Computer Vision I

Fundamentals of Artificial Intelligence

Instructor: Chenhui Chu

Email: chu@i.kyoto-u.ac.jp

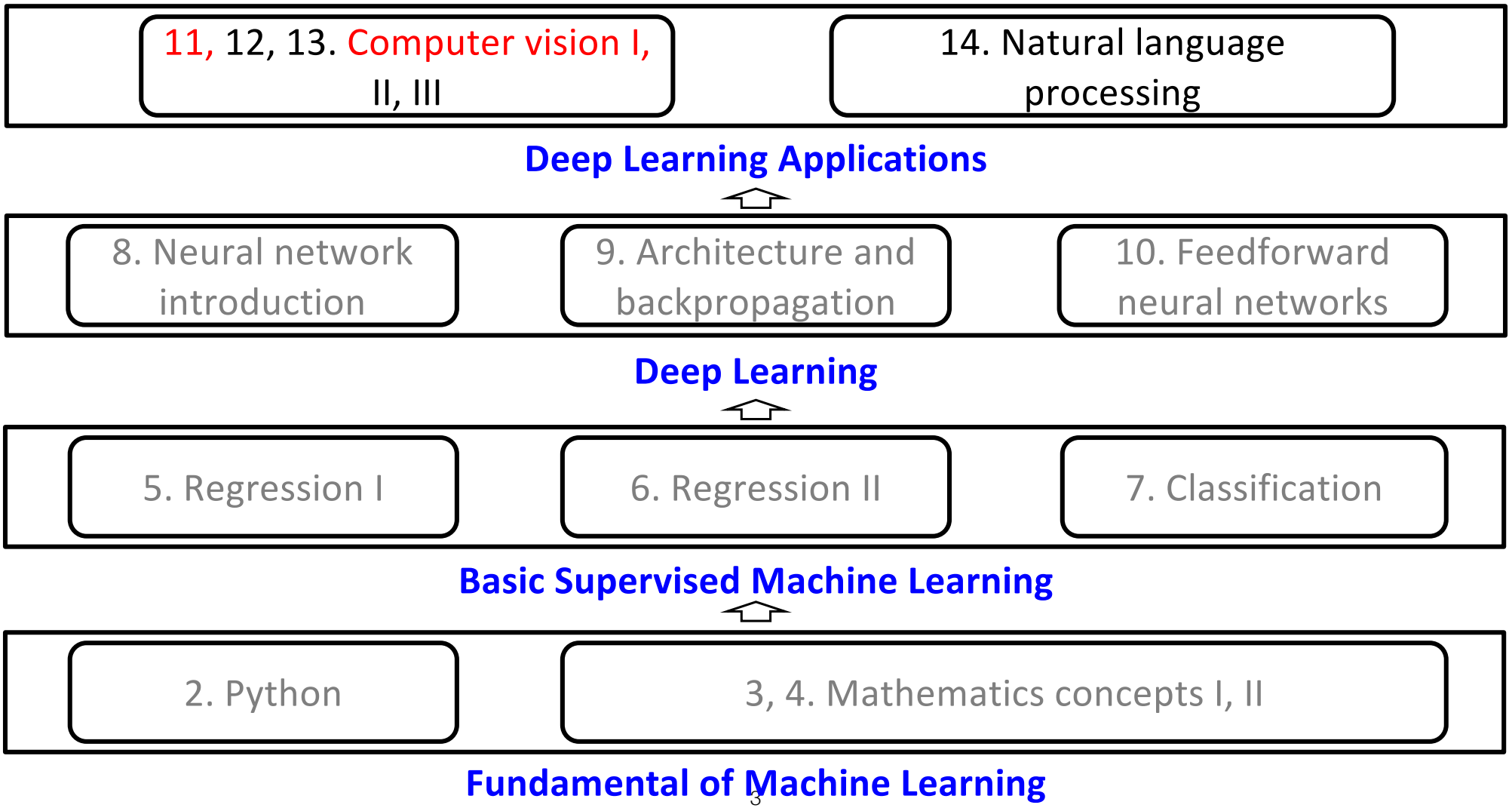
Teaching Assistant: Youyuan Lin

E-mail: youyuan@nlp.ist.i.kyoto-u.ac.jp

Schedule

- 1. Overview of AI and this Course (4/14)
- 2. Introduction to Python (4/21)
- 3, 4. Mathematics Concepts I, II (4/28, 5/12)
- 5, 6. Regression I, II (5/19, 5/26)
- 7. Classification (6/2)
- 8. Introduction to Neural Networks (6/9)
- 9. Neural Networks Architecture and Backpropagation (6/16)
- 10. Fully Connected Layers (6/23)
- 11, 12, 13. Computer Vision I, II, III (6/30, 7/7, 7/14)
- 14. Natural Language Processing (7/17)

Overview of This Course



Computer Vision (1/3)

- Computer Vision: Having computer understand images the way human do

Object detection

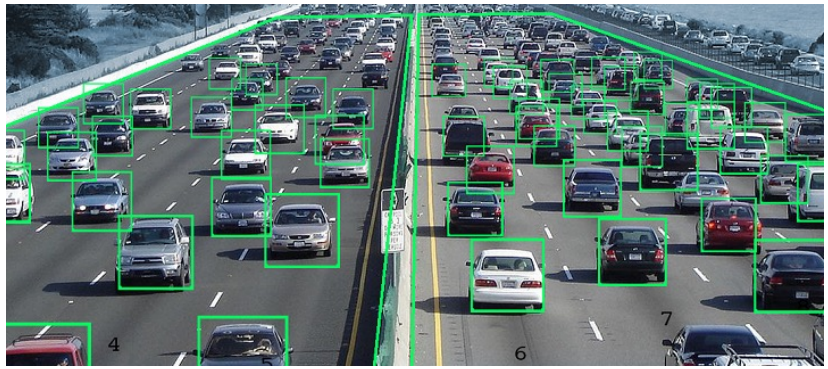


Image: http://vision.seecs.edu.pk/ats/

Image Recognition

CAT

Emotion Recognition

SAD

“FaceID”

Is phone owner?

Yes

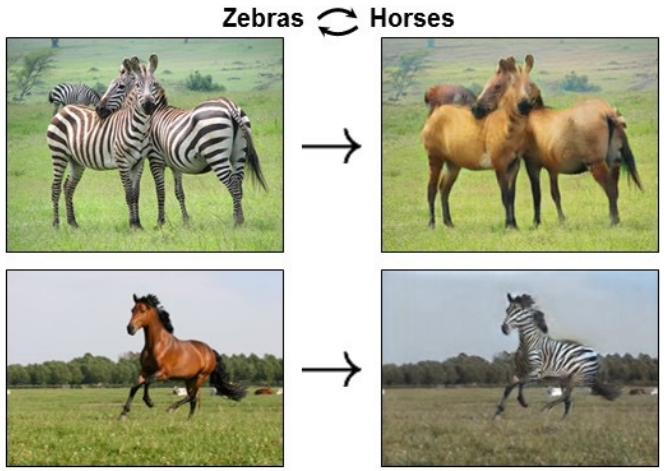
No

https://www.flickr.com/photos/dalemoorephotography/4445558459

Computer Vision (2/3)

- Recently, also much progress in image (and video) generation

Image from Nvidia



Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Computer Vision (3/3)

- Computer Vision: Having computer understand images the way human do

Object detection

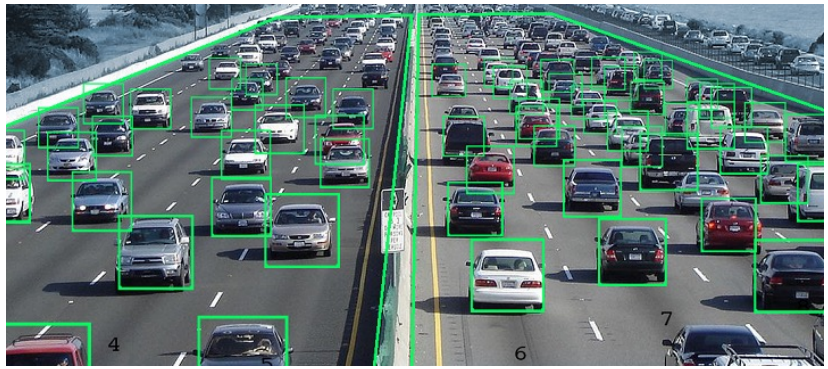


Image: http://vision.seecs.edu.pk/ats/

Classification tasks

Image Recognition

CAT

Emotion Recognition

SAD

"FaceID"

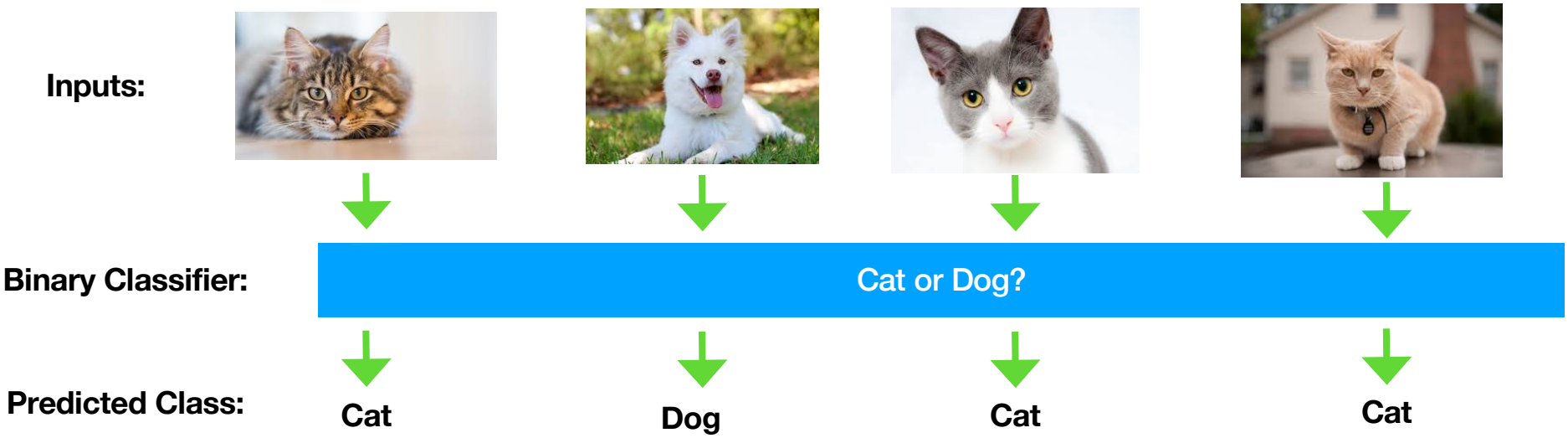
Is phone owner?

Yes

No

https://www.flickr.com/photos/dalemoorephotography/4445558459

Binary Classification (1/2)

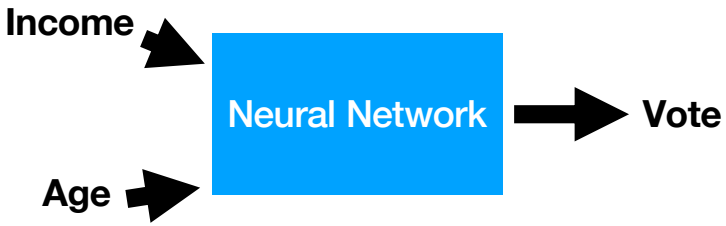


- Remember that binary classifiers are about assigning one of two categories to an input (voting for LEFT or RIGHT-wing party, being a CAT or a DOG, etc…)

Binary Classification (2/2)

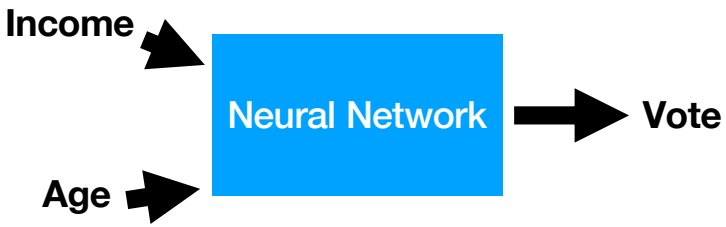
- We know that we can use a logistic classifier to do binary classification:

- We know that we can actually combine many Logistic Classifiers (a.k.a "Neurons" to be able to do more complex processing

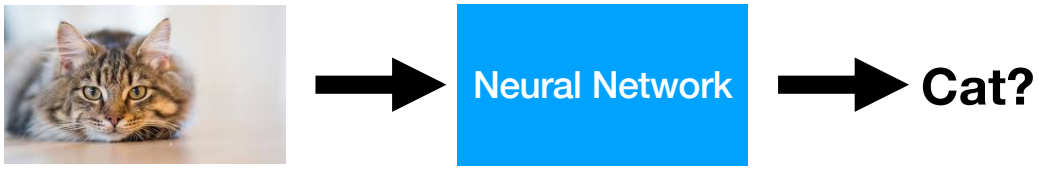


Binary Classification for Computer Vision

- We know that we can actually combine many Logistic Classifiers (a.k.a “Neurons” to be able to do more complex processing

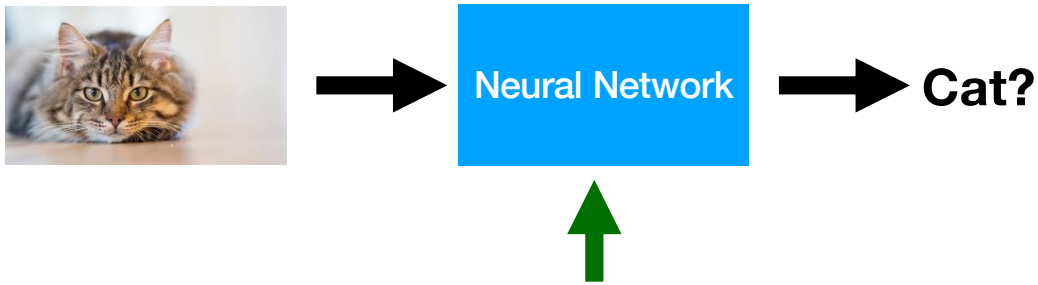


- We just want to do the same, but with an image as input



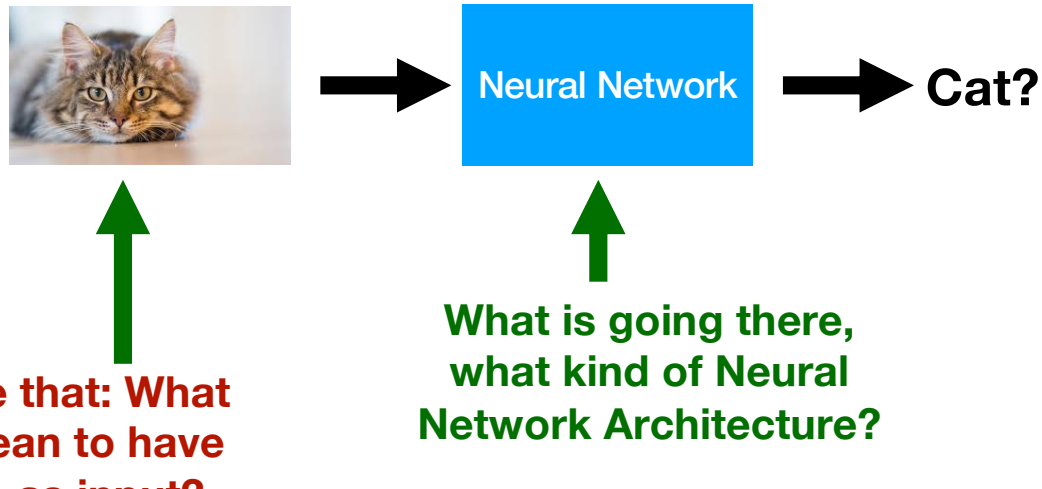
Things to Consider (1/2)

- We just want to do the same, but with an image as input



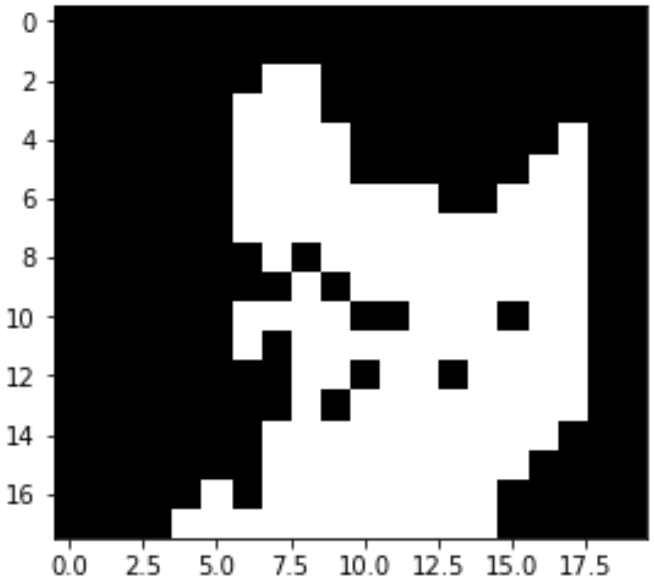
Things to Consider (2/2)

- We just want to do the same, but with an image as input



What is an Image? (1/4)

- For a computer, an image is an array of **pixels**
- Each pixel is one point in the image and has a given color

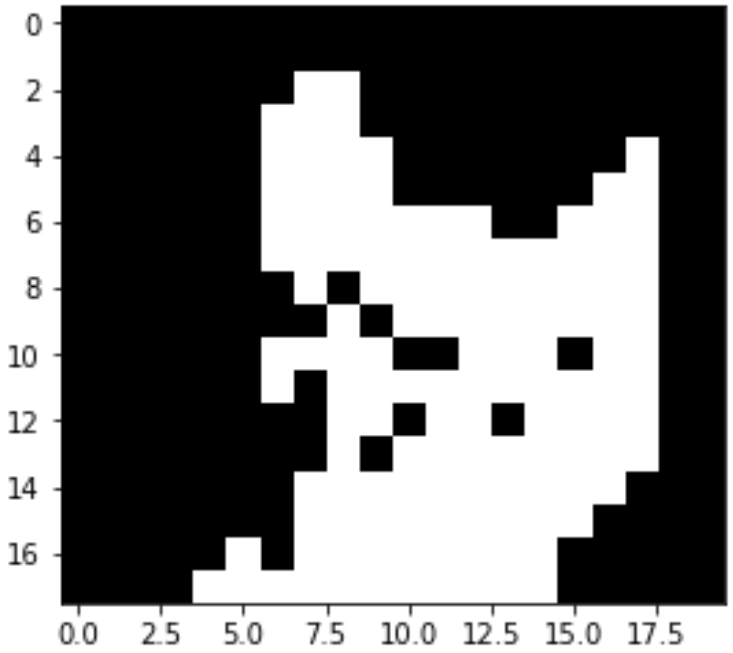


What is an Image? (2/4)

- 1-bit image: Each ***pixel*** is either black (**0**) or white (**1**)

- Our image is an array of 0 and 1

Image with 18x20 pixels (1-bit)



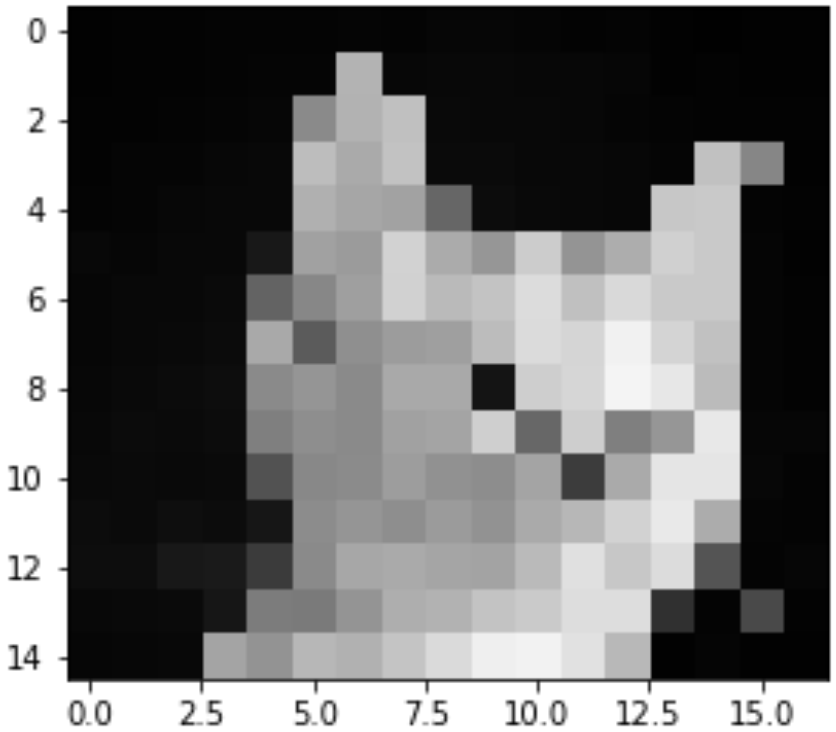
Array with 18x20 numbers

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

What is an Image? (3/4)

- Greyscale image: each pixel has a grey value between 0 (black) and 1 (white)

Image with 18x20 pixels (greyscale)

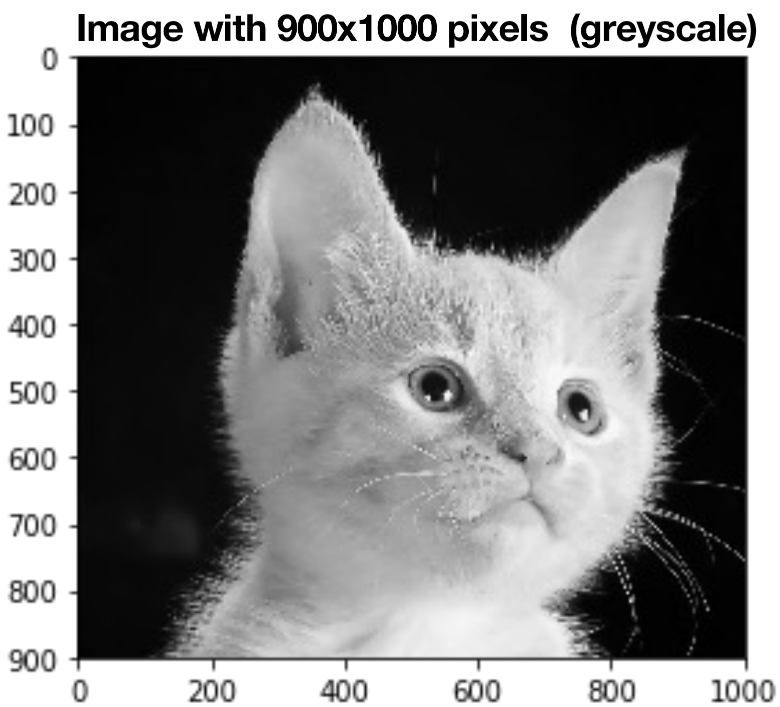


Array with 18x20 numbers

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.6	0.4	0.0	0.0	0.0	0.0	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.1	0.6	0.6	0.8	0.7	0.6	0.8	0.6	0.7	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.6	0.8	0.7	0.8	0.9	0.8	0.9	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.7	0.4	0.6	0.6	0.6	0.7	0.9	0.8	0.9	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.6	0.5	0.7	0.7	0.1	0.8	0.8	1.0	0.9	0.7	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.6	0.5	0.6	0.6	0.8	0.4	0.8	0.5	0.6	0.9	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.5	0.5	0.6	0.6	0.6	0.6	0.2	0.7	0.9	0.9	0.0	0.0
0.0	0.0	0.1	0.0	0.1	0.5	0.6	0.6	0.6	0.6	0.7	0.7	0.8	0.9	0.7	0.0	0.0
0.1	0.1	0.1	0.1	0.2	0.5	0.7	0.7	0.6	0.6	0.7	0.9	0.8	0.9	0.3	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	0.2	0.0	0.3	0.0
0.0	0.0	0.0	0.6	0.6	0.7	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.0	0.0	0.0	0.0

What is an Image? (4/4)

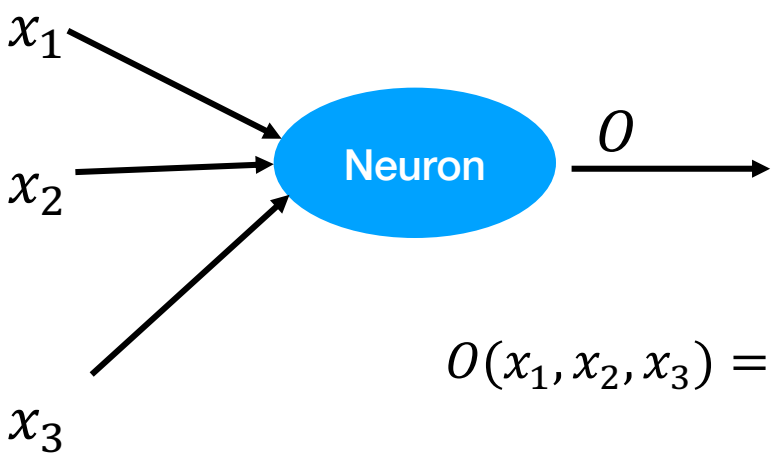
- The resolution is the number of pixels in the image



0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Neuron

- It turns out, our images are just array of numbers
- We know we can give numbers as input to neurons

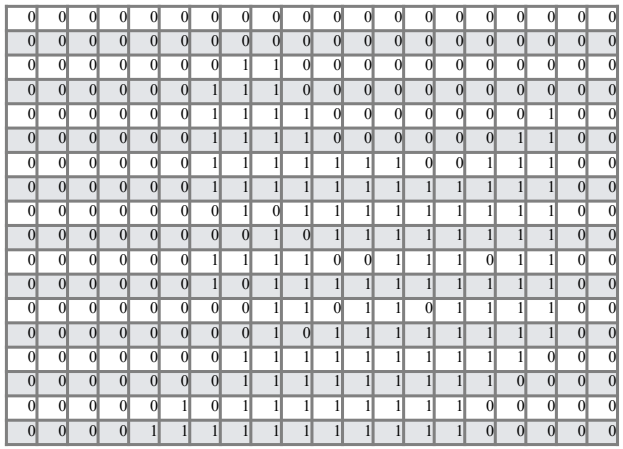


$$O(x_1, x_2, x_3) = \sigma(\theta_0 + \theta_1 \times x_1 + \theta_2 \times x_2 + \theta_3 \times x_3)$$

Binary Classification for An Image

- First problem solved:

Array with 18x20 numbers



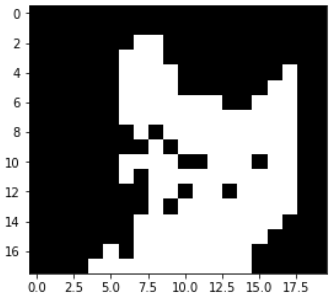


Image with 18x20 pixels

Neural Network

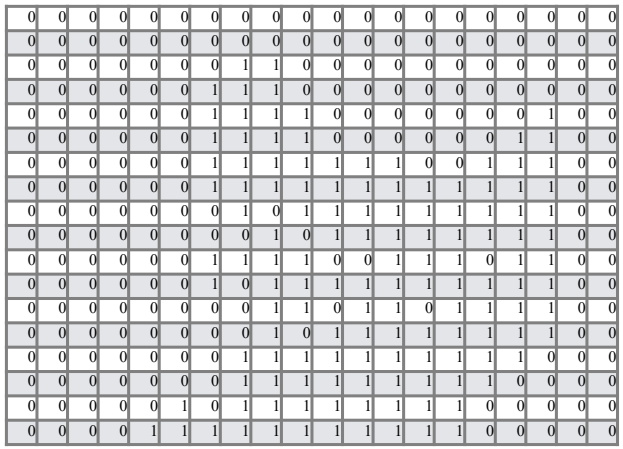
Cat?

What is going there, what kind of Neural Network Architecture?

Single Neuron (1/2)

- How about trying the simplest thing?

Array with 18x20 numbers



Logistic classifier with 18x20 = 360 inputs

Cat?

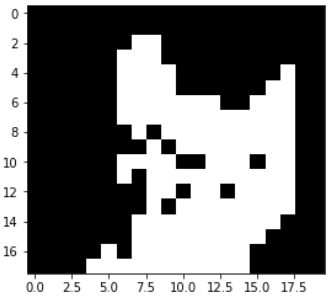


Image with 18x20 pixels

Output is computed like this:

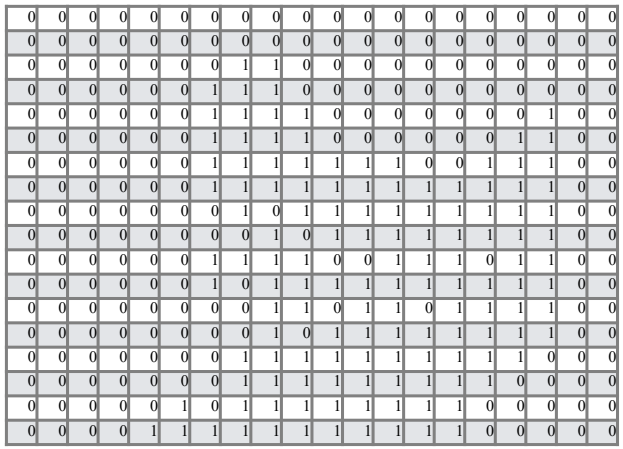
$$O(x_{1x1}, \ldots, x_{18x20}) = \sigma(\theta_0 + \theta_1 \times x_{1x1} + \theta_2 \times x_{2x1} + \ldots + \theta_{360} \times x_{18x20})$$

Single Neuron (2/2)

- How about trying the simplest thing?

Obviously, this will not work!!

Array with 18x20 numbers



Logistic classifier with 18x20 = 360 inputs

Cat?

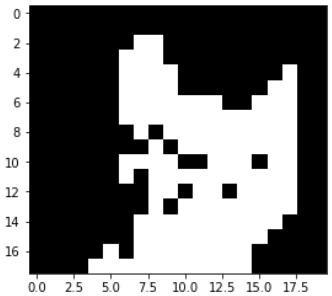


Image with 18x20 pixels

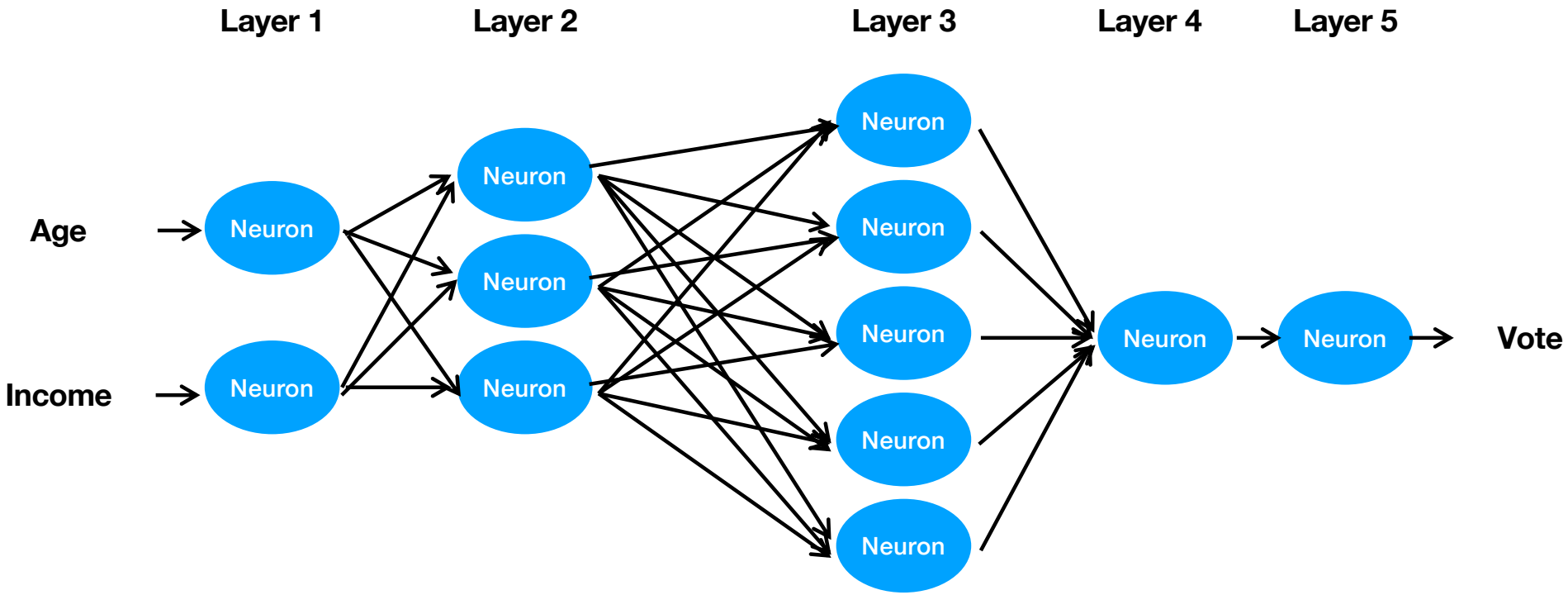
Output is computed like this:

$$O(x_1, x_2, x_3) = \sigma(\theta_0 + \theta_1 \times x_{1x1} + \theta_2 \times x_{2x1} + \ldots + \theta_{360} \times x_{18x20})$$

We are just summing the value of all the pixels (with some coefficients…)

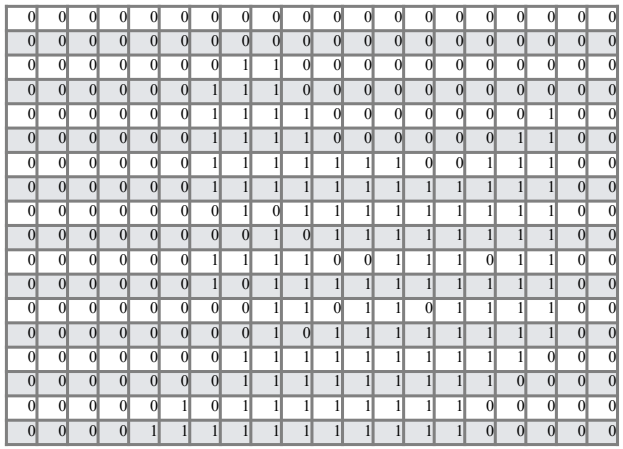
Feed-Forward Networks with Fully Connected Layers

- We have seen we can get much more powerful classifiers by combining neurons in neural networks



FFN for an Image (1/9)

Array with 18x20 numbers



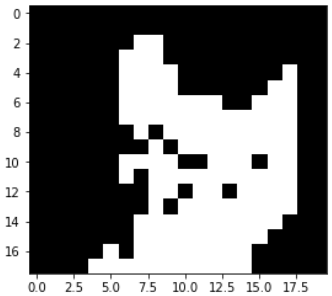


Image with 18x20 pixels

Fully Connected layer with 1000 neurons with 18x20 = 360 inputs each

Fully Connected layer with 200 neurons with 1000 inputs each

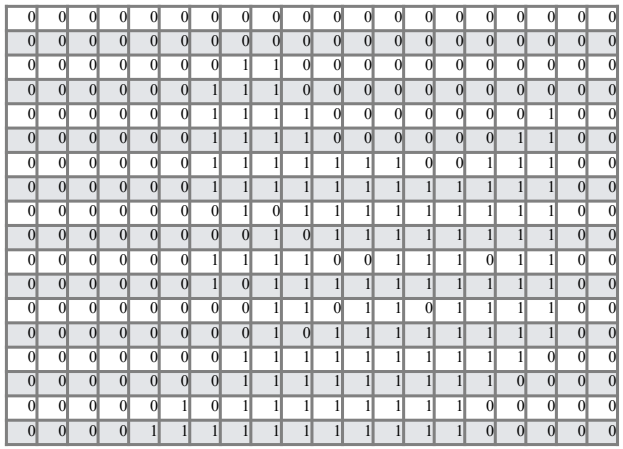
Final logistic classifier 200 inputs 1 output

Cat?

FFN for an Image (2/9)

Will it work? Maybe....

Array with 18x20 numbers



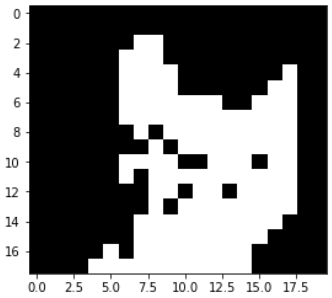


Image with 18x20 pixels

Fully Connected layer with 1000 neurons with 18x20 = 360 inputs each

Fully Connected layer with 200 neurons with 1000 inputs each

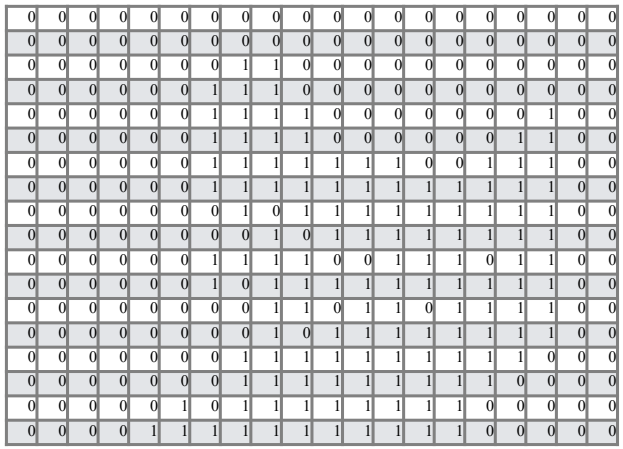
Final logistic classifier 200 inputs 1 output

Cat?

... but what might be the problems here?

FFN for an Image (3/9)

Array with 18x20 numbers



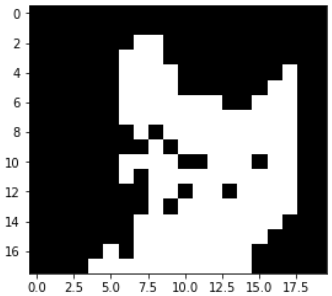


Image with 18x20 pixels

Fully Connected layer with 1000 neurons with 18x20 = 360 inputs each

Fully Connected layer with 200 neurons with 1000 inputs each

Final logistic classifier 200 inputs 1 output

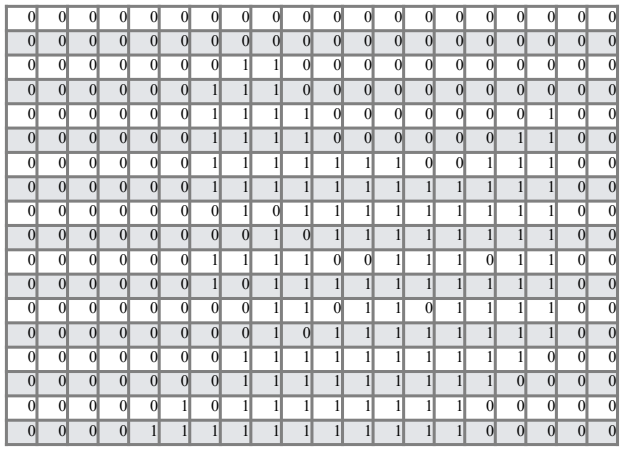
Cat?

Problem 1: The network gets no spatial information

FFN for an Image (4/9)

Array with 18x20 numbers

For example, the network does not know that these two inputs correspond to pixels that are close to each other



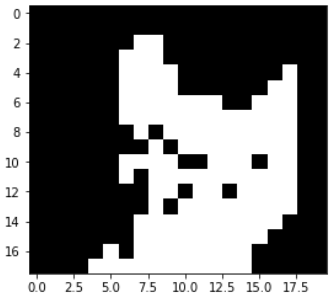


Image with 18x20 pixels

Fully Connected layer with 1000 neurons with 18x20 = 360 inputs each

Fully Connected layer with 200 neurons with 1000 inputs each

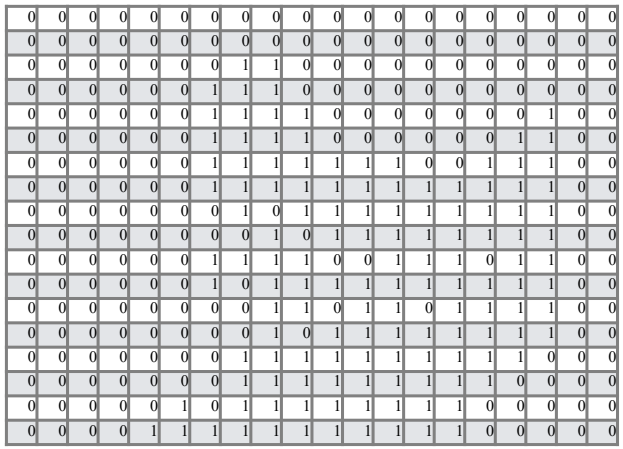
Final logistic classifier 200 inputs 1 output

Cat?

Problem 1: The network gets no spatial information

FFN for an Image (5/9)

Array with 18x20 numbers



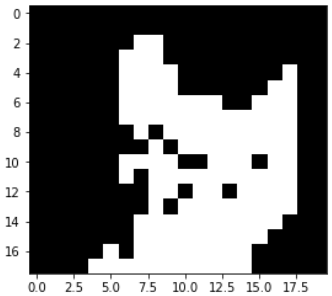


Image with 18x20 pixels

Fully Connected layer with 1000 neurons with 18x20 = 360 inputs each

Fully Connected layer with 200 neurons with 1000 inputs each

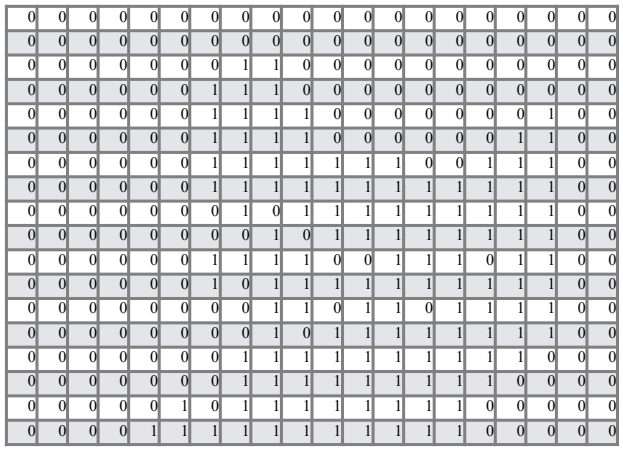
Final logistic classifier 200 inputs 1 output

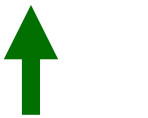
Cat?

Problem 2: How many parameters are we using in our Neural Network?

FFN for an Image (6/9)

Array with 18x20 numbers





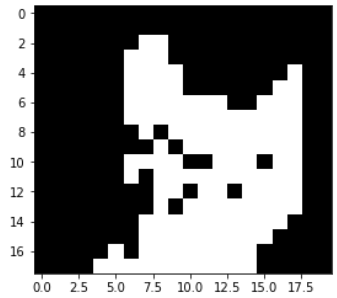
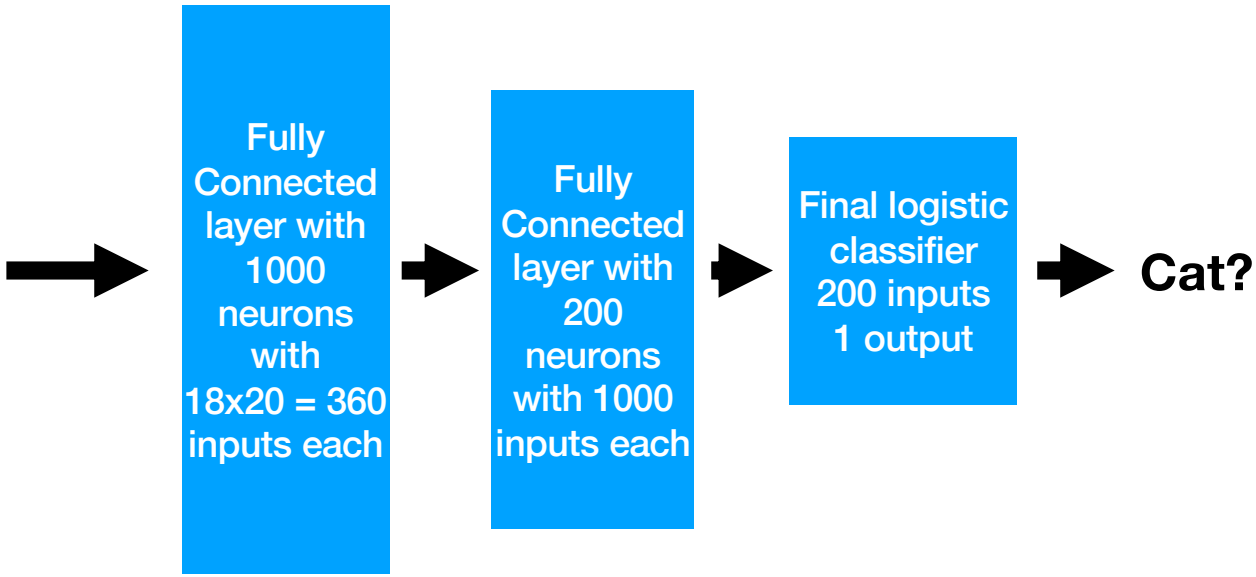


Image with 18x20 pixels



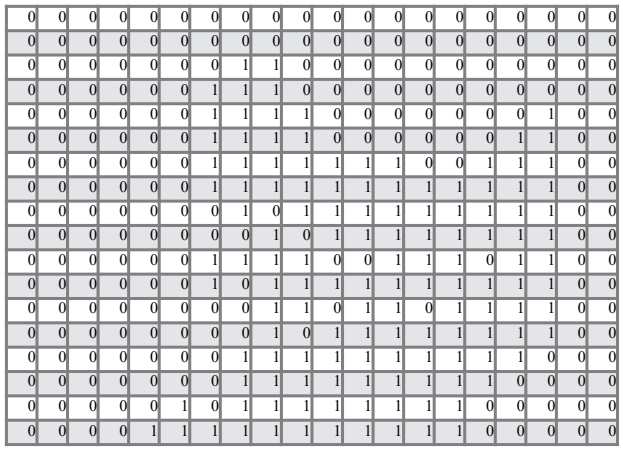
Problem 2: How many parameters are we using in our Neural Network?

(360+1) x 1000 + (1000+1) x 200 + (200+1)x1 = 561401 parameters

FFN for an Image (7/9)

561,000 parameters: quite many, but might still be OK
But how about images with higher resolution?

Array with 18x20 numbers



Fully Connected layer with 1000 neurons with 18x20 = 360 inputs each

Fully Connected layer with 200 neurons with 1000 inputs each

Final logistic classifier 200 inputs 1 output

Cat?

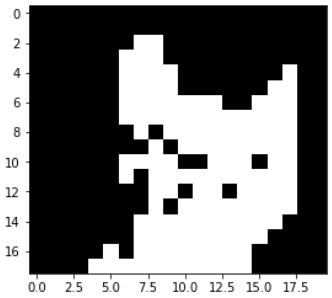


Image with 18x20 pixels

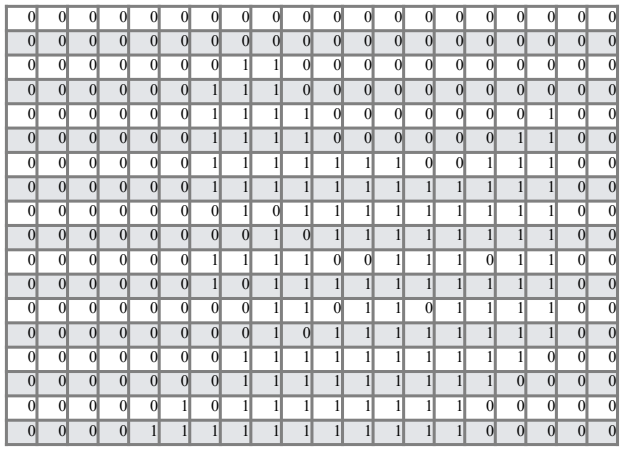
Problem 2: How many parameters are we using in our Neural Network?

(360+1) x 1000 + (1000+1) x 200 + (200+1)x1 = 561401 parameters

FFN for an Image (8/9)

- How about trying the simplest thing?

Array with 900x1000 numbers



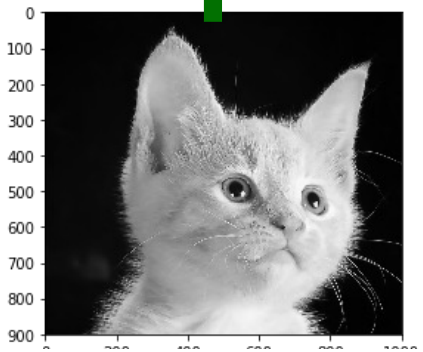


Image with 900x1000 pixels (greyscale)

Fully Connected layer with 1000 neurons with 900x1000 = 900000 inputs each

Fully Connected layer with 200 neurons with 1000 inputs each

Final logistic classifier 200 inputs 1 output

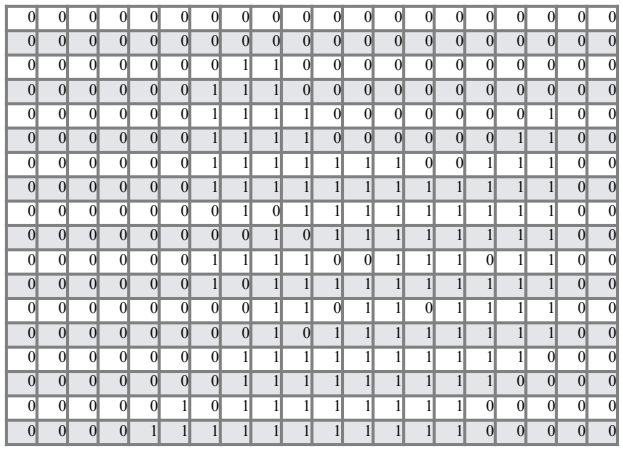
Cat?

Now, how many parameters are we using in our Neural Network?

FFN for an Image (9/9)

- How about trying the simplest thing?

Array with 900x1000 numbers



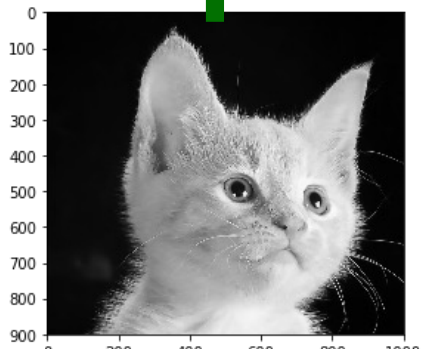


Image with 900x1000 pixels (greyscale)

Fully Connected layer with 1000 neurons with 900x1000 = 900000 inputs each

Fully Connected layer with 200 neurons with 1000 inputs each

Final logistic classifier 200 inputs 1 output

Cat?

We now have almost a billion parameters. It is a lot!!!

Now, how many parameters are we using in our Neural Network?

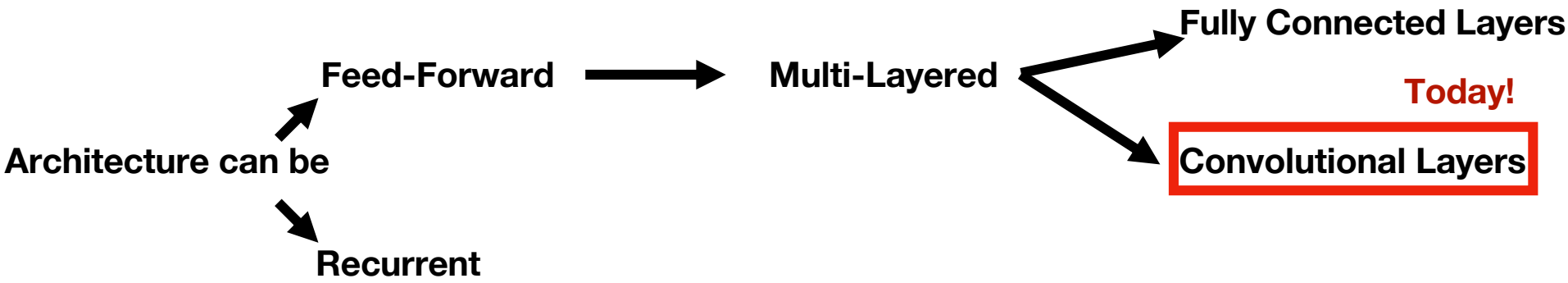
Problems of FFN for Computer Vision

- As we saw, there are two problems with using Fully-Connected Layers for Computer vision:
  - The fully connected layer gets no spatial information (neighbouring pixels and far away pixels are treated similarly)
  - For high-resolution images, we need a very large number of parameters

Convolutional Layers

- The solution to these preoblems will be the ***Convolutional Layers***

Neural Network Architectures



Convolutional Layers

- Like Fully Connected Layers, Convolutional Layers appear in multi-layered Feed-Forward architectures

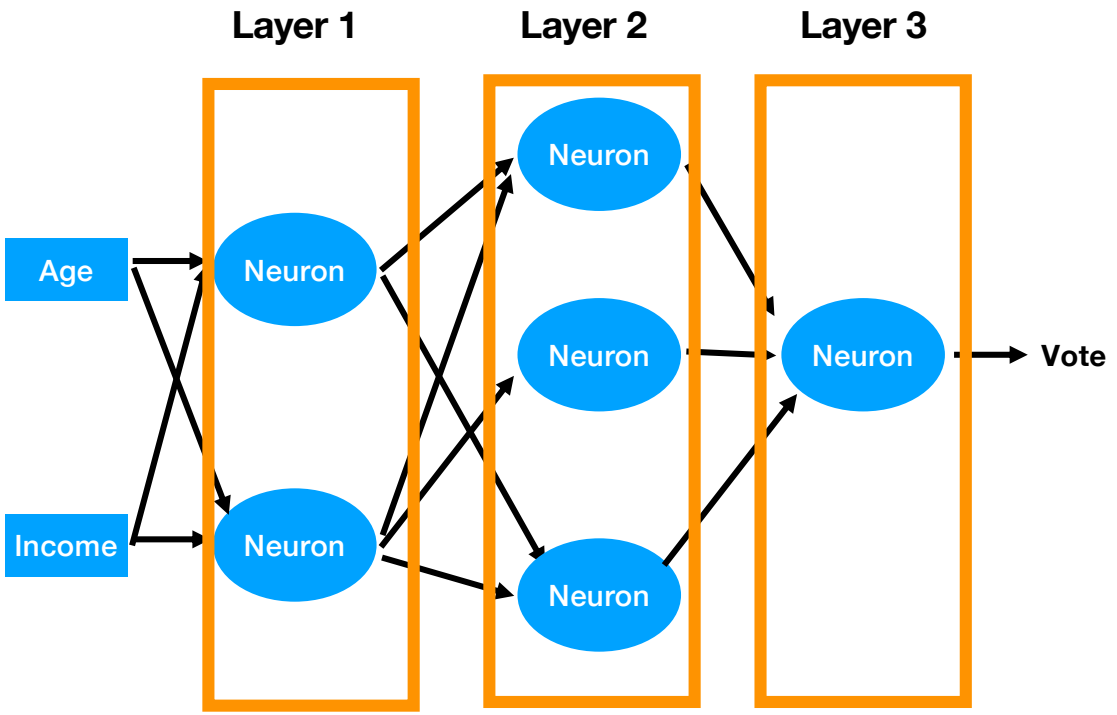
Feed Forward Architectures

- In the case of a feed-forward architecture, we often organize neurons in ***layers***

- Rules:

  1. A neuron is never connected to a neuron in the same layer

  2. A neuron output only goes in the input of a neuron in the next layer



- We will call this a Feed Forward Multi-Layer Architecture

Convolutional Layers (1/4)

- Like Fully Connected Layers, Convolutional Layers appear in multi-layered Feed-Forward architectures
- Therefore, we still have this idea that neurons in one layer only take input from the previous layer
- In a Fully-Connected layer, each neuron in one layer takes input from all neurons in the previous layer
- In a Convolutional Layer, each neuron in one layer takes input from only some neurons in the previous layer

Convolutional Layers (2/4)

- Neurons are organized in 2-dimensional layers
- Neurons in 2 layers are only connected if they roughly belong to the same area of their respective layer
  - Eg. The neuron in the top-left corner of layer 2 is only connected to the 9 neurons in the top-left corner of layer 1

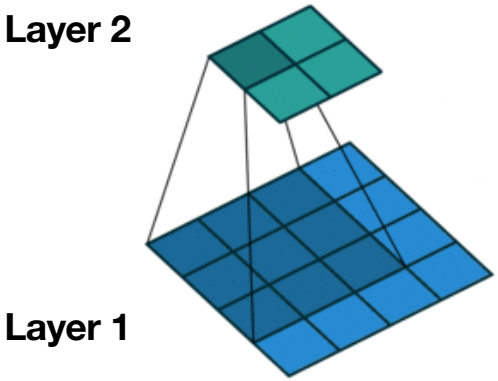


Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolutional Layers (3/4)

- Neurons are organized in 2-dimensional layers
- Neurons in 2 layers are only connected if they roughly belong to the same area of their respective layer
  - Eg. The neuron in the top-left corner of layer 2 is only connected to the 9 neurons in the top-left corner of layer 1

This gives spatial information to the network

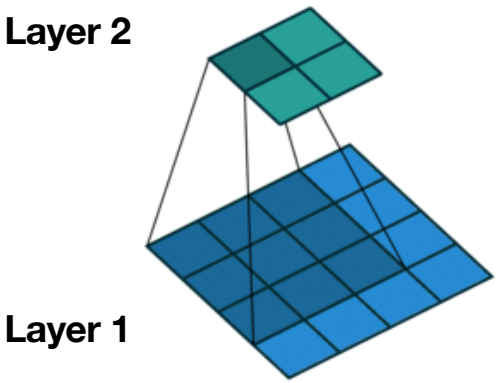
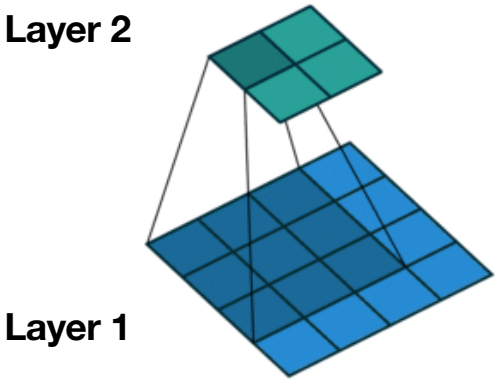


Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolutional Layers (4/4)

- Neurons are organized in 2-dimensional layers
- Neurons in 2 layers are only connected if they roughly belong to the same area of their respective layer
  - Eg. The neuron in the top-left corner of layer 2 is only connected to the 9 neurons in the top-left corner of layer 1



This gives spatial information to the network

Because all of the inputs of one neuron correspond to Neighboring pixels

Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

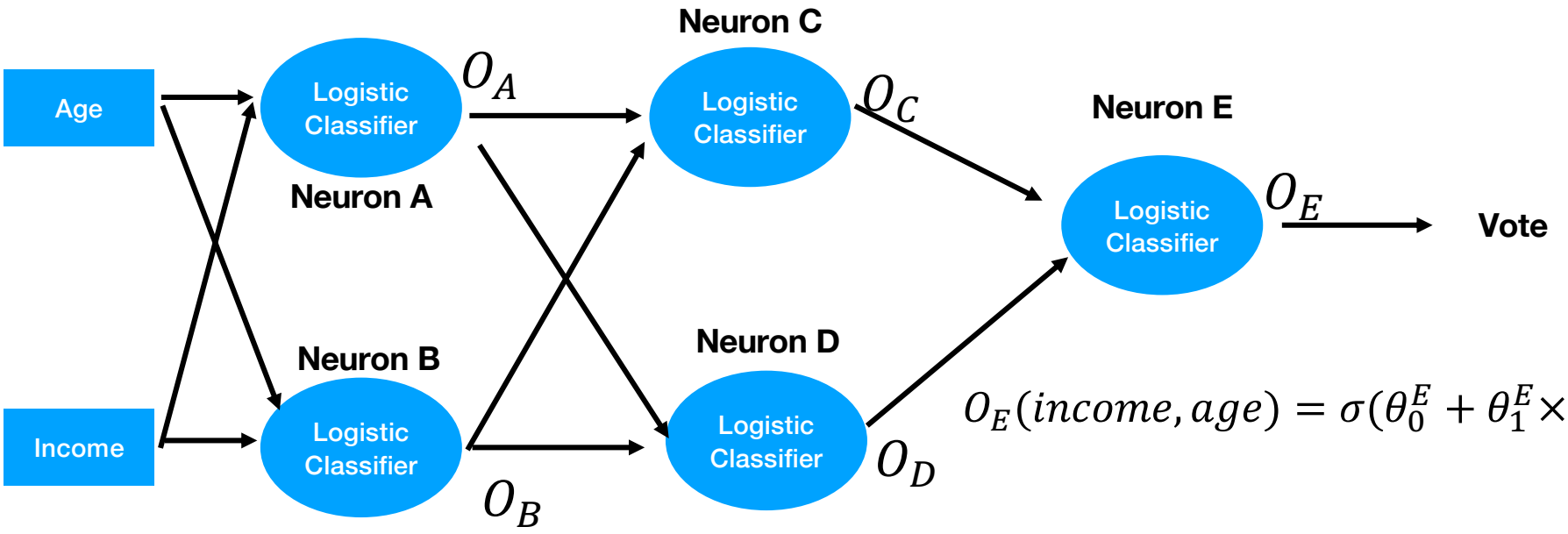
Weight Sharing (1/6)

- Another specificity of convolutional layers: their **weights** are **shared**
- This means all neurons in one layer actually have the same weights

Weight Sharing (2/6)

Weights not shared:

$$O_A(income, age) = \sigma(\theta_0^A + \theta_1^A \times income + \theta_2^A \times age)\, O_C(income, age) = \sigma(\theta_0^C + \theta_1^C \times O_A + \theta_2^C \times O_B)$$



$$O_E(income, age) = \sigma(\theta_0^E + \theta_1^E \times O_C + \theta_2^E \times O_D)$$

$$O_B(income, age) = \sigma(\theta_0^B + \theta_1^B \times income + \theta_2^B \times age)$$

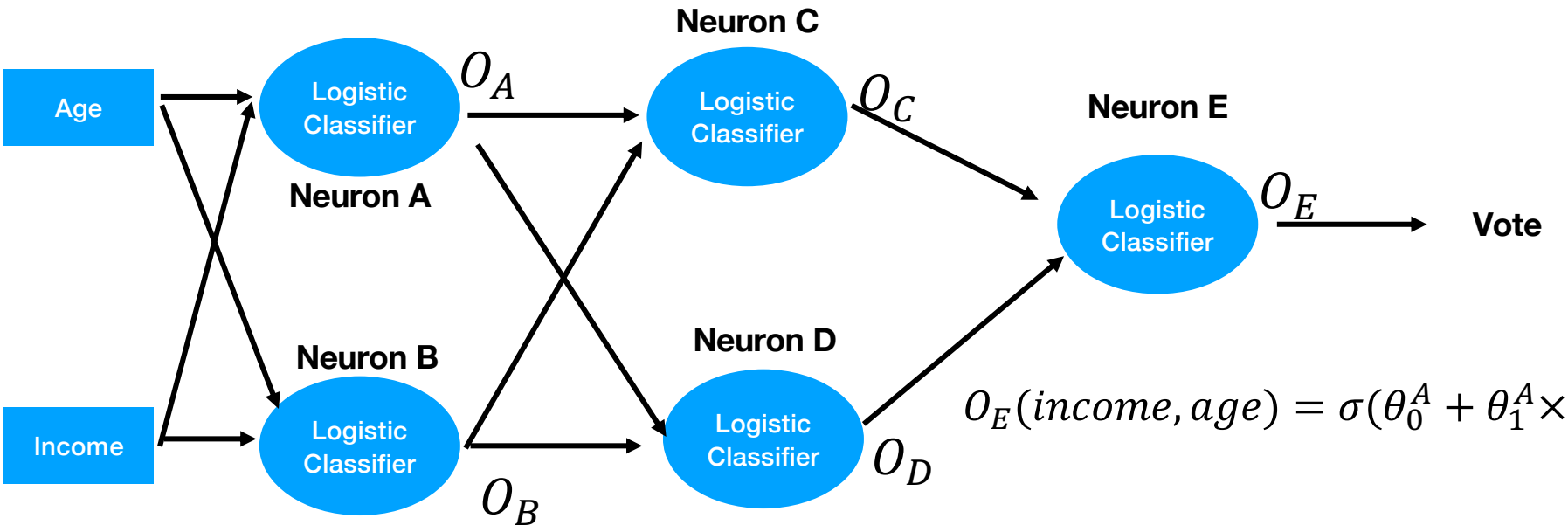
$$O_D(income, age) = \sigma(\theta_0^D + \theta_1^D \times O_A + \theta_2^D \times O_B)$$

Weight Sharing (3/6)

Weights shared:

$$O_A(income, age) = \sigma(\theta_0^A + \theta_1^A \times income + \theta_2^A \times age)$$

$$O_C(income, age) = \sigma(\theta_0^A + \theta_1^A \times O_A + \theta_2^A \times O_B)$$



$$O_E(income, age) = \sigma(\theta_0^A + \theta_1^A \times O_C + \theta_2^A \times O_D)$$

$$O_B(income, age) = \sigma(\theta_0^A + \theta_1^A \times income + \theta_2^A \times age)$$

$$O_D(income, age) = \sigma(\theta_0^A + \theta_1^A \times O_A + \theta_2^A \times O_B)$$

Weight Sharing (4/6)

- Another specificity of convolutional layers: their **weights** are **shared**
- This means all neurons in one layer actually have the same weights
- This is good for two reasons:
  - Much less parameters for each layer
  - Translation invariance of the network

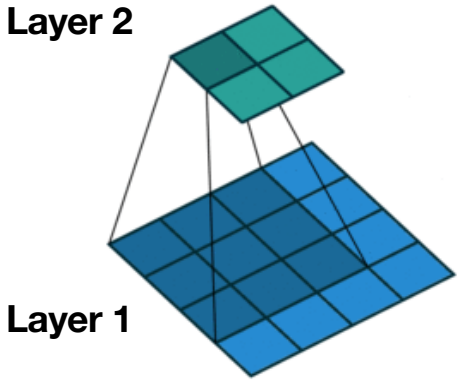


Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Weight Sharing (5/6)

- With that in mind, how many parameters would we actually have for the convolutional layer on the right?

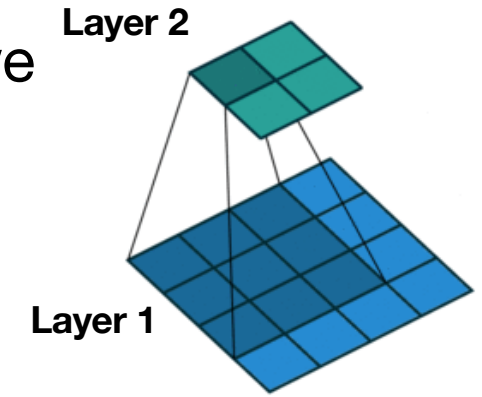
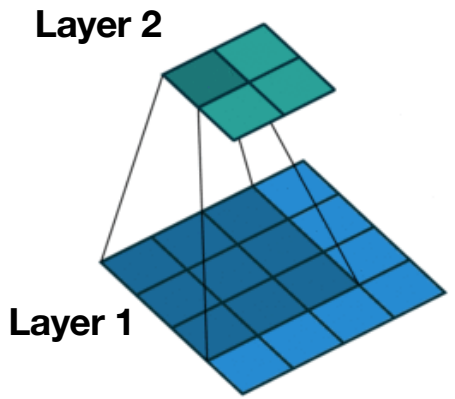


Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Weight Sharing (6/6)

- With that in mind, how many parameters would we actually have for the convolutional layer on the right?
  - First neuron has 9 input neurons -> 9 + 1 parameters
  - The three other neurons share the parameters:
    - 10 parameters in total
    - Instead of 40 parameters without weight sharing



  - How many parameters if it was a fully connected layer?

Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolution Layers: the Mathematical Point of View (1/2)

- We saw previously that the computation of Fully-Connected layers could be represented efficiently by the mathematical operation called ***Matrix Multiplication***

- Now, we are going to see which mathematical operation correspond to a convolution layer

Convolution Layers: the Mathematical Point of View (2/2)

- The mathematical operation we need to look at is called (unsurprisingly) the ***"convolution" operation***
  - (Note: in mathematic and signal processing, the operation we are going to describe is actually called *cross-correlation*)

Convolution

- A convolution operation takes 2 arrays:
  - The input array
  - The kernel array
- It produces **a new array**

Convolution (1/4)

Input Array

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

*

Kernel Array

2	0	2
0	1	0
-1	1	0

=

Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

Convolution (2/4)

- How we compute:

Input Array

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

*

Kernel Array

2	0	2
0	1	0
-1	1	0

=

Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

0x2 + 0x0 + 0x2 + 0x0 + 0x1 + 1x0 + 0x-1 + 1x1 + 1x0 = 1

Convolution (3/4)

- How we compute:

Input Array

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

*

Kernel Array

2	0	2
0	1	0
-1	1	0

=

Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

0x2 + 0x0 + 1x2 + 0x0 + 1x1 + 1x0 + 0x-1 + 1x1 + 1x0 = 4

Convolution (4/4)

- How we compute:

Input Array

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

*

Kernel Array

2	0	2
0	1	0
-1	1	0

=

Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

0x2 + 1x0 + 1x2 + 1x0 + 1x1 + 1x0 + 1x-1 + 1x1 + 1x0 = 3

Padding (1/2)

- In practice, if the kernel array has dimension ***k***x***k***, the output array dimension is reduced by ***(k-1)*** in each dimension

Input Array (6x6)

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

*

Kernel Array (3x3)

2	0	2
0	1	0
-1	1	0

=

Output Array (4x4)

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

Padding (2/2)

- If one wants the ***output size*** to be the same as the ***input size***, the input is padded with zeros
- In practice, we will always suppose we use padding in the following

Input Array (6x6)

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

Padded Input Array (8x8)

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

*

Kernel Array (3x3)

2	0	2
0	1	0
-1	1	0

=

Output Array (6x6)

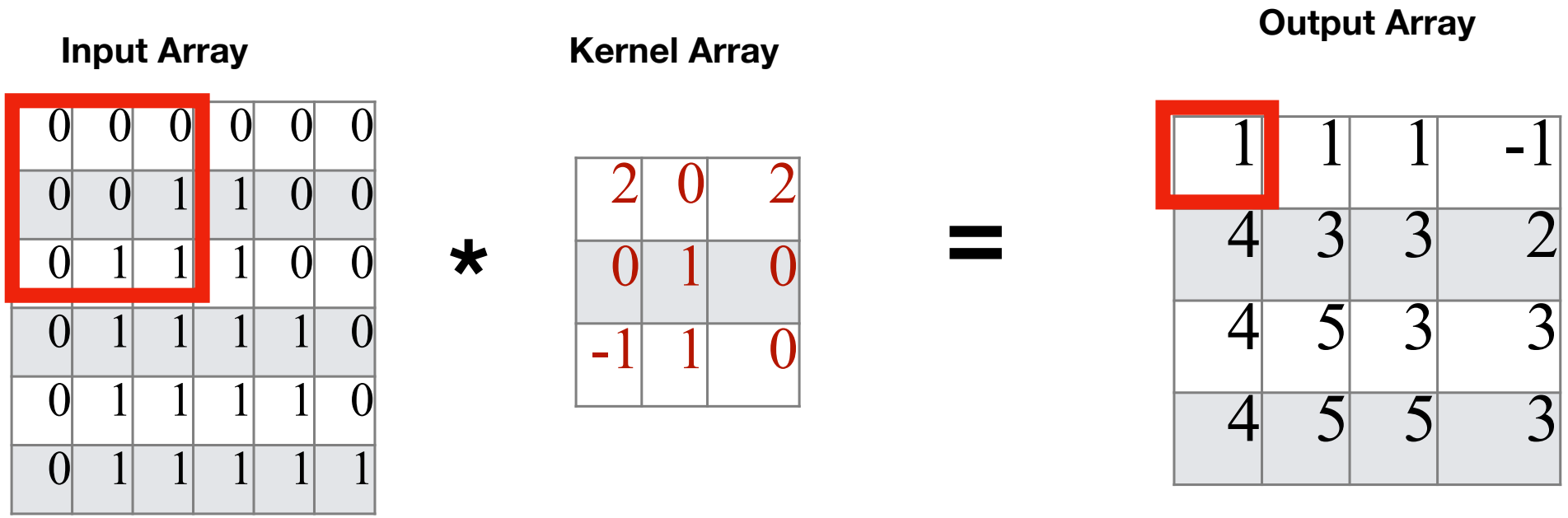
0	0	1	0	-1	0
0	1	1	1	-1	0
0	4	3	3	2	-1
2	4	5	3	3	-1
2	4	5	5	3	2
2	3	5	5	3	3

Convolution

- This convolution operation is interesting for two reasons:
  - 1- It allows an efficient representation of convolutional layers
  - 2- It is connected to image filters called “edge detectors”

Convolution Operation and Convolutional Layers (1/3)

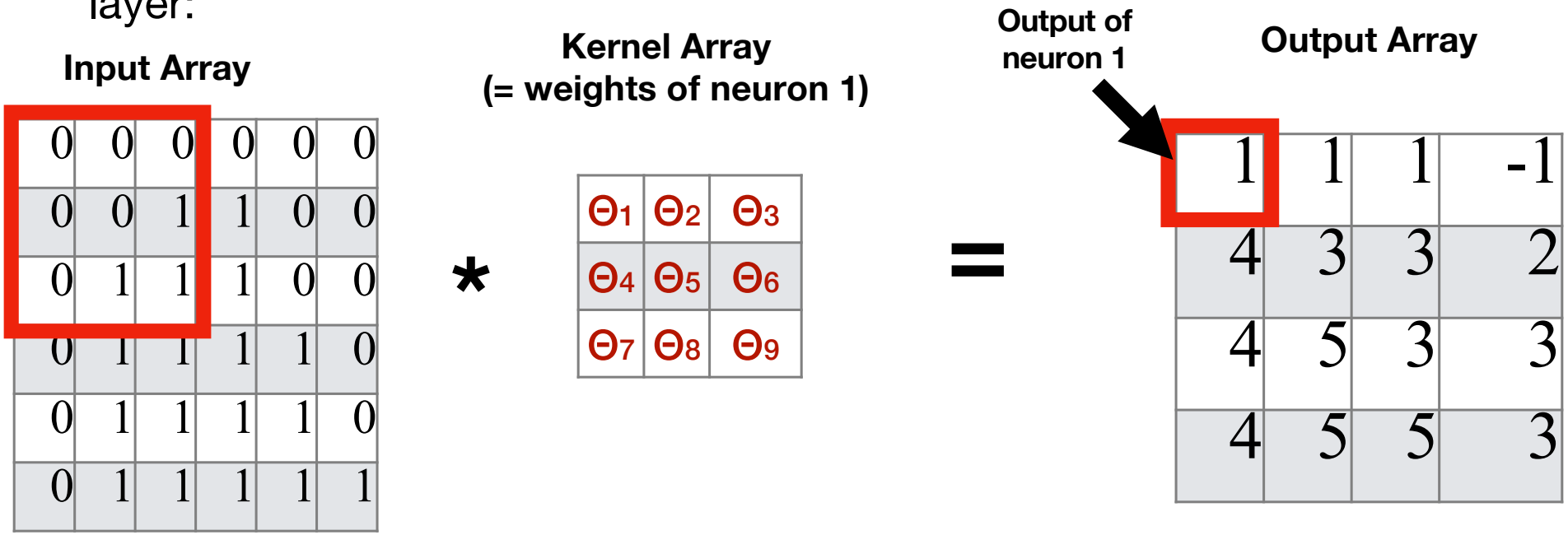
- Let us look again at our convolution operation



0x2 + 0x0 + 0x2 + 0x0 + 0x1 + 1x0 + 0x-1 + 1x1 + 1x0 = 1

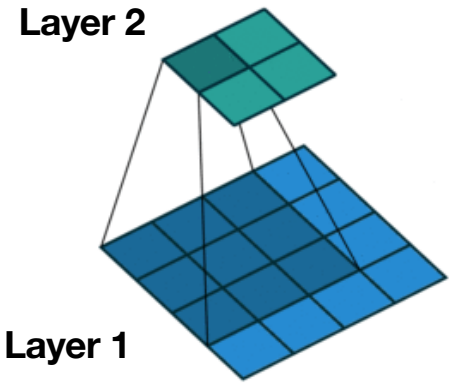
Convolution Operation and Convolutional Layers (2/3)

- If the kernel corresponds to the weights of a neuron in a convolution layer:



- The convolution operation actually computes the output of this convolution layer!

Convolution Operation and Convolutional Layers (3/3)



- The convolution operation actually computes the output of this convolution layer!

Input Array

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

*

Kernel Array (= weights of neuron 1)

Θ_1	Θ_2	Θ_3
Θ_4	Θ_5	Θ_6
Θ_7	Θ_8	Θ_9

=

Output of neuron 1

Output Array

1	1
4	3

- Note that we have <u>one additional parameter</u>: the *bias* θ_0
- We have therefore 9 + 1 = 10 parameters as we computed previously for this case

Padding

- Just a visualization of what happens with padding
  - It is as if the neurons on the edge of the output layer are connected to four inputs instead of nine

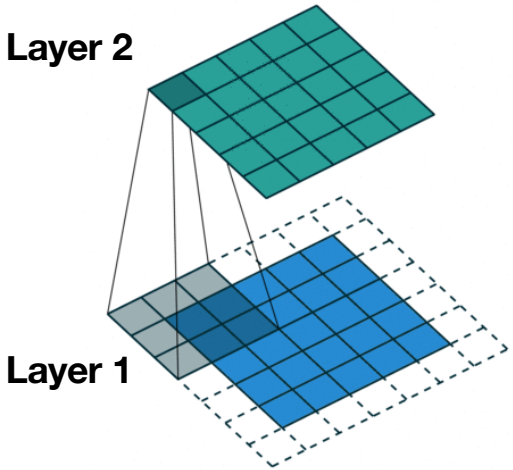


Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolution

- This convolution operation is interesting for two reasons:
  - 1- It allows an efficient representation of convolutional layers
  - 2- It is connected to image filters called “edge detectors”

Edge Detectors

- Edge detectors are simple image processing operations that were performed even prior to the use of convolutional layers in Neural Networks

- They can also be expressed as a convolution operation on an image

Convolutions as “Edge Detectors”

- Intuitively, this kernel “detect” vertical edges

Input Array

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

*

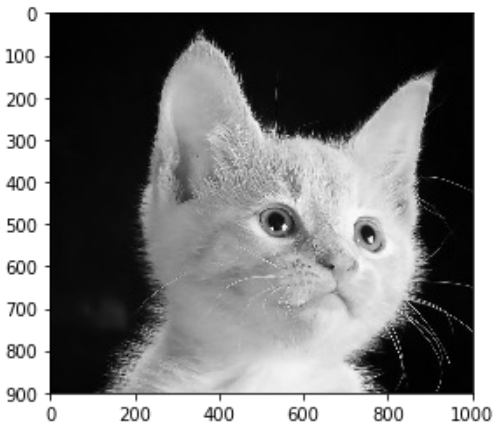
Kernel Array

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

=

2	1	-2	-2
3	1	-2	-3
3	0	-1	-3
3	0	0	-2

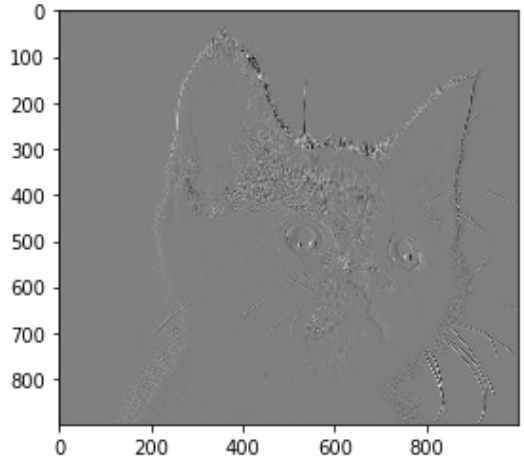
Edge Detectors

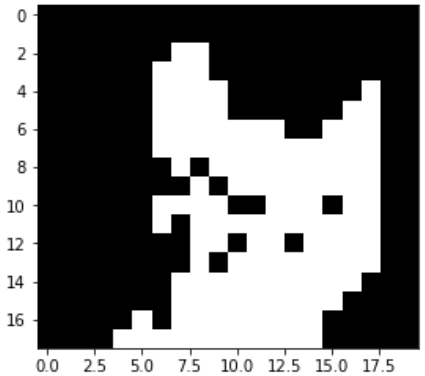


*

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

=

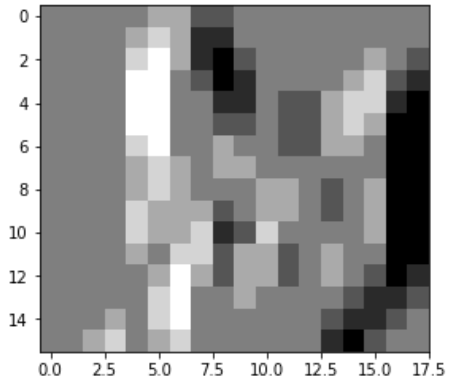




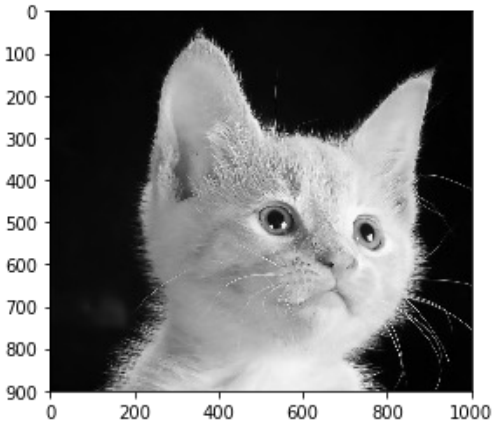
*

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

=



Learnable Kernels and Edge Detectors



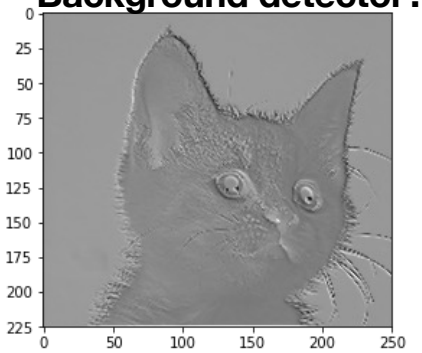
*

In practice, we will be learning these parameters from examples:

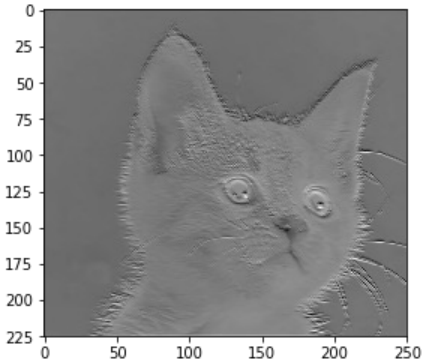
Θ_1	Θ_2	Θ_3
Θ_4	Θ_5	Θ_6
Θ_7	Θ_8	Θ_9

=

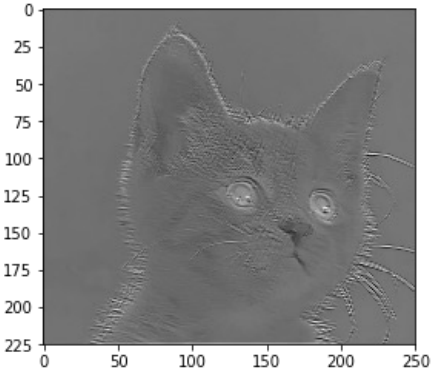
Background detector?



Fur detector?



Eyes detector?



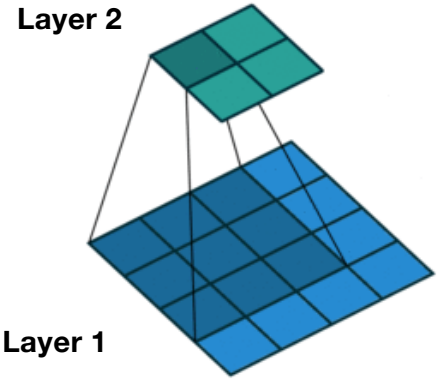
- We know some kernels can emphasize some edges in an image (edge detectors)
- Maybe by training the parameters, we discover kernels that can emphasize interesting aspects of the image?

Size of Kernels

- We only considered convolutions with kernels of size 3x3
  - Corresponds to neurons having 9 inputs
- We can use smaller and larger kernels
- 3x3 is the most common
- 5x5 and 7x7 are also often used
- The convolution operation remains the same

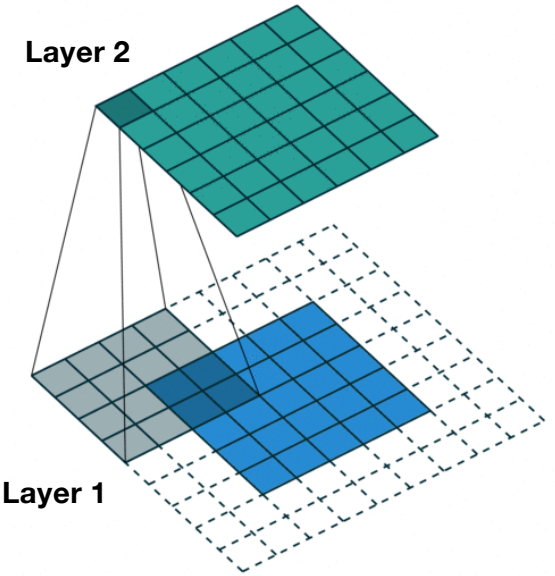
Kernel Array
(= weights of neuron)

Θ_1	Θ_2	Θ_3
Θ_4	Θ_5	Θ_6
Θ_7	Θ_8	Θ_9



Kernel Array
(= weights of neuron)

Θ_1	Θ_2	Θ_3	Θ_4
Θ_5	Θ_6	Θ_7	Θ_8
Θ_9	Θ_{10}	Θ_{11}	Θ_{12}
Θ_{13}	Θ_{14}	Θ_{15}	Θ_{16}



Mathematic and Neurons

- We have now seen the two mathematic operations most used in Neural Networks:
- ***Matrix Multiplication*** represents the computation of a Fully Connected Layer of Neurons
- ***Convolution*** represents the computation of a Convolutional Layer
- The only other computations that *usually* happen in a Neural Network are so-called "*activations operations*" like the ***sigmoid*** function

Summary

- Convolutional Layers are good for image processing because:
  - They can have much less parameters than Fully Connected Layers (thank to having less input connections and to weight sharing)
  - They implicitly represent some spatial information (neighbouring pixels are connected to the same neurons; weights of neurons are invariant by translation)
  - They can represent useful image filters like Edge Detectors
- Next time: How to use them to build a Neural Network that can classify images

Exercise

Input Array

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

*

Kernel Array

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

=

Output Array

Report

- Submit the report of the exercise **in pdf** via PandA
  - Submission due: **next lecture**
  - Name the pdf file as **student id_name**