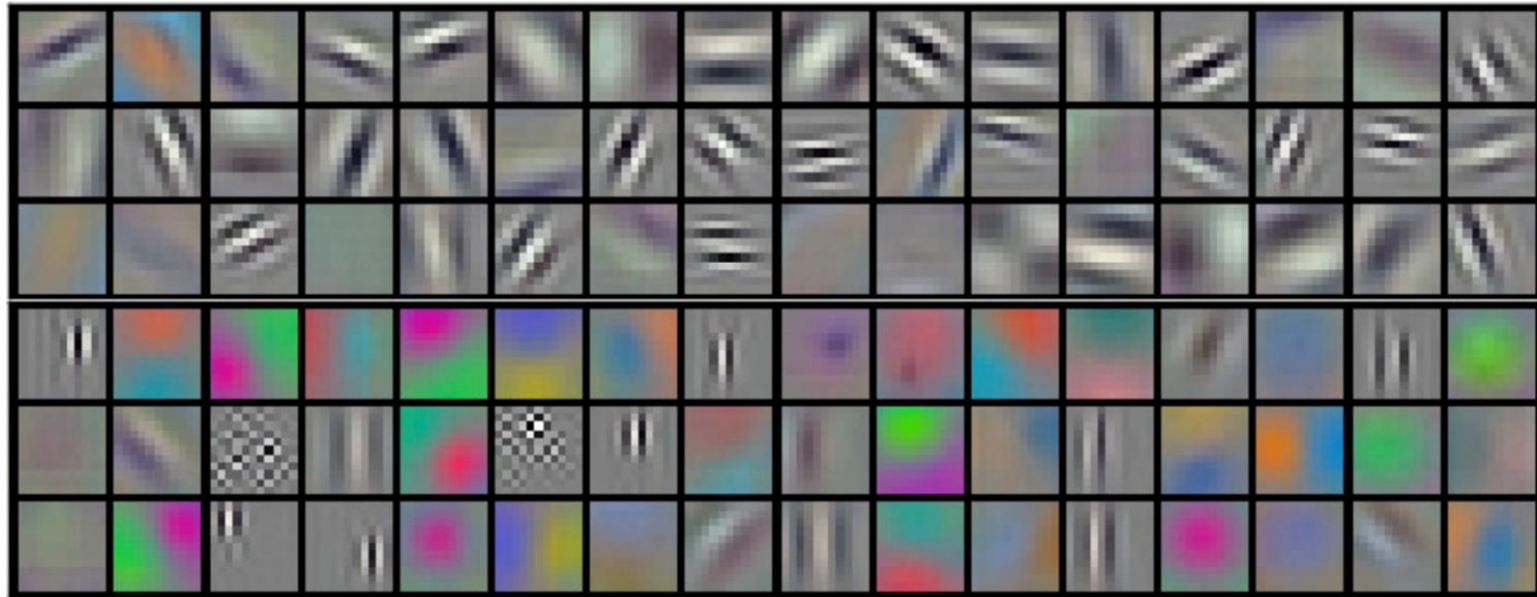
Feature map



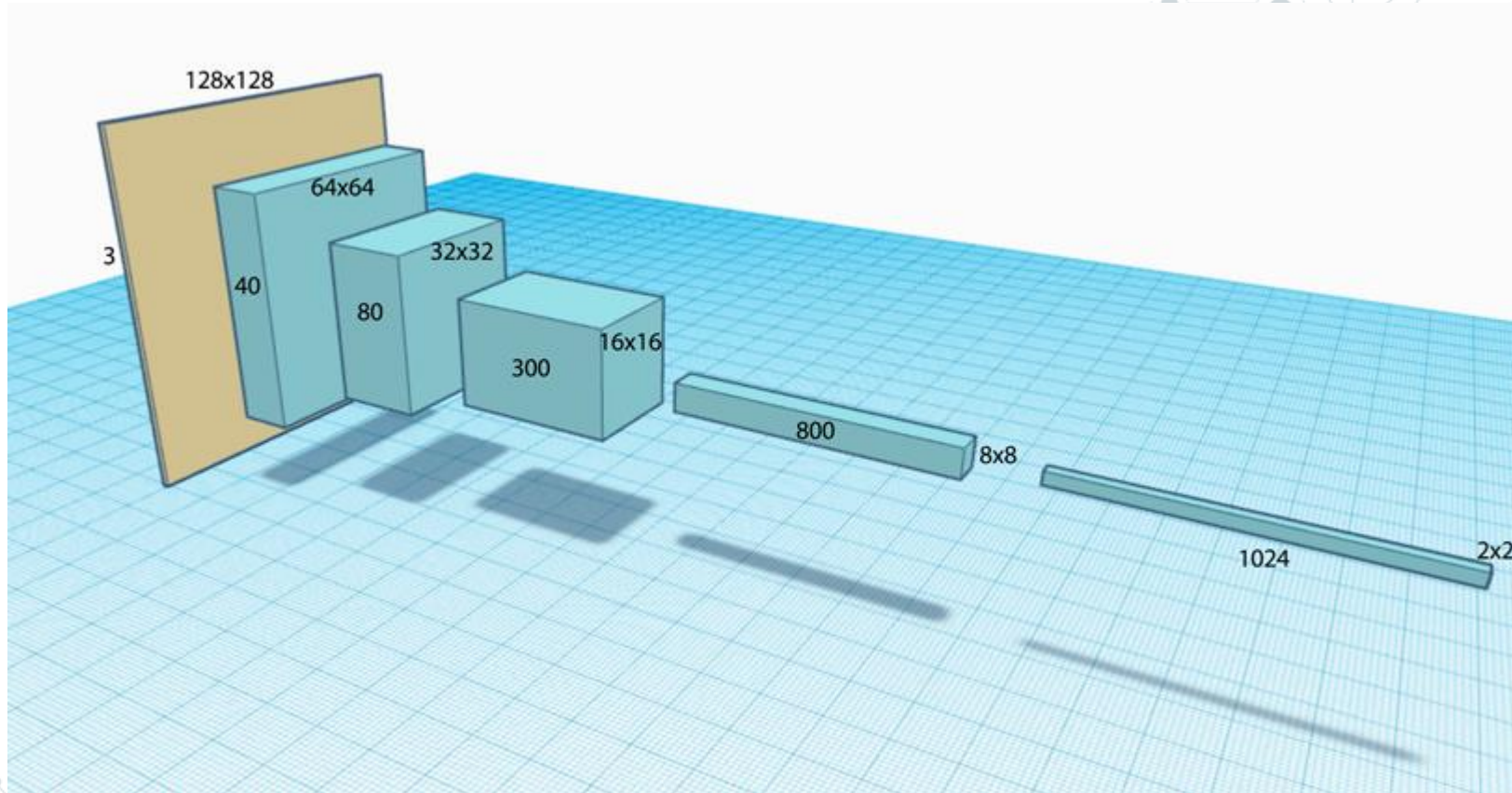
Filters



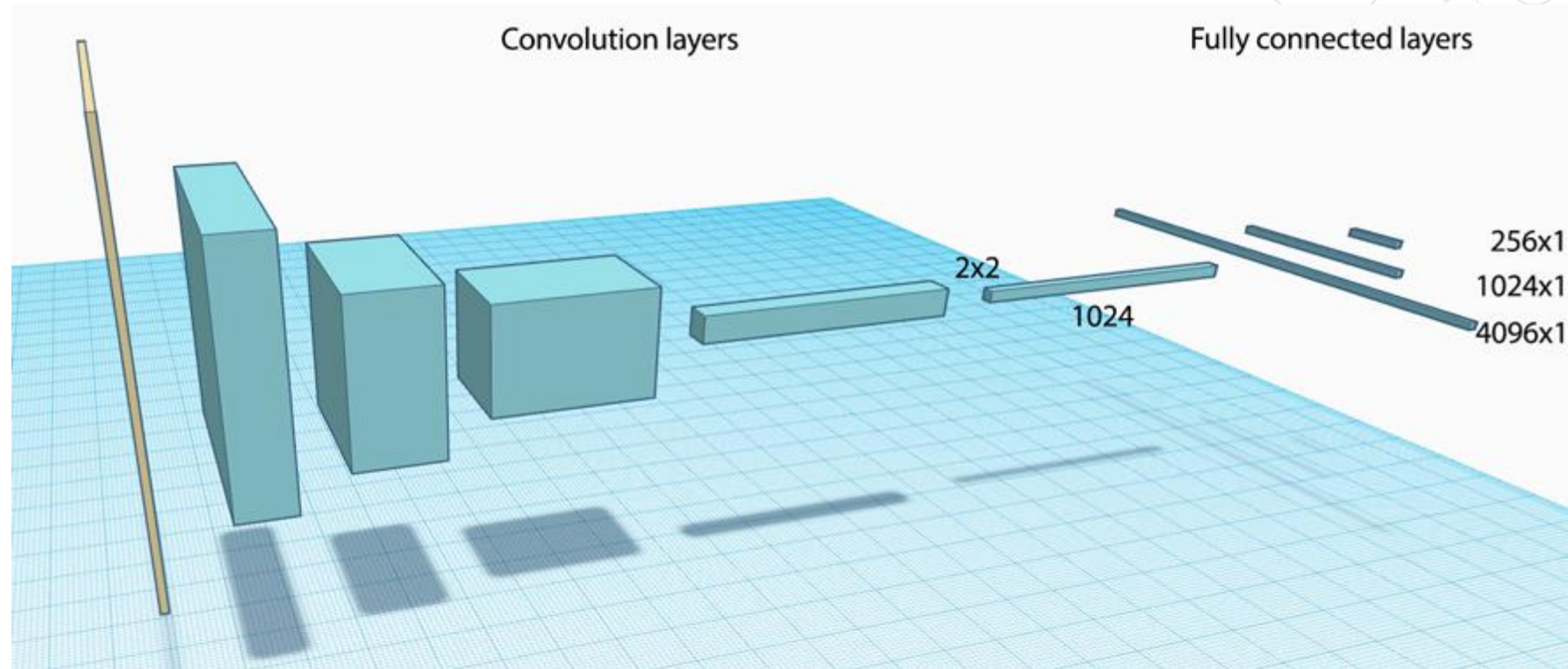
*Brighter colors mean higher numbers



Filters from the first convolution layer of AlexNet. Krizhevsky et al.



Visualization of CNN. Jonathan Hui Blog. <https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/>



Visualization of CNN. Jonathan Hui Blog. <https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/>

Visualizing what the filters are doing

<https://www.youtube.com/watch?v=f0t-OCG79-U>

(Israel Vicars)

Convolution

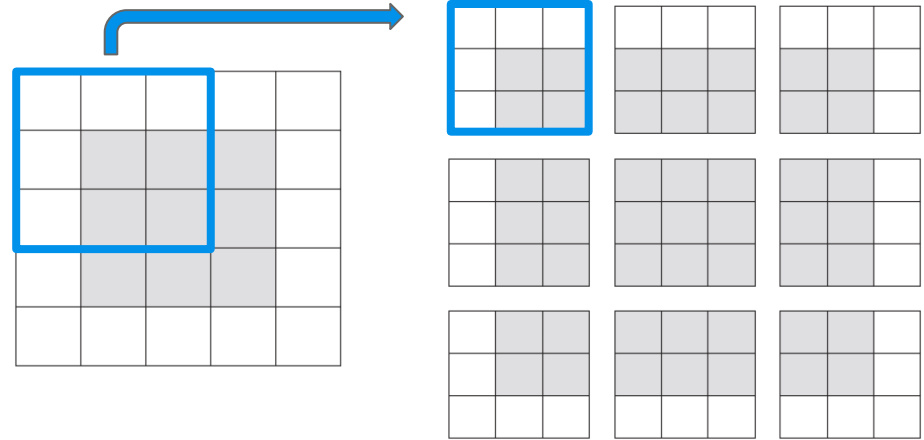
- ◎ Convolution leverages 3 important ideas that can help improve a machine learning system:
 - **Sparse interactions** / sparse connectivity / sparse weights
 - ◎ Filters are smaller than the input images and thus fewer parameters (weights) need to be stored
 - ◎ This reduces computational expense and improves statistical efficiency
 - **Parameter sharing**
 - ◎ The same parameter is used for more than one function in the model
 - ◎ In a CNN, each element of the filter is applied to every position of the input

Padding

- ⦿ Applying a filter to an input image shrinks it - the output dimensions are smaller than the input dimensions
- ⦿ Additionally, when we perform the convolution operation on an input, the pixels on the edges aren't used as much as the pixels in the middle
- ⦿ To make the amount of information used more equal across pixels, you can use **padding**
- ⦿ Padding consists of adding an appropriate number of rows and columns to each side of the image (think a border of pixels)
- ⦿ **This enables an output with the same dimensions as the input**

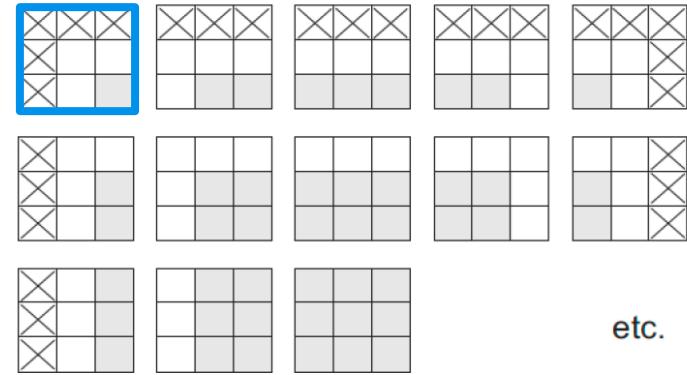
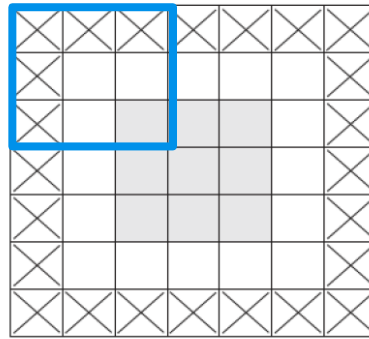
Valid Padding

- In Keras, the default is **no padding**, or “**valid**” padding
- This means the output will not be the same dimension as the input, and will instead depend on the dimension of the input and the size of the filter
- On the right is a 5 x 5 image
- If we apply a 3 x 3 filter, the output will also be 3 x 3



Same Padding

- On the right is the same 5 x 5 image, but with an added border
- If we use a 3 x 3 filter, we need to add $p = 1$ padding - here, p is the number of rows to add to the border of an image
- The output will then be 5 x 5
- Because the input and output have the same dimensions, this is called **“same” padding**



etc.

General Padding Formula

- ⊙ There is a general formula that can help you decide how much padding you need or want
 - Let the input be $n \times m$ and the filter $f \times f$
 - Without padding, the output would be:

$$(n - f + 1) \times (m - f + 1)$$

- ⊙ For an output with the same dimension as the input, need p such that:

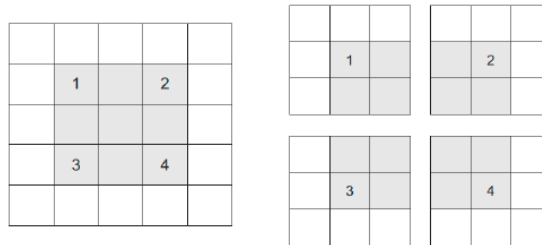
$$n \times m = (n + 2p - f + 1) \times (m + 2p - f + 1)$$

which boils down to $p = \frac{f - 1}{2}$

Convolutional Strides

- ◎ Another factor that can influence output size is convolution **strides**
- ◎ So far we have assumed that we slide the filter over a single space to extract a new patch - but what if we wanted to slide over 2 spaces, 3 spaces, or more?
- ◎ Convolutional strides are convolutions with a stride greater than 1
- ◎ If your stride is set equal to 2, you will downsample the width and height of the input image by a factor of 2
- ◎ If s is the size of the stride, the output will have dimensions

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{m + 2p - f}{s} + 1 \right\rfloor$$



Pooling

- ◎ Strided convolutions are rarely used in practice - **max-pooling** is more common
- ◎ A typical layer of a CNN consists of 3 stages:
 - **Convolution stage** - linear transformation of the input
 - **Detector stage** - Nonlinear activation (ex: relu activation function is applied)
 - **Pooling stage** - further modification (downsizing) of the output
- ◎ A pooling function replaces the output at a certain location with a summary statistic of the nearby outputs
- ◎ Pooling greatly reduces the computational expense of the network by decreasing the number of parameters to be learned

Pooling

- ⊙ There are different types of pooling

- **Max Pooling**

- ⊙ Outputs the maximum value from a patch for each channel
 - ⊙ Similar to convolution, but instead of transforming patches via a learned linear transformation, they're transformed via a hardcoded max tensor operation
 - ⊙ Very common
 - ⊙ Usually done with 2 x 2 windows

- **Average Pooling**

- **Weighted Pooling**

- ⊙ Based on the distance from the central pixel

Max pooling tends to work better than other pooling functions and convolutional strides

Max Pooling

Input

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

maxpool →

Output

8	6
9	9

Average Pooling

Average Pooling (kernel: (2, 2), stride: (2, 2), padding: (0, 0))

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

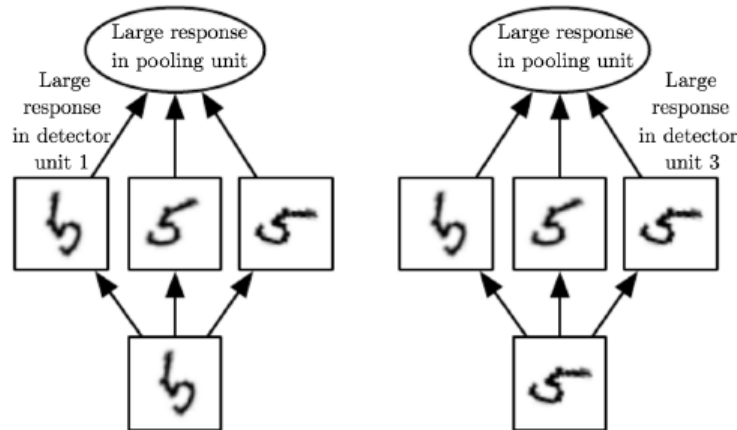
Input

7.75		

Output

Pooling and Invariance

- ◎ A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input
- ◎ Here, a set of 3 learned filters and a max pooling unit can learn to become invariant to rotation



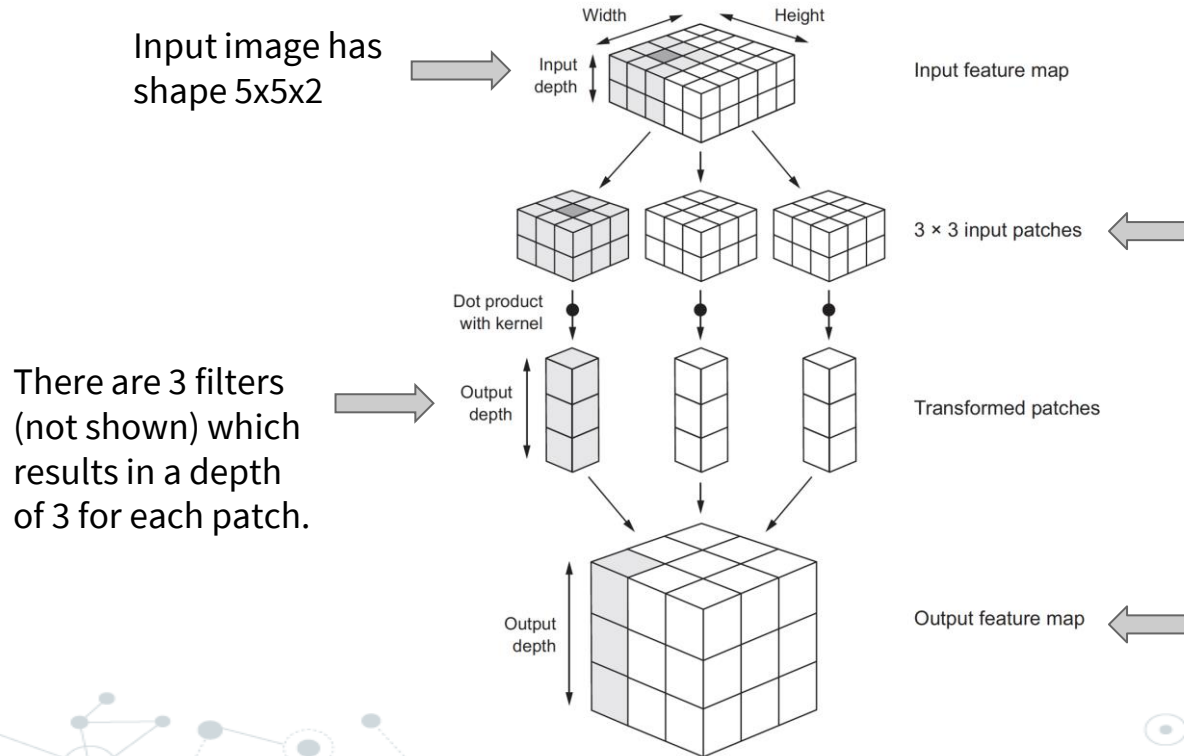
Terminology

- ◎ Convolutions operate over 3D tensors with two spatial axes, **height** and **width**, as well as a **depth** axis (or **channels** axis)
- ◎ For a color (RGB) image, the depth is equal to 3
- ◎ For black and white (grayscale) images, the depth is equal to 1
- ◎ The convolution operation extracts different **patches** from the input image and applies the same transformation to each of them, resulting in a response map that is also a 3D tensor

Terminology

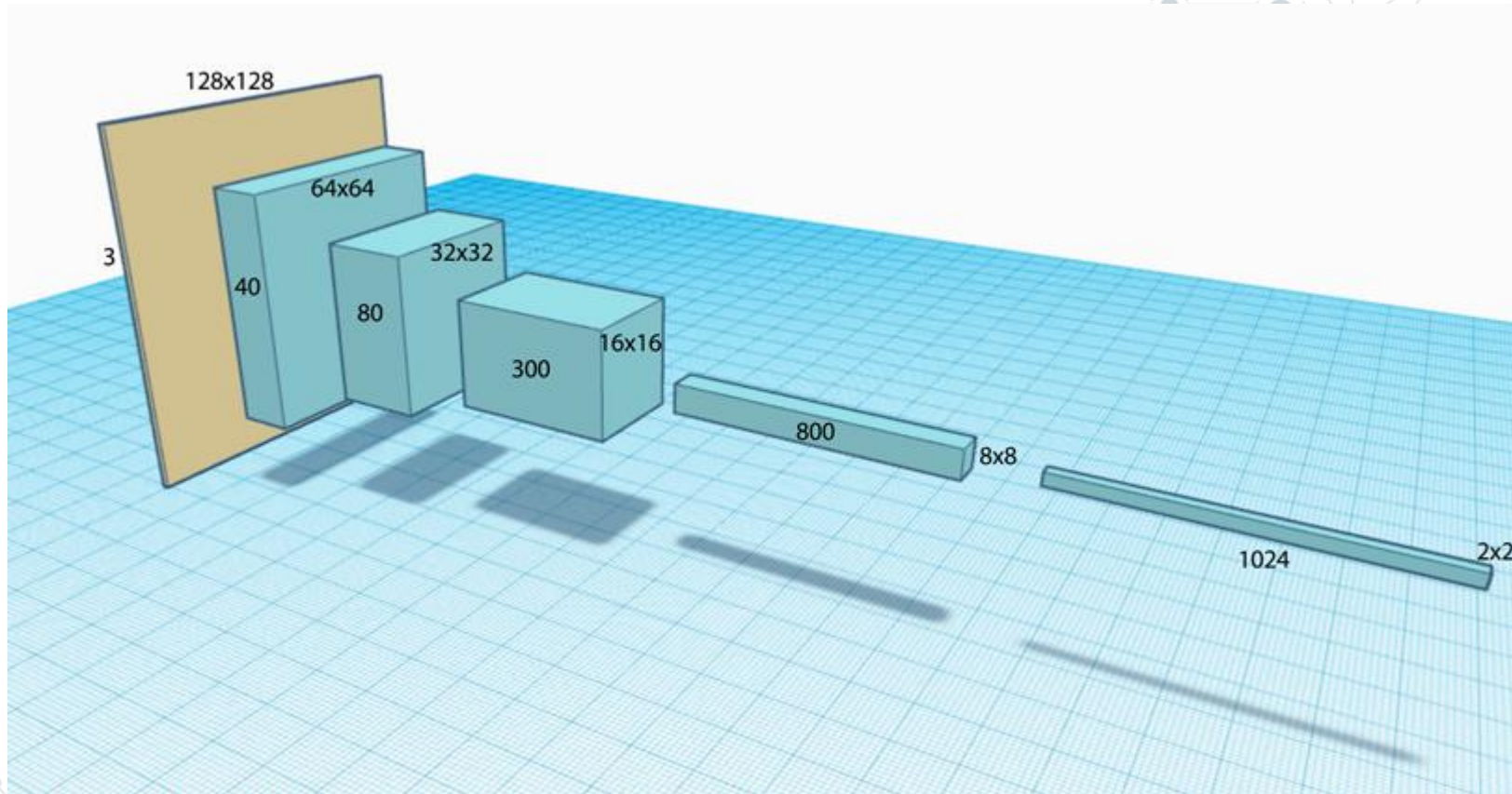
- ◎ The response map has a width, height, and depth, all of which depend on the input image, filter, padding and stride
- ◎ Convolutions are defined by 2 key parameters:
 - Size of the filter
 - Depth of the output response map, i.e., how many filters are applied to the input
- ◎ Convolution works by sliding the filter over the 3D input image, stopping at every possible location, and extracting the 3D patch at each location
 - Each 3D patch is transformed into a 1D vector
 - All 1D vectors are then reassembled into a 3D output map

Convolution Schematic



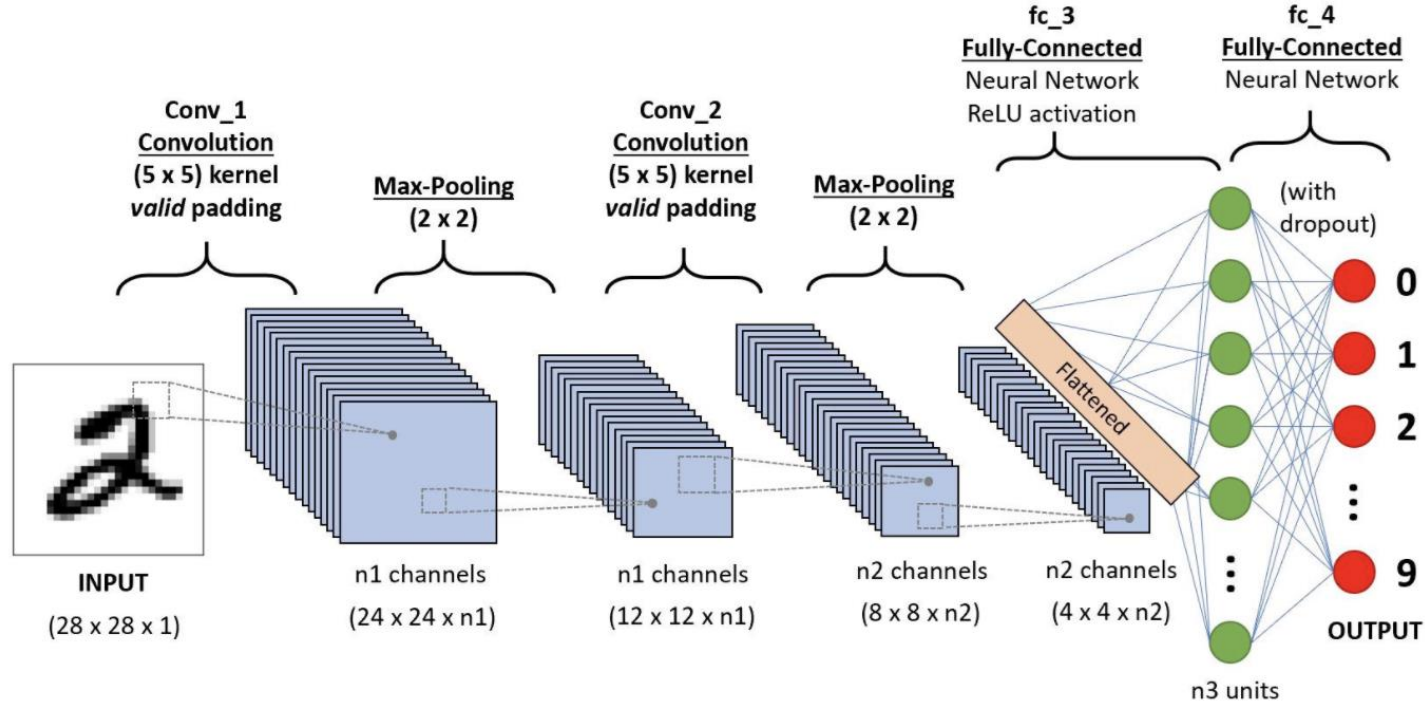
Here, each filter has a shape of $3 \times 3 \times 2$, which results in a total of 9 patches from the original input. Only 3 of those patches are shown here.

9 total patches, each with a depth of 3 (one number for each filter), resulting in an output shape of $3 \times 3 \times 3$.



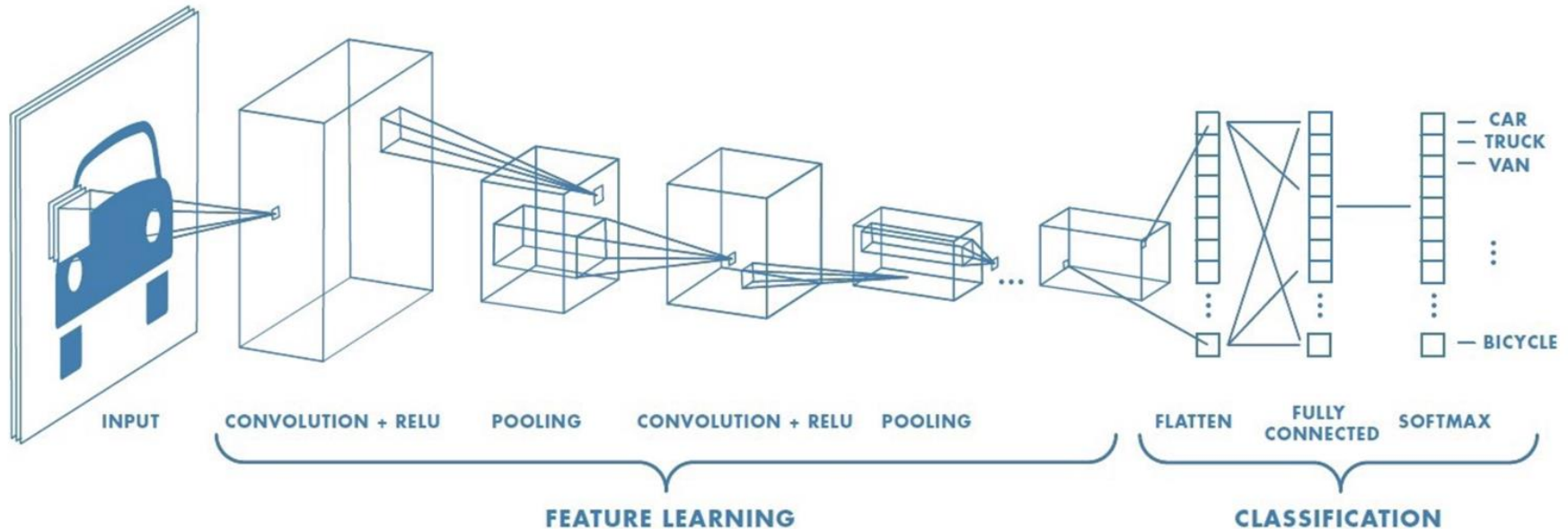
Visualization of CNN. Jonathan Hui Blog. <https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/>

CNN Schematic

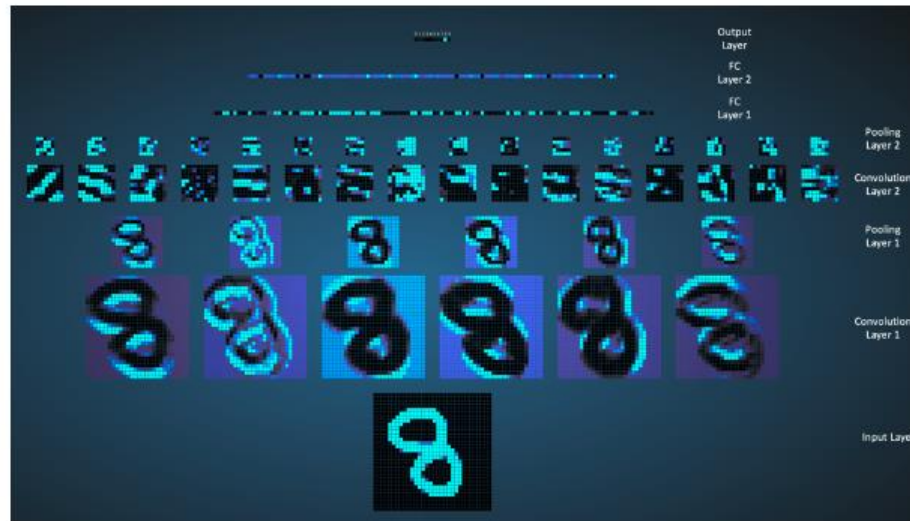


<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

CNN Schematic



MNIST Example




MNIST Example

```
1 # Define model
2 model = keras.Sequential([
3     # Convolutional layer with 32 filters that are 3x3, relu activation function
4     layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
5     # Pooling layer with 2x2 windows
6     layers.MaxPooling2D((2, 2)),
7
8     # Convolutional layer with 64 filters that are 3x3, relu activation function
9     layers.Conv2D(64, (3, 3), activation='relu'),
10    # Pooling layer with 2x2 windows
11    layers.MaxPooling2D((2, 2)),
12
13    # Convolutional layer with 64 filters that are 3x3, relu activation function
14    layers.Conv2D(64, (3, 3), activation='relu'),
15
16    # Collapse the 3D tensor into a vector
17    layers.Flatten(),
18
19    # Fully connected layer with 64 hidden units, relu activation function
20    layers.Dense(64, activation='relu'),
21
22    # Softmax output function with 10 classes
23    layers.Dense(10, activation='softmax')
24 ])
```

Have to provide
the shape of the
data only in the
first layer

MNIST Example

[Colab notebook](#)



Why we need non-linearity (intensity patterns)

How many parameters do we need in a CNN?