

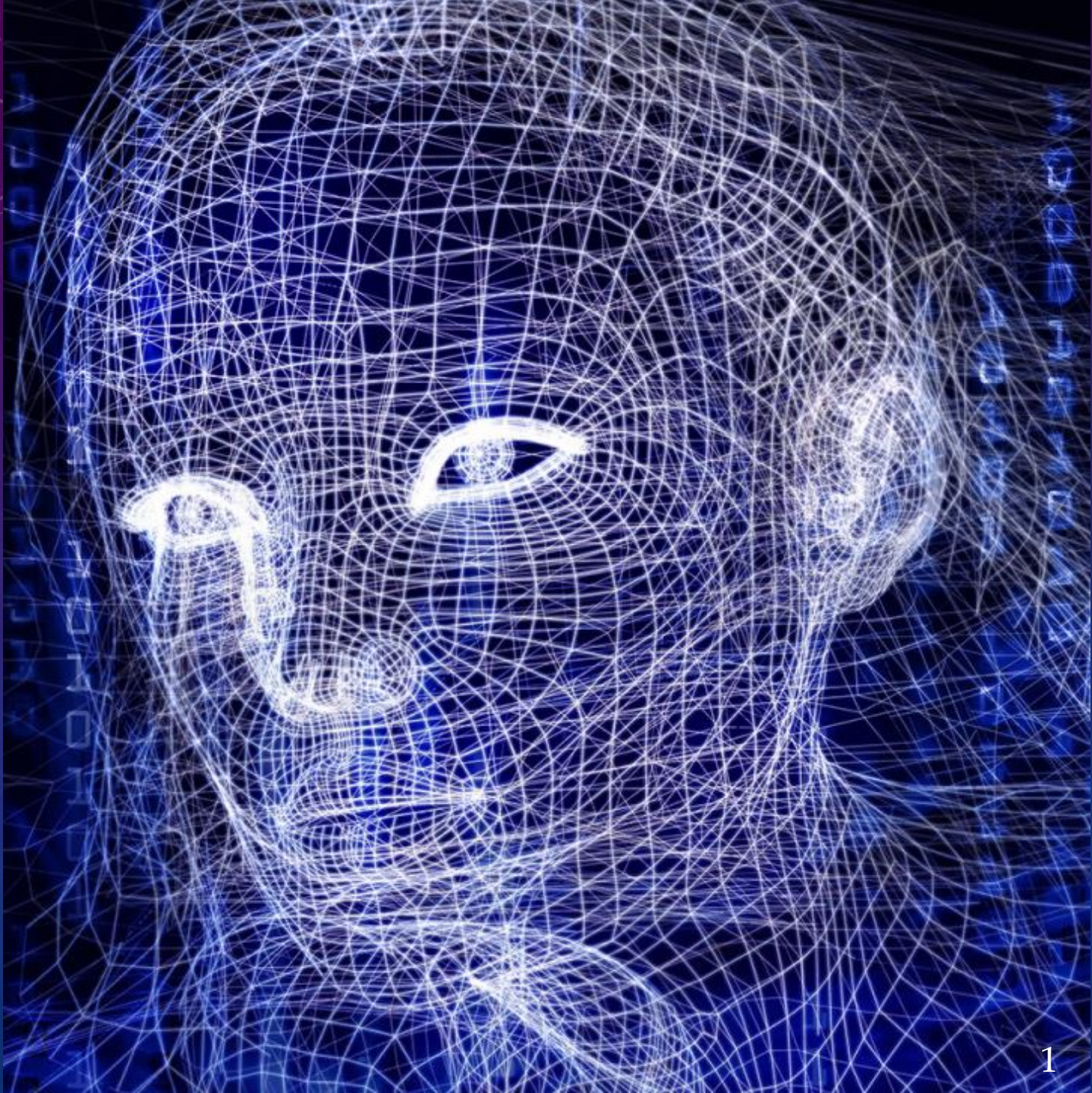
*LECTURE SERIES FOR DIGITAL  
SURVEILLANCE SYSTEMS AND  
APPLICATION*

*Chapter 2*  
*Introduction to*  
*Neural Network*

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of  
Science and Technology





# Neural Networks

- 1) Perceptron and Multi Layer Perceptron (MLP)
- 2) Feedforward Network
- 3) Activation Functions
- 4) Gradient Descent for Estimating Network Parameters

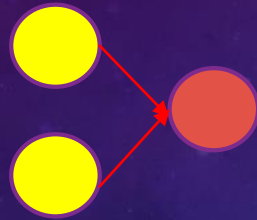
# What is Neural Network?



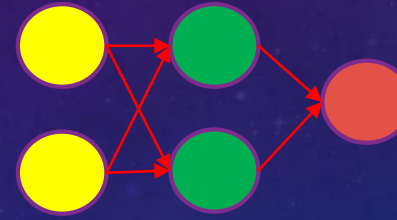
<https://www.youtube.com/watch?v=aircAruvnKk> [19:13]

# Neural Network — Feedforward / MLP

Perceptron (P)



Feed Forward(FF)

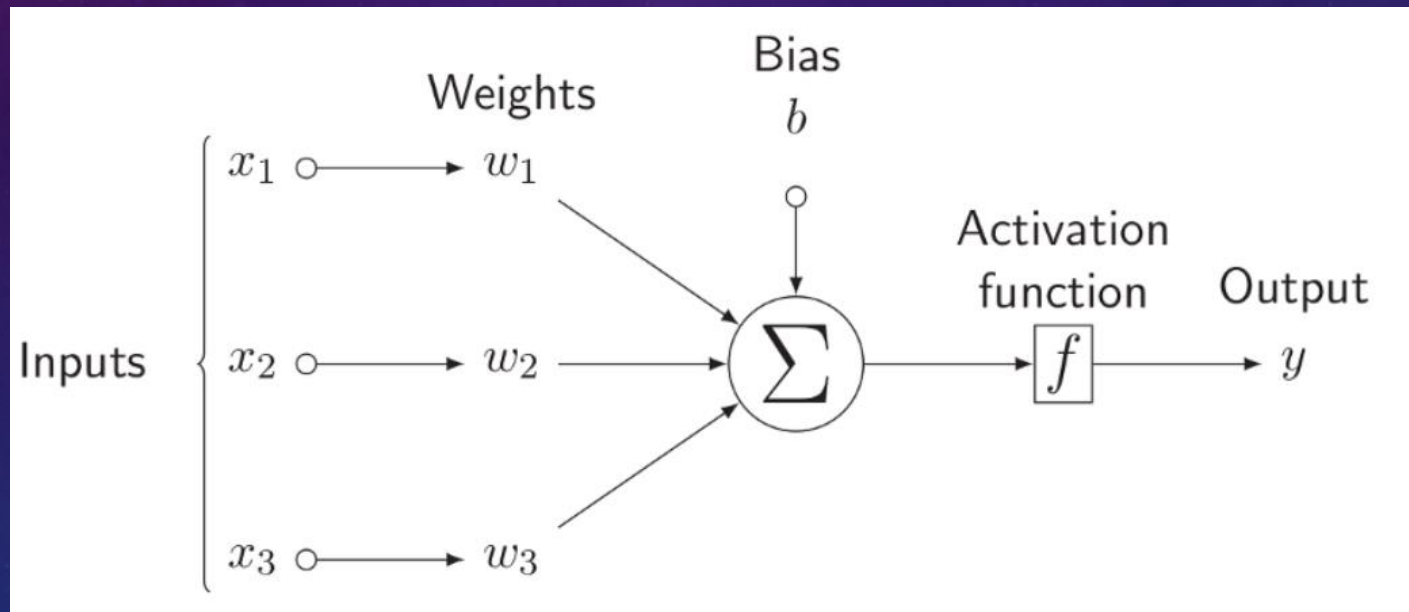


Yellow : Input layer  
Green : Hidden layer  
Orange : Output layer

<https://medium.com/biaslyai/pytorch-introduction-to-neural-network-feedforward-neural-network-model-e7231cff47cb>

# Single-Layer Perceptron

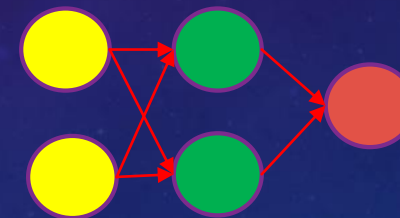
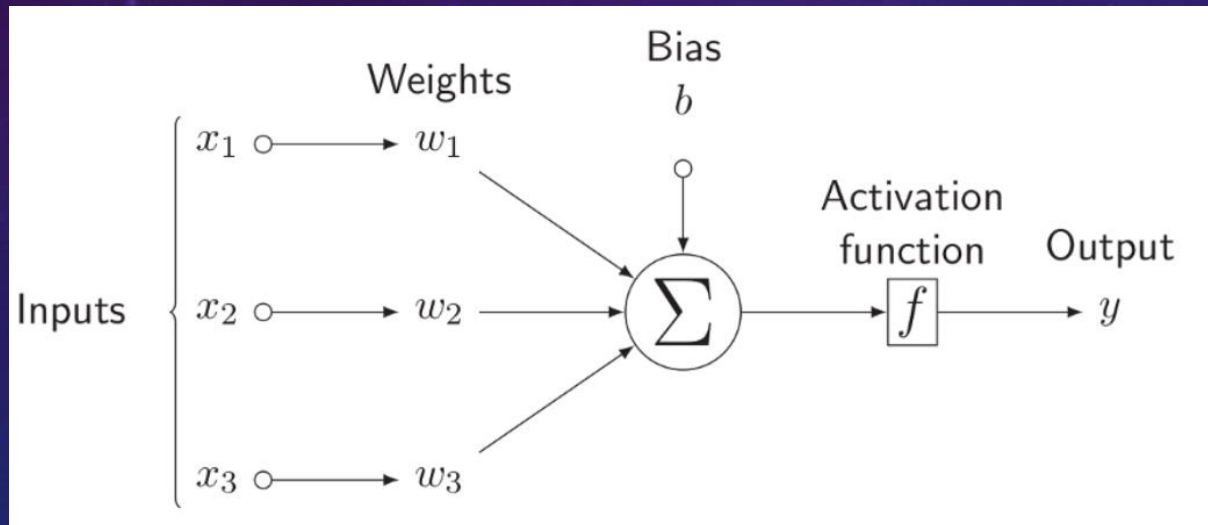
Single-layer perceptron takes data as input and its weights are summed up then an activation function is applied before sent to the output layer.



<https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network>

# Activation Function

- The activation functions in the neural network introduce the non-linearity to the linear output.
- It defines the output of a layer, given data, meaning it sets the threshold for making the decision of whether to pass the information or not.

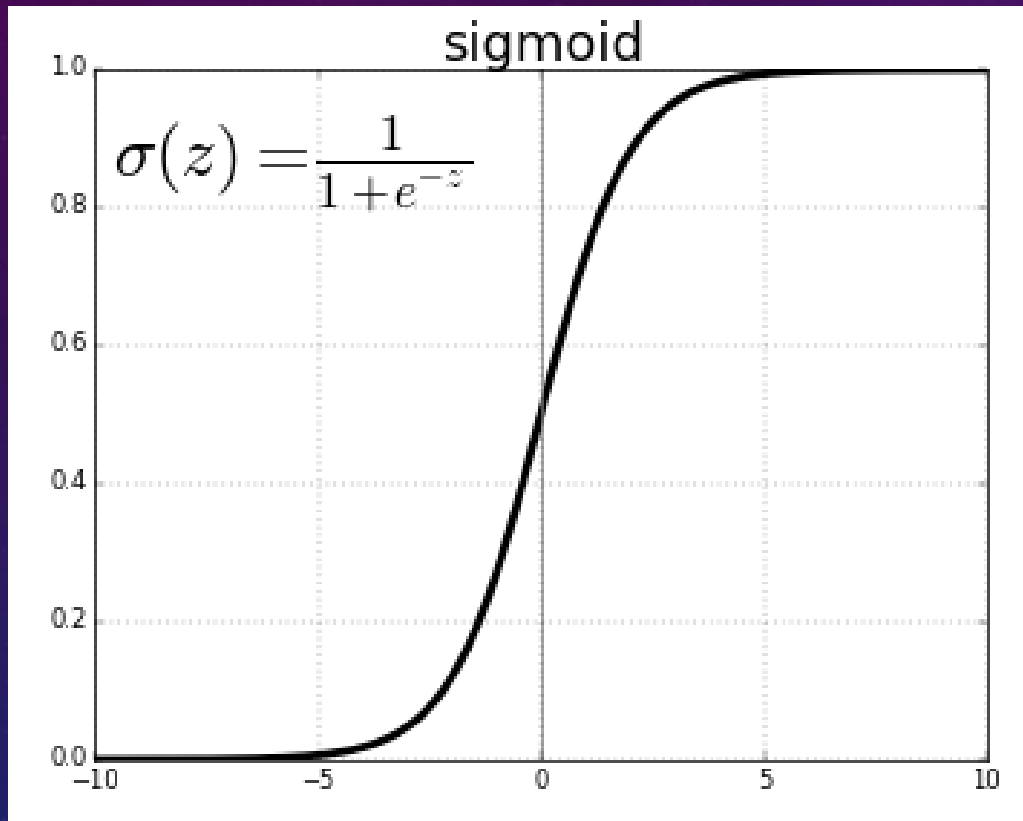


Yellow : Input layer  
Green : Hidden layer  
Orange : Output layer

<https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network>

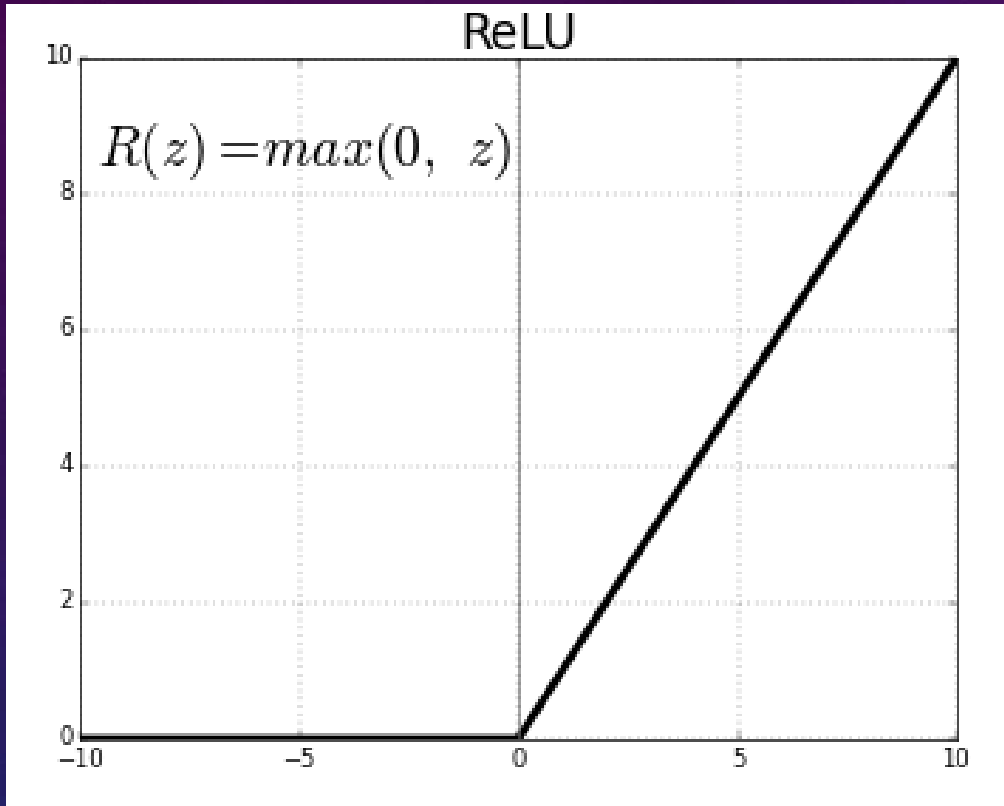


# Sigmoid Activation Function



- In order to map predicted values to probabilities, we use the Sigmoid function.
- The function maps any real value into another value between 0 and 1.

# ReLU (Rectified Linear Unit) Activation Function

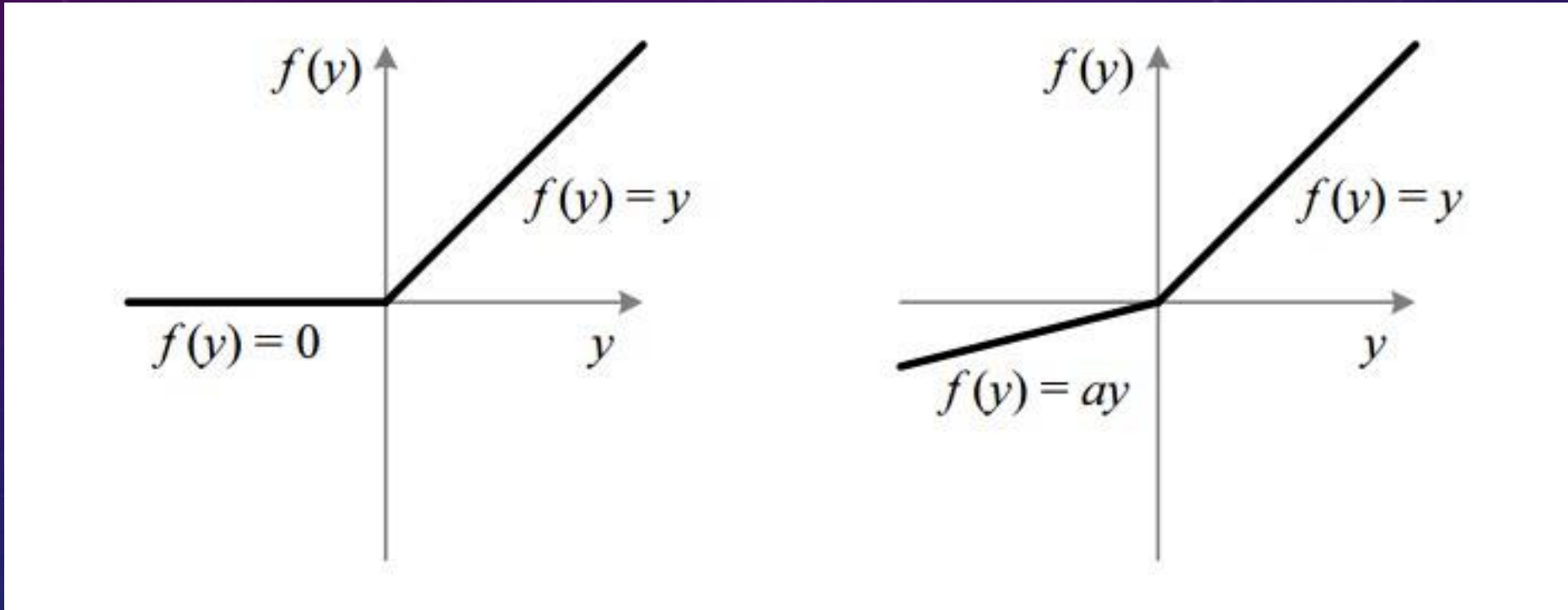


- The ReLU is half rectified (from bottom).  $f(z)$  is zero when  $z$  is less than zero and  $f(z)$  is equal to  $z$  when  $z$  is above or equal to zero.
- Issue :
  - All the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly.
  - That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.



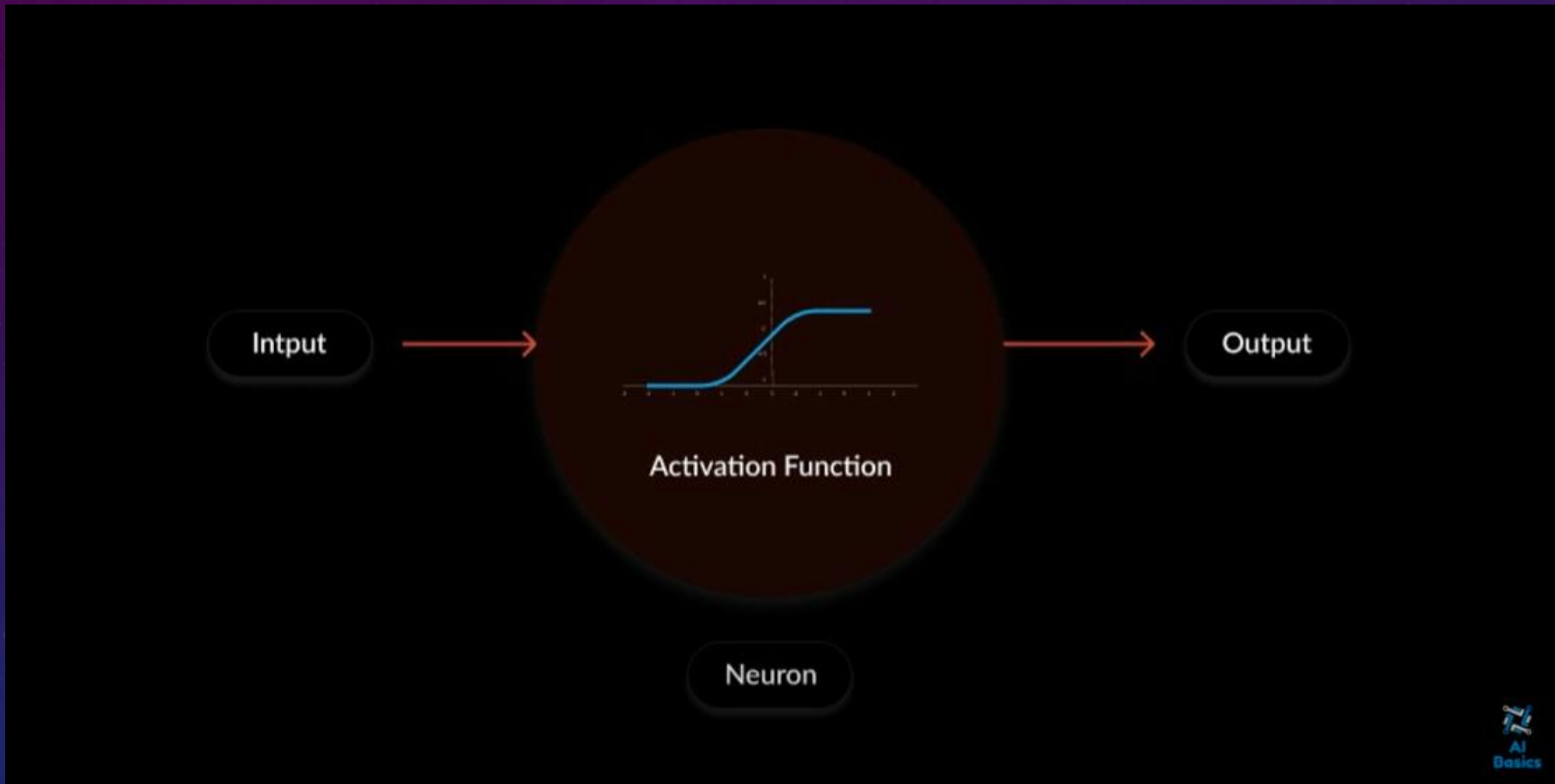
# Leaky ReLU Activation Function

It is an attempt to solve the dying ReLU problem



The leak helps to increase the range of the ReLU function.  
When  $a$  is not 0.01 then it is called Randomized ReLU.

# Which Activation Function Should I Use?



[https://www.youtube.com/watch?v=WK9P\\_s6hXIc&ab\\_channel=AIBasics](https://www.youtube.com/watch?v=WK9P_s6hXIc&ab_channel=AIBasics) [5:09]

# Which Activation Function Should I Use?



<https://www.youtube.com/watch?v=-7scQpJT7uo> [8:58]



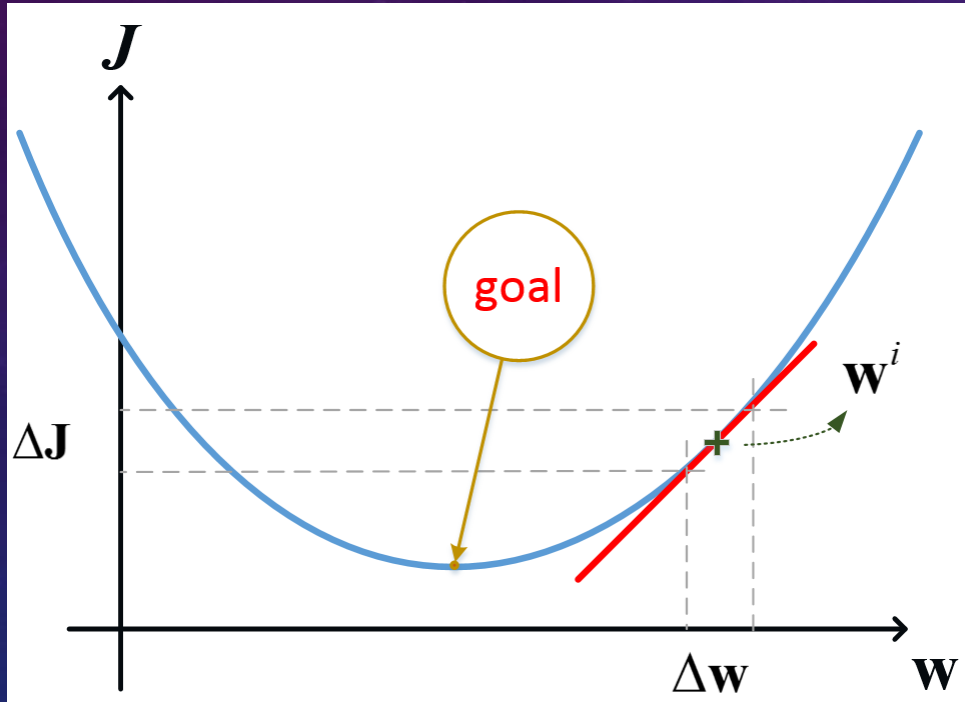
# Gradient Descent



<https://www.youtube.com/watch?v=IHZwWFHWa-w> [21:00]

# Gradient Descent

- Idea : update  $w$  with the data  $(t, y)$
- The update rule :



$$w^{i+1} = w^i - \alpha \left( \frac{\partial J}{\partial w^i} \right)$$

where  $\alpha$  is the learning rate

$$J = (\vec{t} - \vec{y})^T (\vec{t} - \vec{y})$$

$$\Rightarrow \frac{\partial J}{\partial w} = -(\vec{t} - \vec{y})^T \frac{\partial y}{\partial w}$$

# What is Backpropagation Really Doing?



<https://www.youtube.com/watch?v=Ilg3gGewQ5U> [13:53]



# Example 2.1 Build A Neural Network

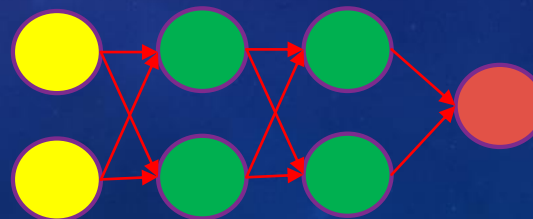
Please download [2-1\\_log\\_regression\\_visualize.zip](#) and unzip it.

- Given an architecture with input size=2, hidden layer=2 and output size=1.
- Each layer is followed by ReLU activation function.
- In this case, we can know how to use “Autograd” and the computed loss to derive the gradient; and how to establish a neural network in Pytorch.

Step:

- Autograd
- Build A Neural Network

```
Feedforward(  
  (input): Linear(in_features=2, out_features=2, bias=True)  
  (relu1): ReLU()  
  (fc1): Linear(in_features=2, out_features=2, bias=True)  
  (relu2): ReLU()  
  (fc2): Linear(in_features=2, out_features=2, bias=True)  
  (relu3): ReLU()  
  (output): Linear(in_features=2, out_features=1, bias=True)  
)
```



Yellow : Input layer  
Green : Hidden layer  
Orange : Output layer

# Example 2.2 Activation Function

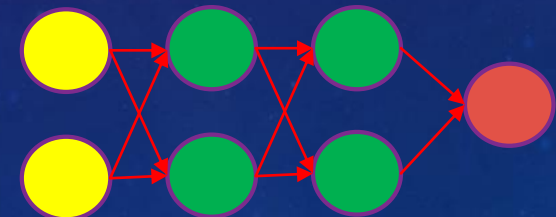
Please download [2-2\\_Activation\\_Function.zip](#) and unzip it.

- Given an architecture with input size=2, hidden layer=2 and output size=1.
- Each layer is followed by the Leaky ReLU activation function.
- For more activation functions, please refer to the following link:

<https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity>

- In this case, we can how to use the activation function.

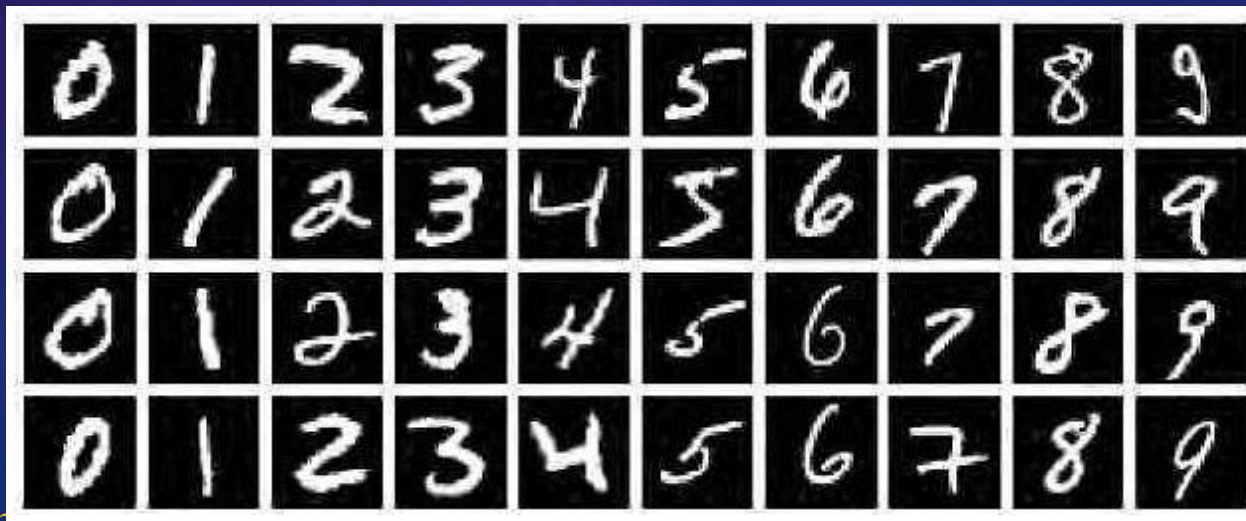
```
Feedforward(  
  (input): Linear(in_features=2, out_features=2, bias=True)  
  (Leakyrelu1): LeakyReLU(negative_slope=0.01)  
  (fc1): Linear(in_features=2, out_features=2, bias=True)  
  (Leakyrelu2): LeakyReLU(negative_slope=0.01)  
  (fc2): Linear(in_features=2, out_features=2, bias=True)  
  (Leakyrelu3): LeakyReLU(negative_slope=0.01)  
  (output): Linear(in_features=2, out_features=1, bias=True)  
)
```



# Example 2.3 Train the MLP Model

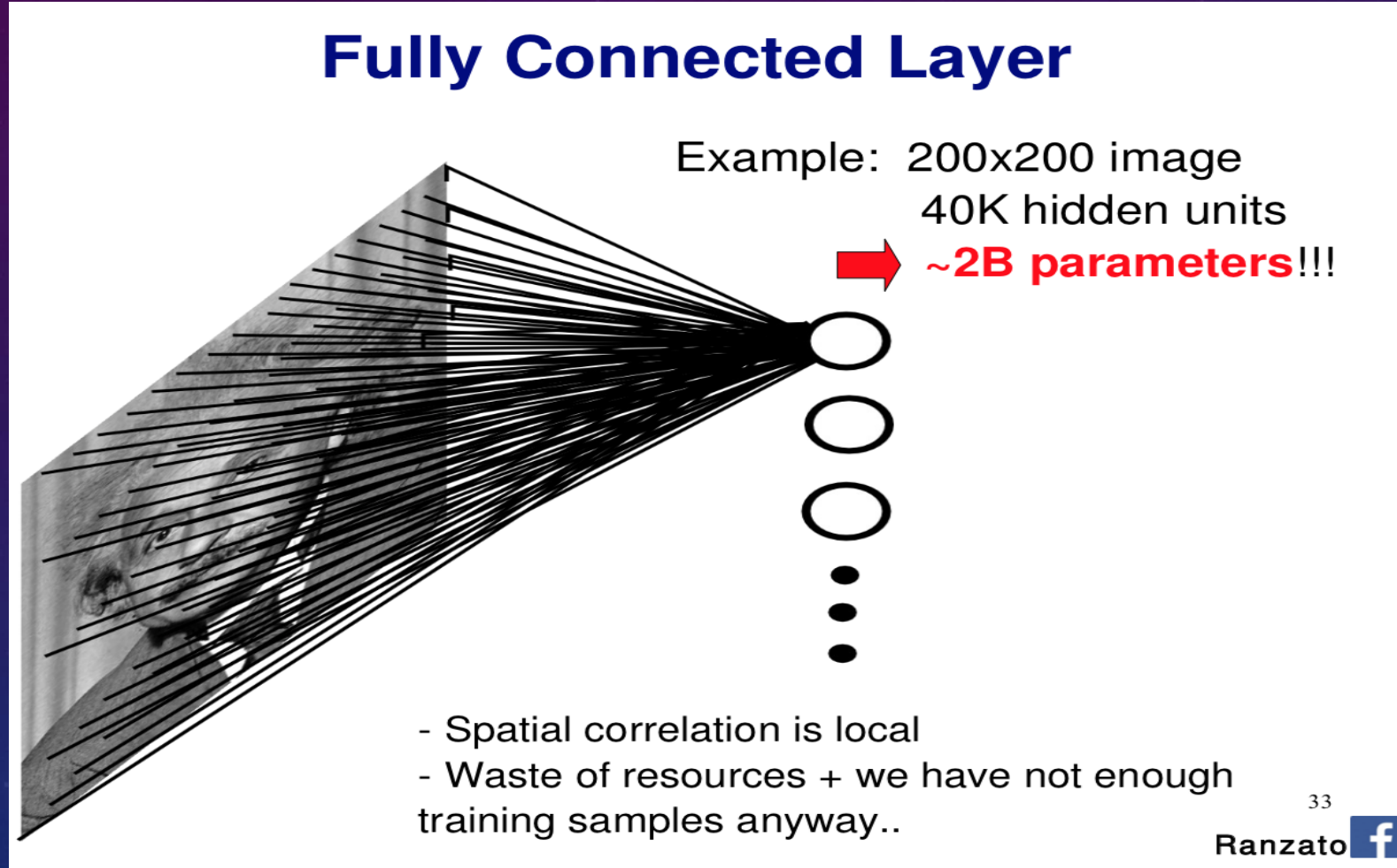
Please download [2-3\\_MLP\\_MNIST.zip](#) and unzip it.

- Given an architecture with input size=28x28, hidden layer=3 and output size=10.
- The 1<sup>st</sup> hidden layer includes 500 neurons, the 2<sup>nd</sup> layer includes 250 neurons and the 3<sup>rd</sup> layer includes 125 neurons.
- Each layer is followed by the ReLU activation function.
- Use MNIST dataset as the input data: 90% for training and 10% for testing.
- Please follow the instruction to train a MLP base handwritten digits classifier.

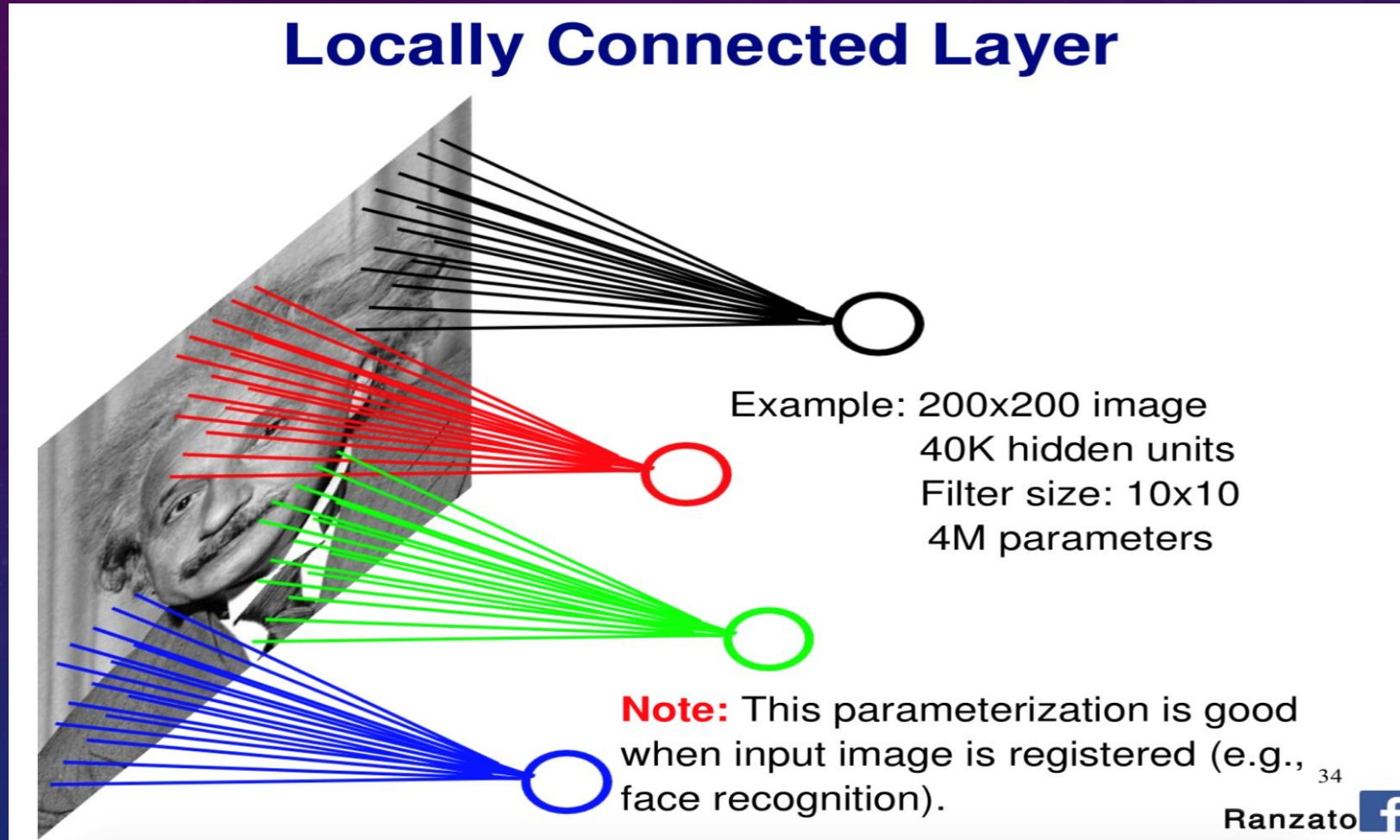




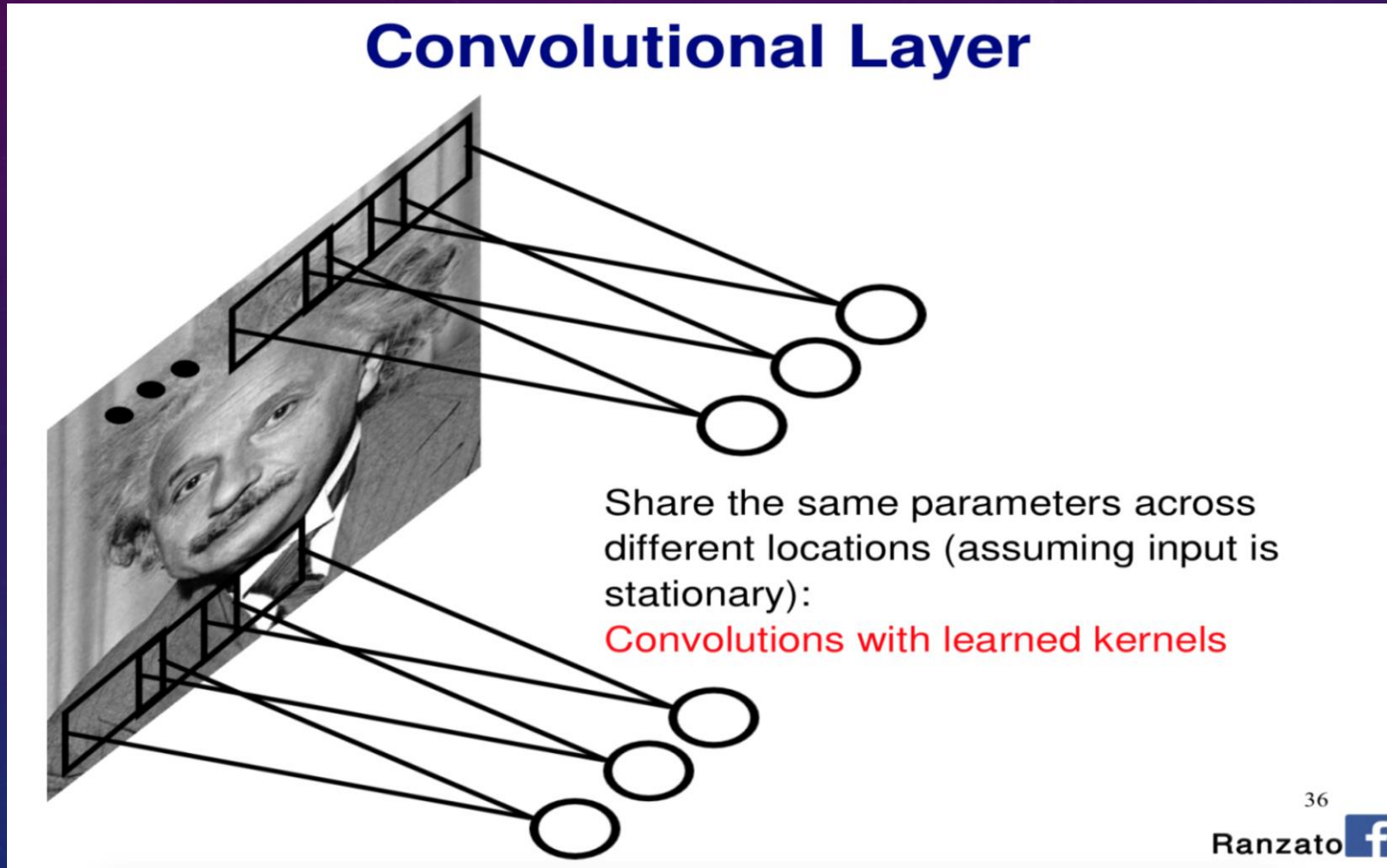
# Fully Connected Layer



# Locally Connected Layer



# Convolutional Layer





# Convolutional Neural Networks

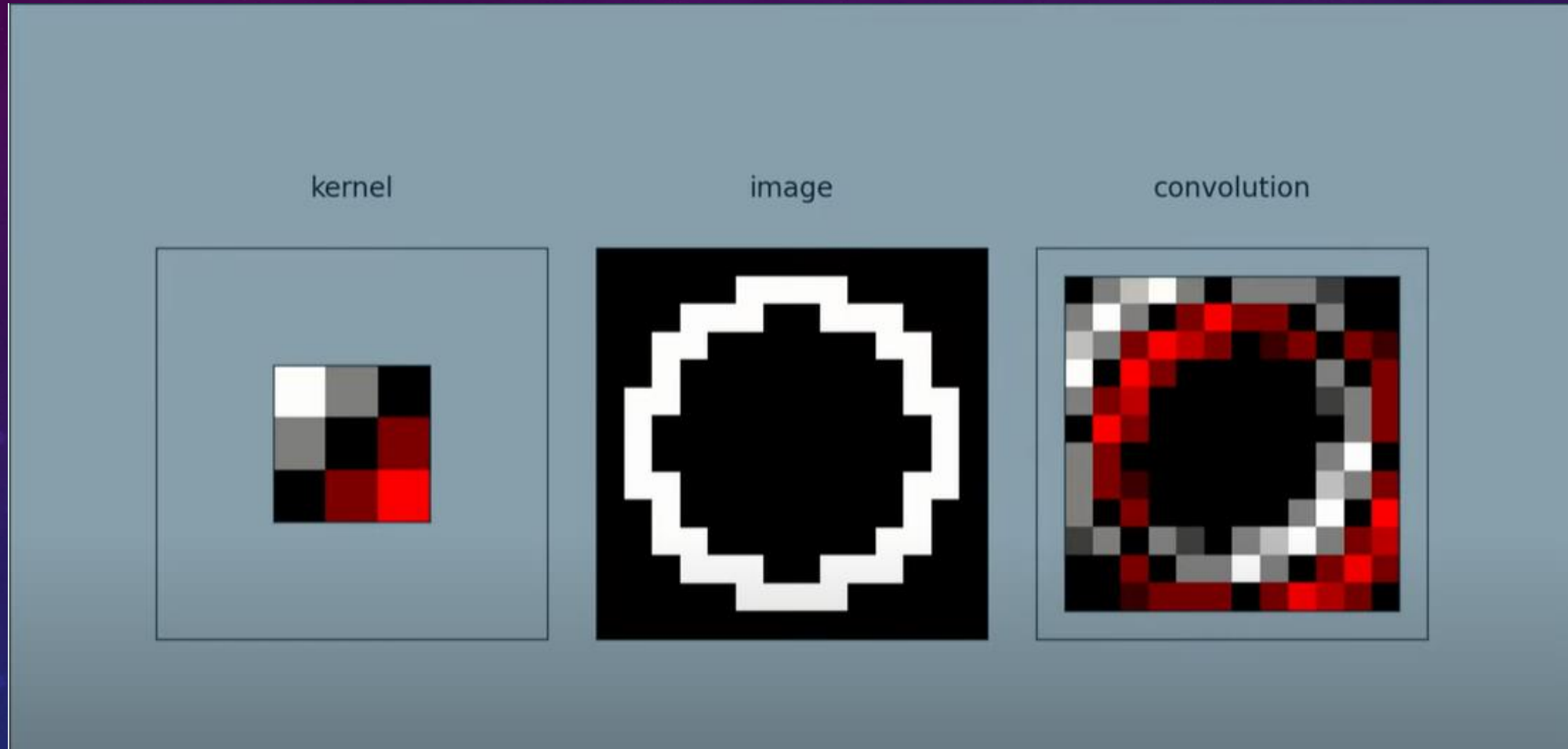
- 1) Convolution Filter
- 2) Why Convolution?
- 3) Pooling Layers
- 4) Loss Functions

# Convolutional Neural Networks (CNNs) Explained



[https://www.youtube.com/watch?v=YRhxdVk\\_sIs&t=62s](https://www.youtube.com/watch?v=YRhxdVk_sIs&t=62s) [8:36]

# 2D Convolution Operation

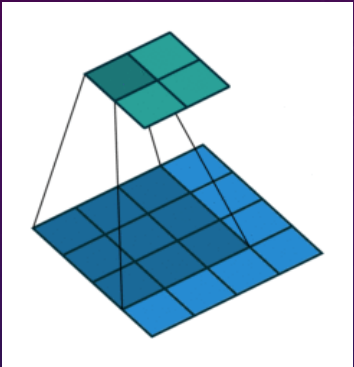
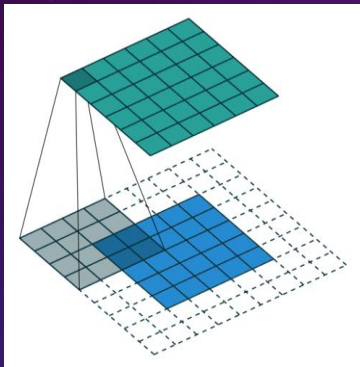
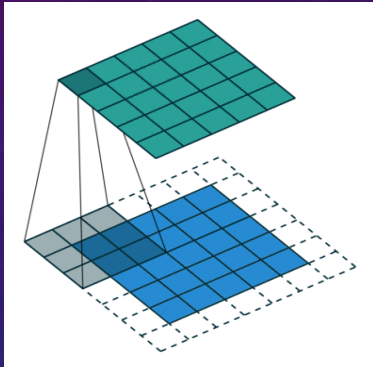
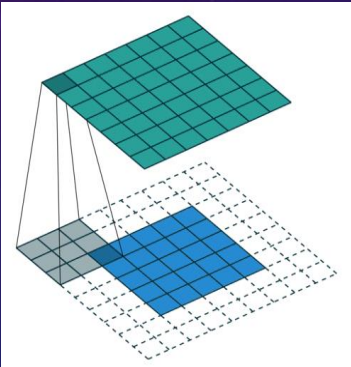
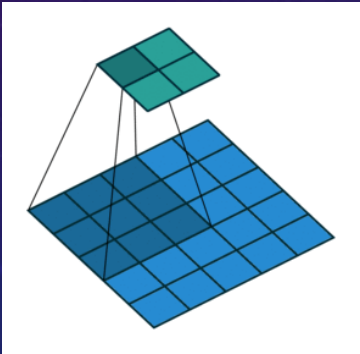
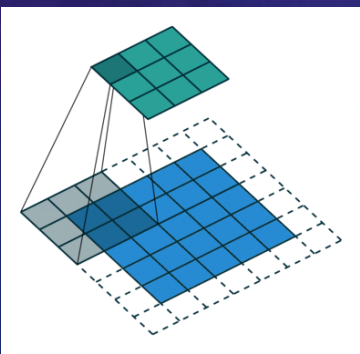
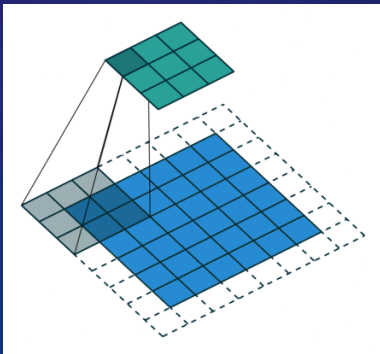


<https://www.youtube.com/watch?v=B-M5q51U8SM> [20:04]



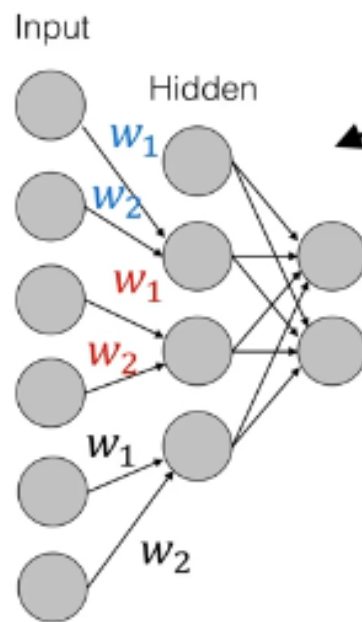
# Convolution Animations

*N.B.: Blue maps are inputs, and cyan maps are outputs.*

			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

# Why Convolution?

## Why convolution?

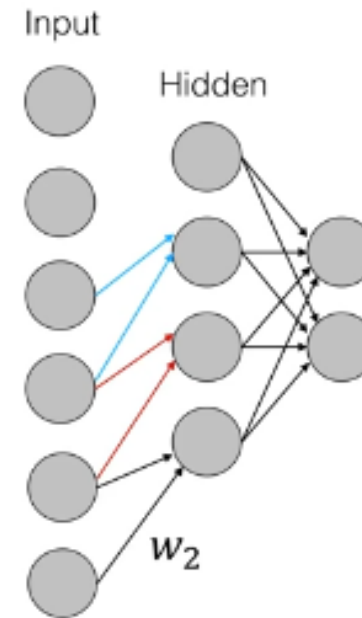


1-d convolution with

- filters: 1
- filter size: 2
- stride: 2

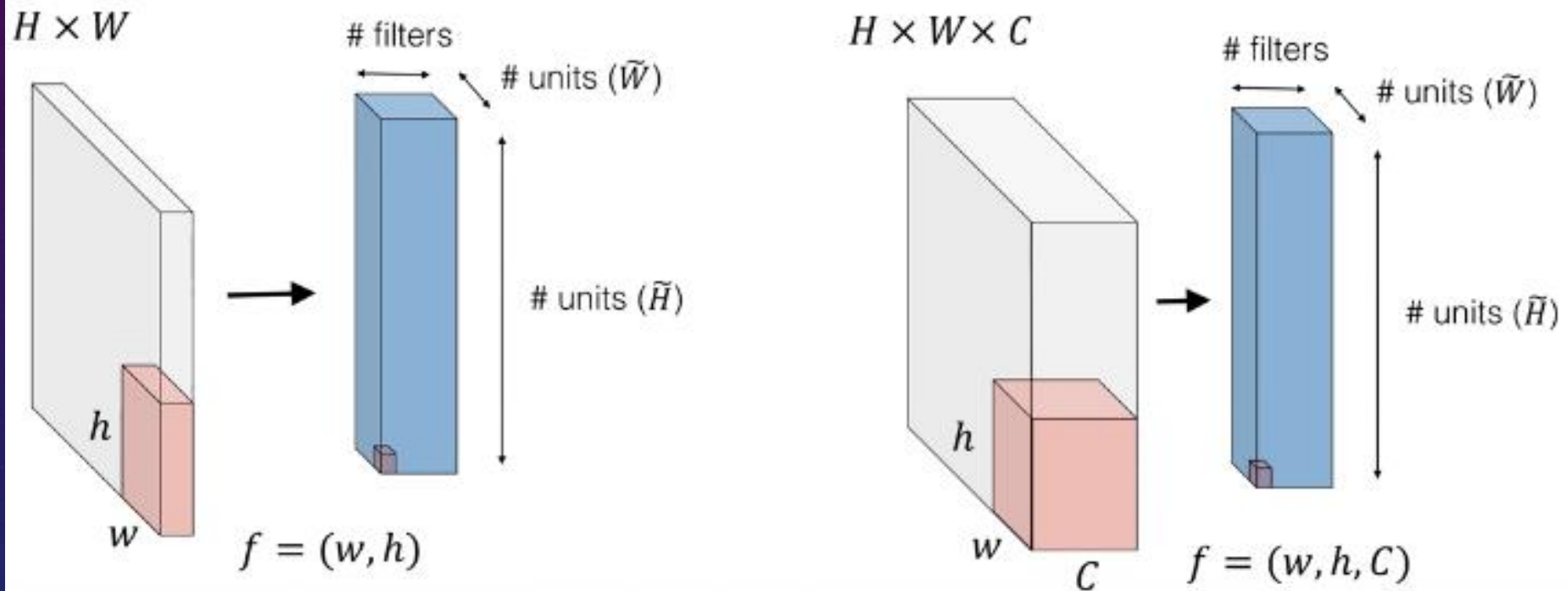
1-d convolution with

- filters: 1
- filter size: 2
- stride: **1**



# 2-D Convolution

## 2-D convolution

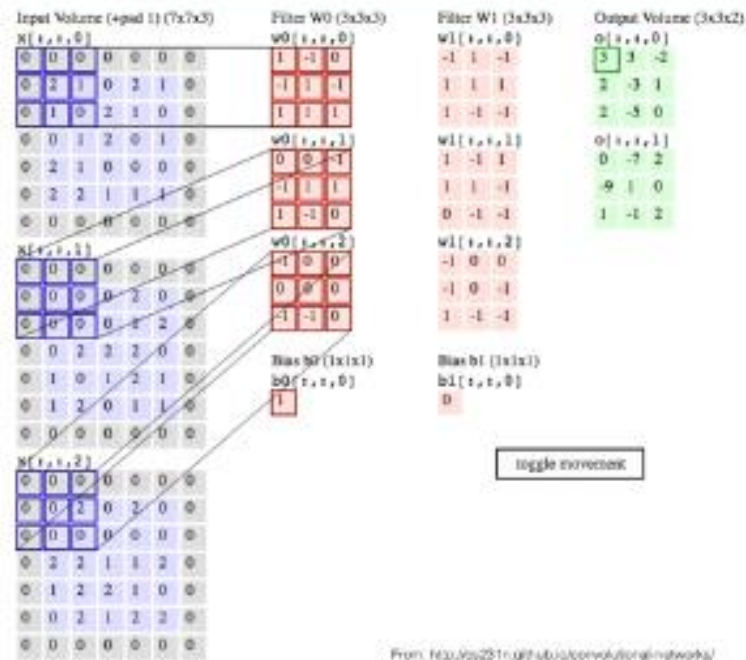


nervana

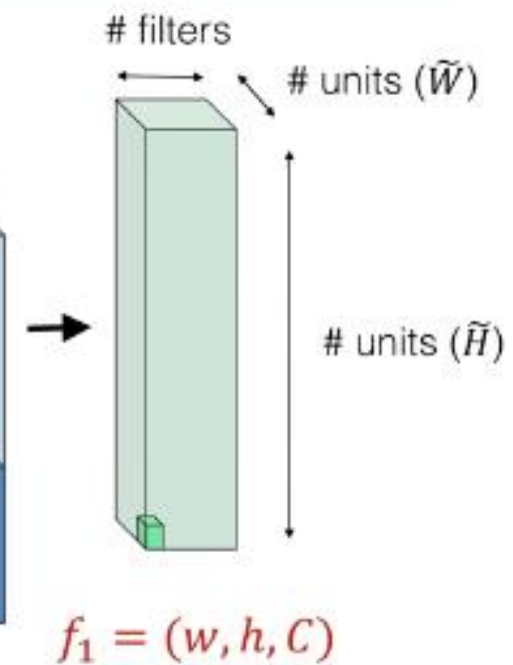
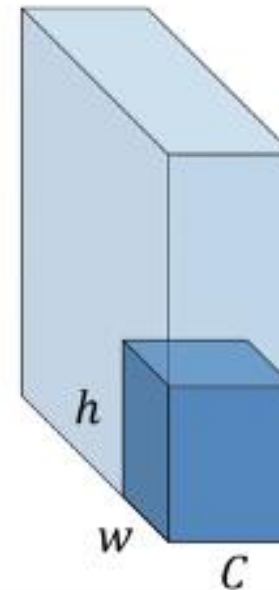


# 2-D Convolution

## 2-D Convolution



$H \times W \times C$



# Strided Convolutions – Andrew Ng



deeplearning.ai

Convolutional  
Neural Networks

---

Convolutions over  
volumes

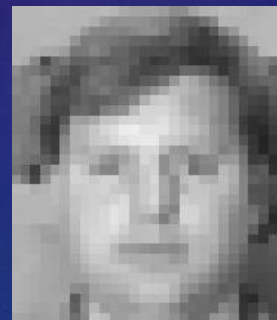
[https://www.youtube.com/watch?v=KTB\\_OFoAQcc](https://www.youtube.com/watch?v=KTB_OFoAQcc) [10:44]

# Example 2.4 Convolution Filter

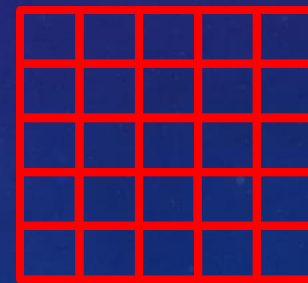
- Please download [2-4\\_Convolution\\_Example.zip](#) and unzip it.
- Given a face image  $I_{img}$ , please generate  $F_{conv}$ , a 5x5 convolution filter with random coefficients in uniform distribution
- Use “same zeros padding” with unit stride.
- Compute the output by convolving  $I_{img}$  with  $F_{conv}$  and calculate the information loss between original image and the one after convolving.

```
KERNAL_SIZE = 5
STRIDE = 1
PADDING = (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

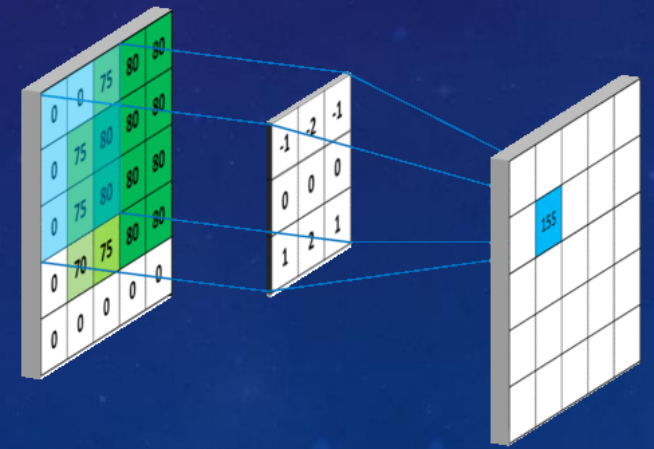
img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
Conv_Filter = np.random.rand(KERNAL_SIZE,KERNAL_SIZE)
#Conv_Filter =
np.random.normal(mean,std,(KERNAL_SIZE,KERNAL_SIZE))
#Normal distribution
Conv_Filter = Conv_Filter/np.sum(Conv_Filter)
img_F = img
```



Input Image



Conv kernel

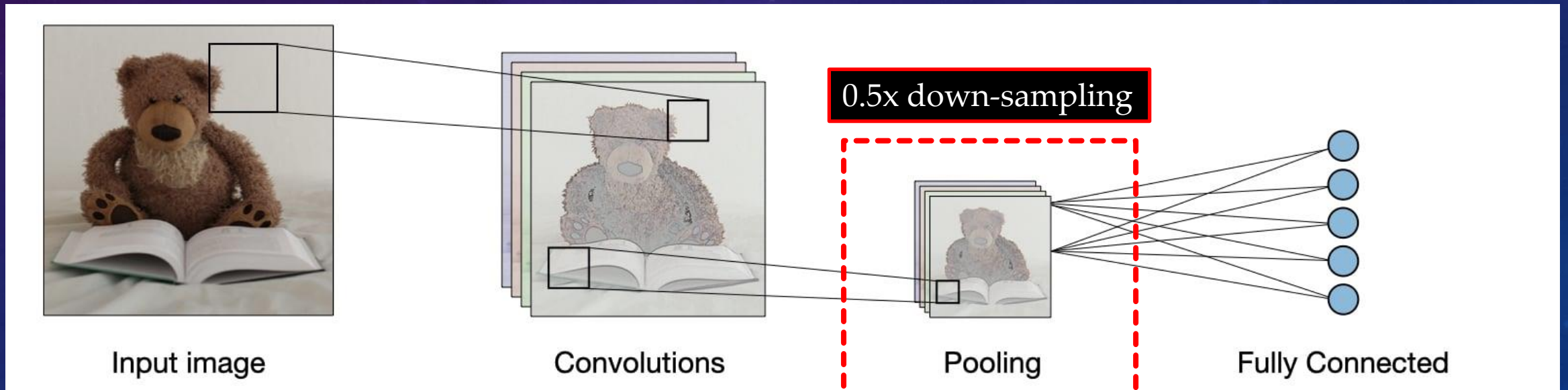
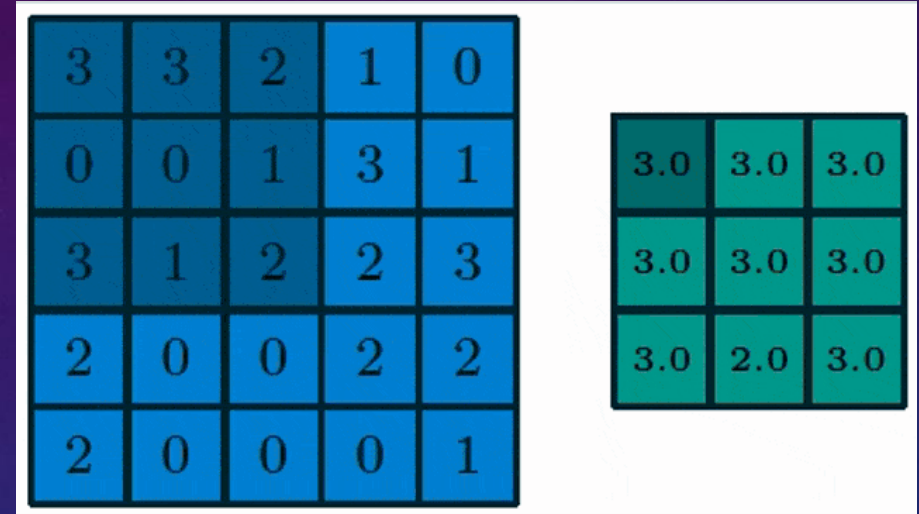




# Pooling Layer

Pooling is the operation of down-sampling input, two reasons that we need to use pooling:

- To reduce the input size.
- To achieve local translation and rotation invariance.



# Max Pooling in Convolutional Neural Networks



[https://www.youtube.com/watch?v=ZjM\\_XQa5s6s](https://www.youtube.com/watch?v=ZjM_XQa5s6s) [10:49]



# Pooling : Average Pooling

(1)

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

(2)

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

(3)

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

(4)

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

(5)

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

(6)

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

(7)

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

(8)

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

(9)

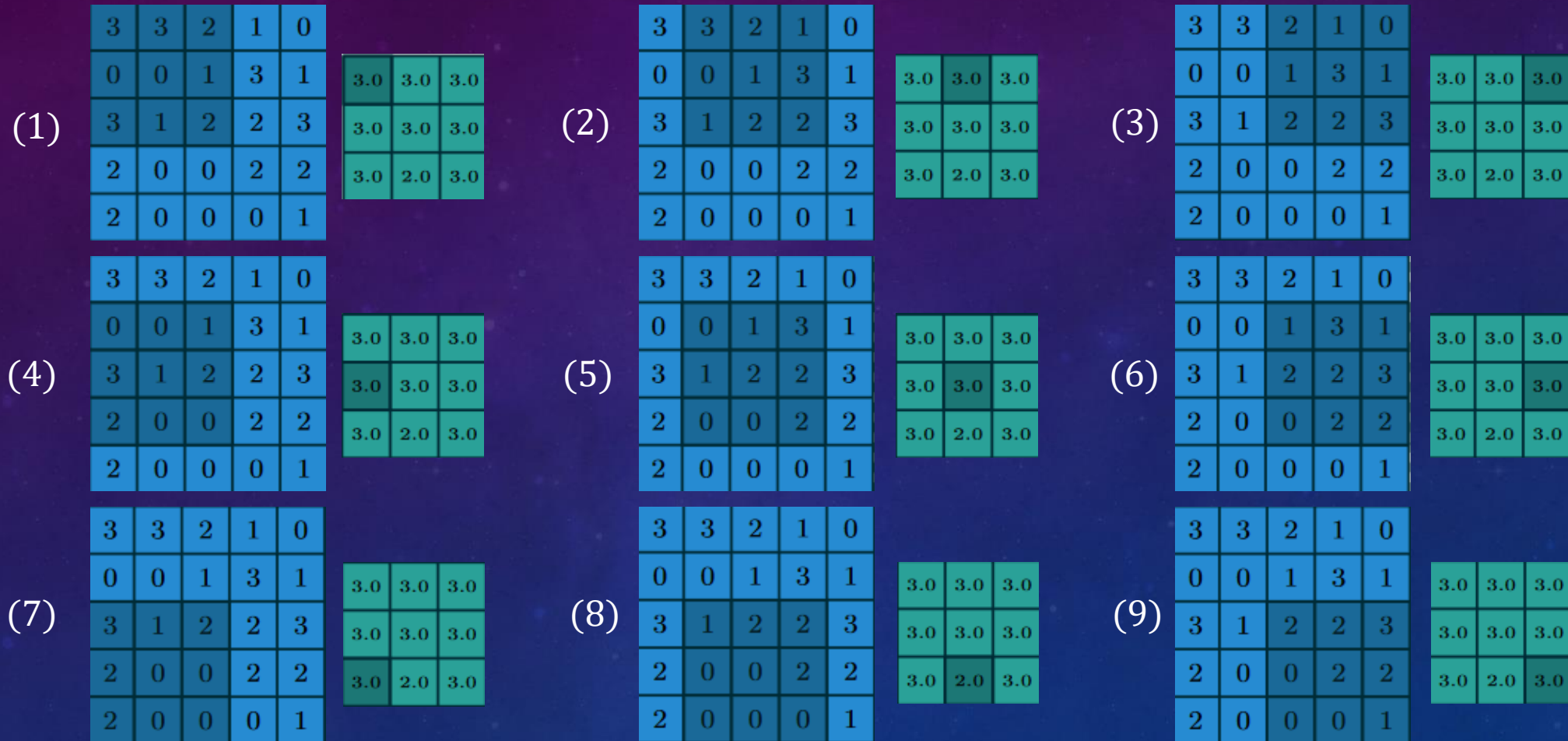
3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

Computing the output values of a 3×3 average pooling operation on a 5×5 input using 1×1 strides.



# Pooling : Max Pooling



Computing the output values of a 3×3 max pooling operation on a 5×5 input using 1×1 strides.

## Example 2.5 Max Pooling

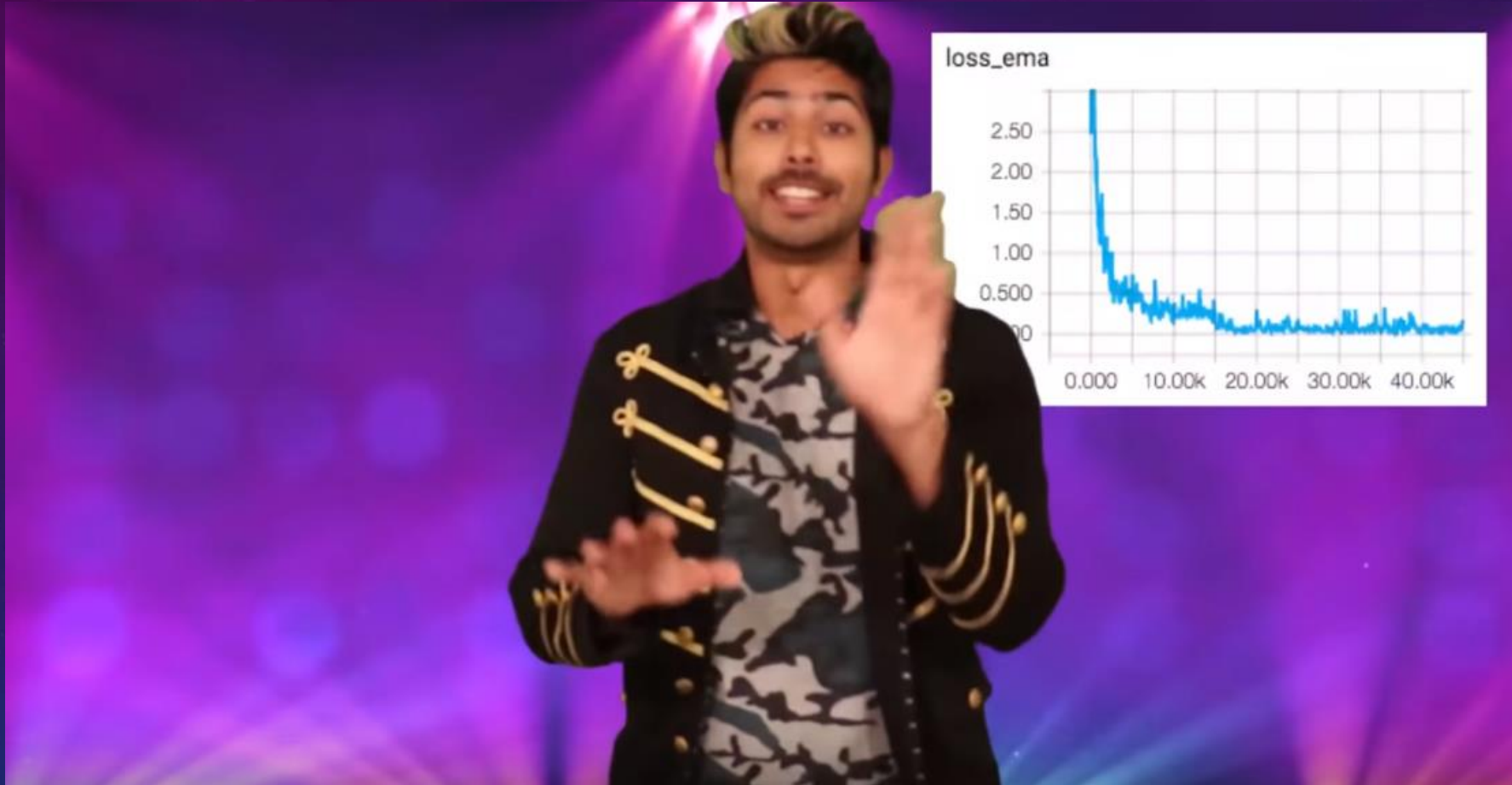
- Given a face image  $I_{img}$ , please generate  $F_{conv}$ , a 5x5 convolution filter with random coefficients in normal distribution (std = 0, mean=2), and use “same zeros padding” with unit stride. Use “max pooling” with 3x3 kernel size and unit stride. Please compute the output by convolving  $I_{img}$  with  $F_{conv}$  and calculate the information loss between original image and the one after convolving.



```
img_new = pooling(img_new, 3, 1, 'max')  
img_new = img_new.astype(np.uint8)
```

Use max pooling

# Loss Functions Explained

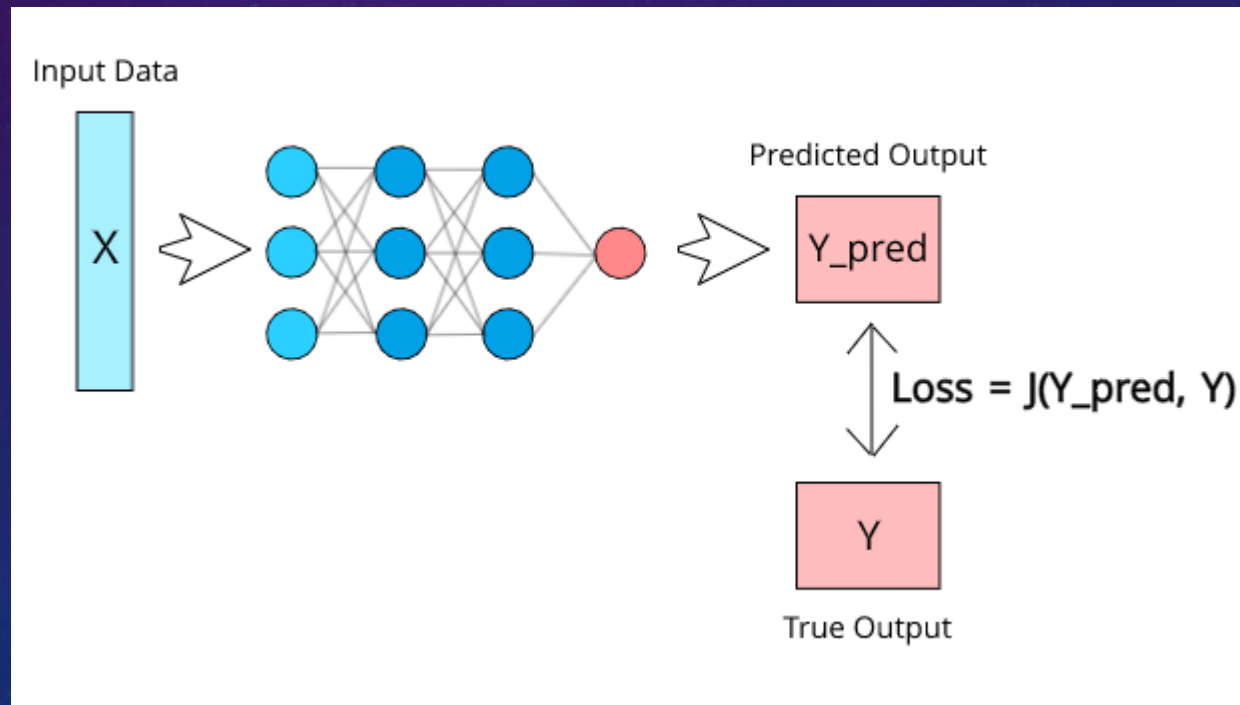


<https://www.youtube.com/watch?v=IVVVjBSk9N0> [12:55]



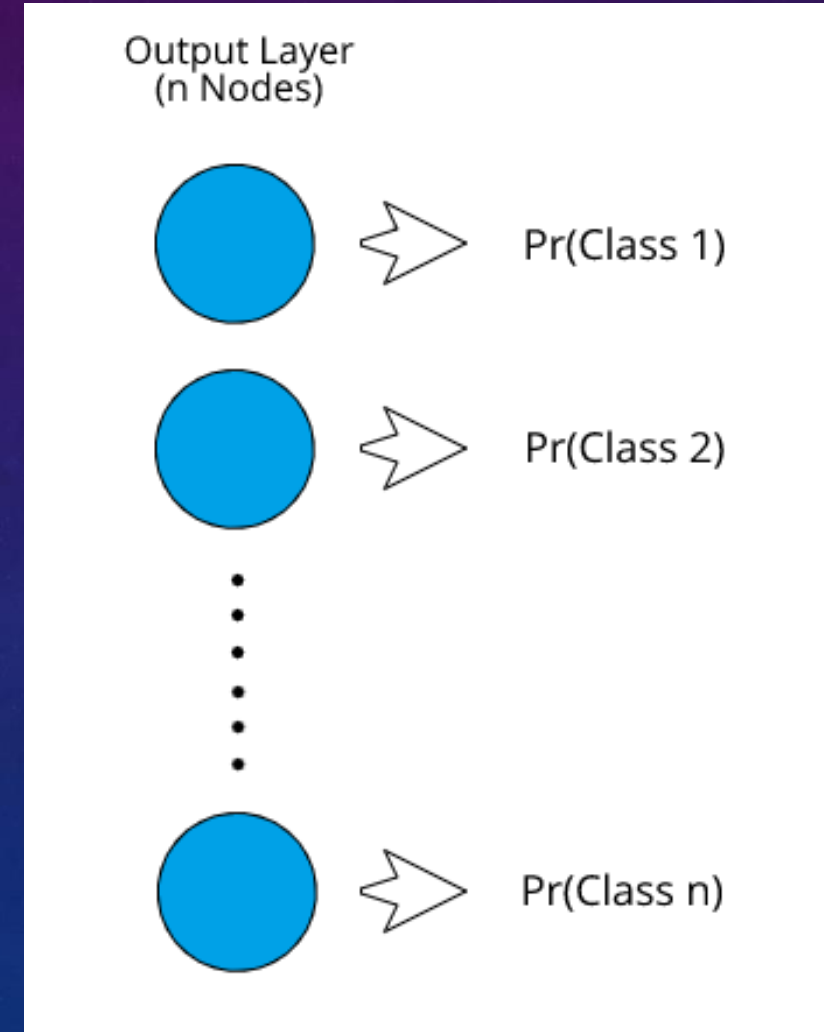
# Loss Function

- Loss function is used to measure the performance of a prediction model in terms of the difference between the prediction and ground-truth output.
- From a very simplified perspective, the loss function  $J$  can be defined as a function of the prediction and ground-true output.



# Classification Loss

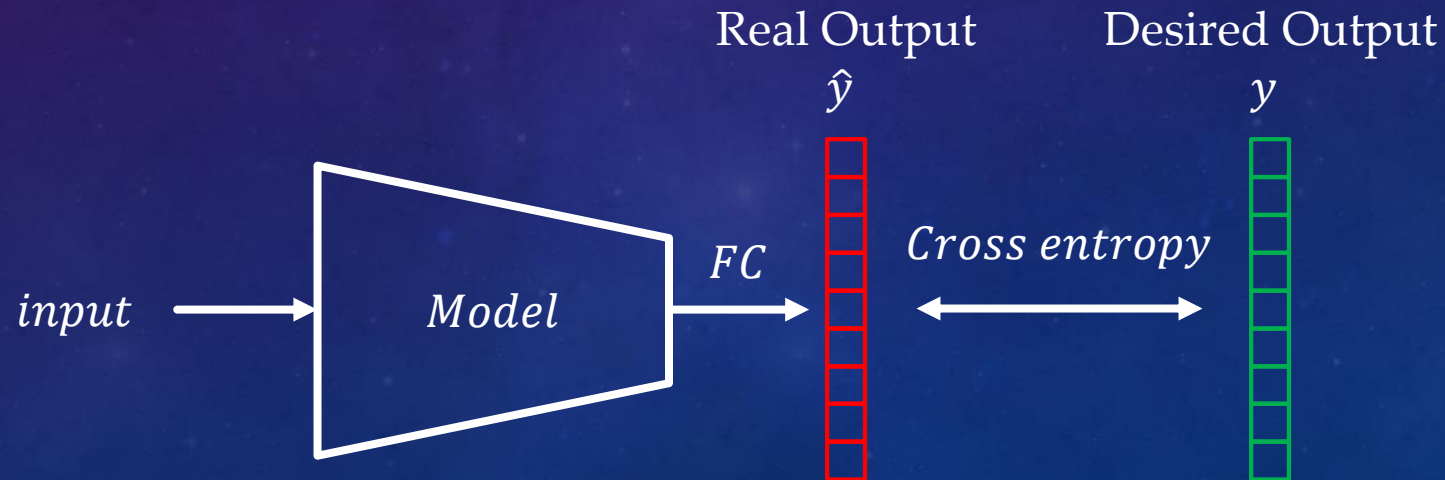
- When a neural network is used to predict a class label, we can consider it a classification model.
- The number of the output nodes is the number of classes in the data.
- Each node will represent a single class.
- The value of each output node represents the probability of that class being the correct class.



# Classification Loss

Given an input to the Model.

- We hope that the prediction of the model can be close to the ground-truth probabilities  $y = (1.0, 0.0, 0.0)^T$ .
- If our model predicts a different distribution, say  $\hat{y} = (0.4, 0.1, 0.5)^T$ , then we'd like to nudge the parameters so that  $\hat{y}$  gets closer to  $y$ .
- Cross entropy is a reasonable way for the task of classification.





# Softmax Function and Cross Entropy

## Softmax & Cross-Entropy

NN1

Row	Dog^	Cat^	Dog	Cat
#1	0.9	0.1	1	0
#2	0.1	0.9	0	1
#3	0.4	0.6	1	0

$$1/3 = 0.33$$

NN2

Row	Dog^	Cat^	Dog	Cat
#1	0.6	0.4	1	0
#2	0.3	0.7	0	1
#3	0.1	0.9	1	0

$$1/3 = 0.33$$

Classification Error

<https://youtu.be/C-PMt7ah1lQ> [18:20]

# Softmax Function

- The **softmax function**
  - As known as **soft-argmax** or **normalized exponential function**
  - It is a function that takes as input a vector of  $K$  real numbers, and normalizes it into a probability distribution consisting of  $K$  probabilities.
- Softmax is often used in *Neural Network*, to map the non-normalized output of a network to a probability distribution over predicted output classes.
- The standard (unit) softmax function  $\sigma: \mathbb{R}^K \rightarrow \mathbb{R}^K$  is define as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \text{ for } i = 1, \dots, k \text{ and } z = (z_1, \dots, z_k) \in \mathbb{R}^K$$

# Softmax Function

- We apply the standard exponential function to each element  $z_i$  of the input vector  $z$  and normalize these values by dividing by the sum of all these exponentials; this normalization ensures that the sum of the components of the output vector  $\sigma(z)$  is 1.
- Instead of  $e$ , a different base  $b > 0$  can be used; choosing a larger value of  $b$  will create a probability distribution that is more concentrated around the positions of the largest input values. Writing  $b = e^\beta$  or  $b = e^{-\beta}$  (for real  $\beta$ ) yields the expressions:

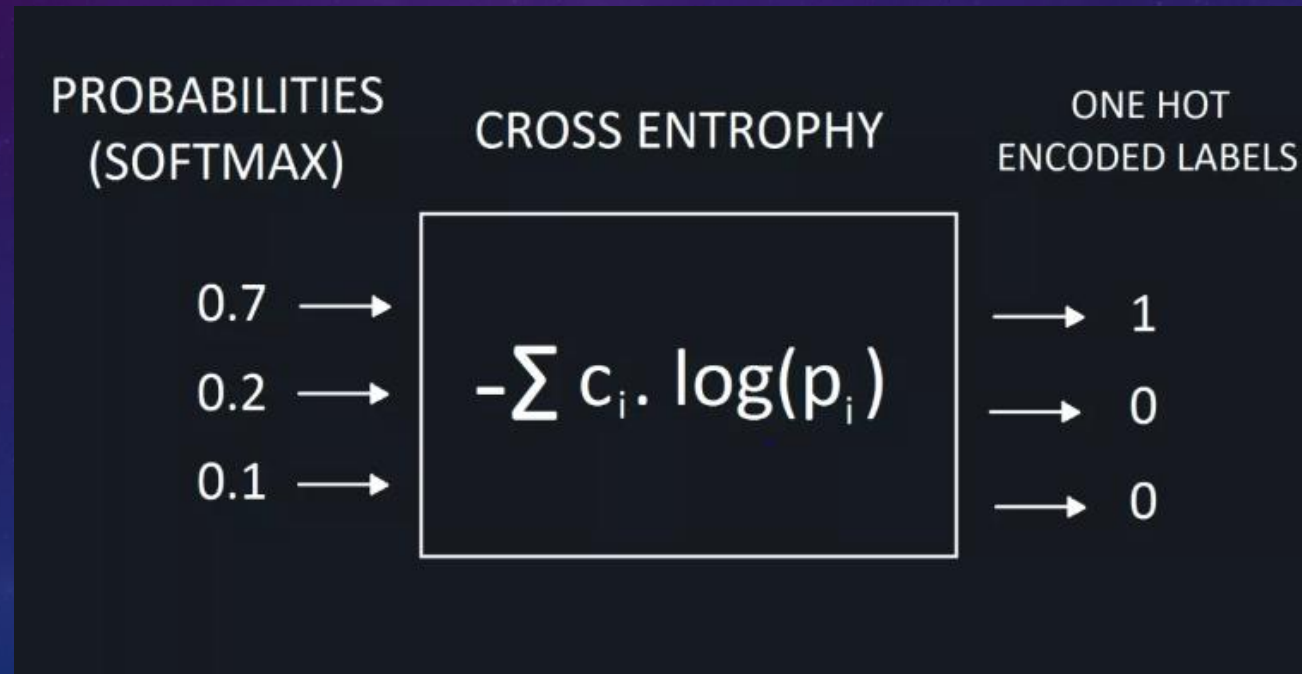
$$\sigma(z)_i = \frac{e^{\beta z_i}}{\sum_{j=1}^k e^{\beta z_j}} \quad \text{or} \quad \sigma(z)_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^k e^{-\beta z_j}} \quad \text{for } i = 1, \dots, k$$



# Cross Entropy

- Apply softmax function to normalize output in scale of [0, 1].
- Sum of outputs will always be equal to 1 when softmax is applied
- Apply one hot encoding transforms outputs in binary form.

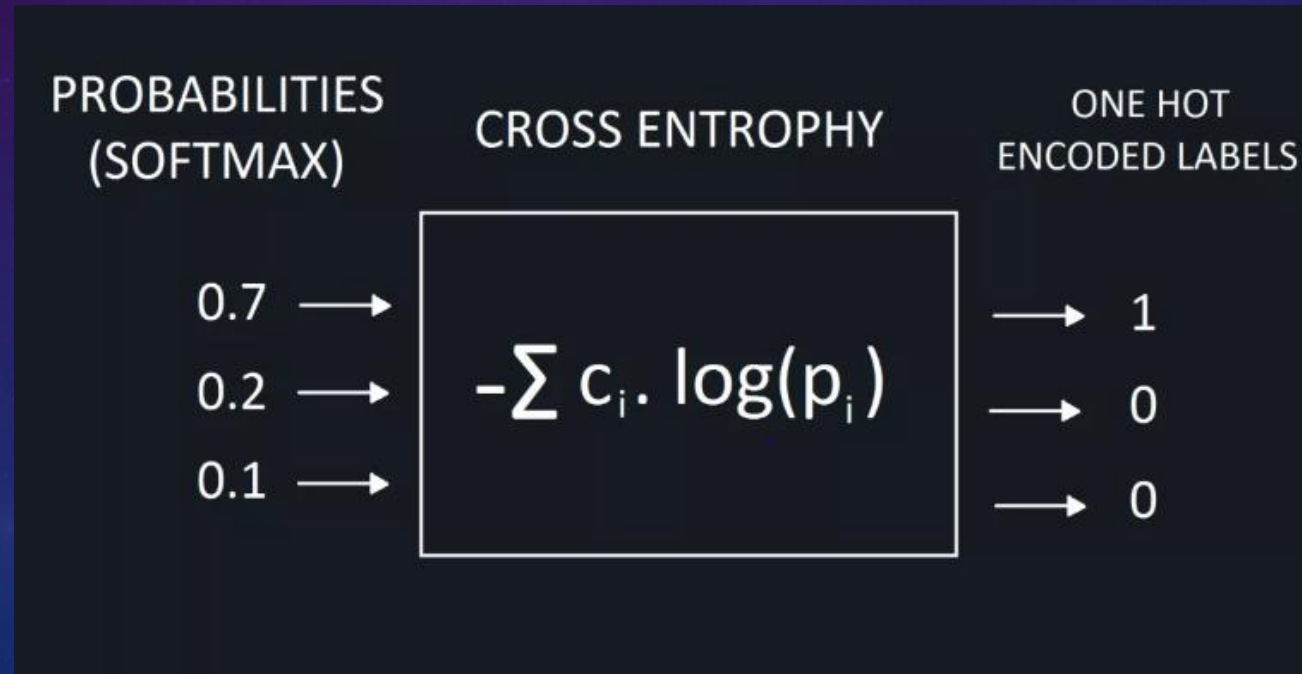
$\hat{y} = \sigma(z)$   
NN output



# Cross Entropy

- Softmax and one hot encoding is applied respectively to neural networks output layer.
- True labeled output would be predicted classification output.
- The cross entropy function correlates between probabilities and one hot encoded labels.

$\hat{y} = \sigma(z)$   
NN output



$y$   
(Ground-Truth)

# Cross Entropy

The derivative of loss function to back-propagate.

- If loss function were MSE, then its derivative would be easy (expected and predicted output).
- Things become more complex when error function is cross entropy.

$$E(y, \hat{y}) = - \sum_i y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)$$

$y$  refers to one hot encoded classes (or labels) whereas  $\hat{y}$  refers to softmax applied probabilities.

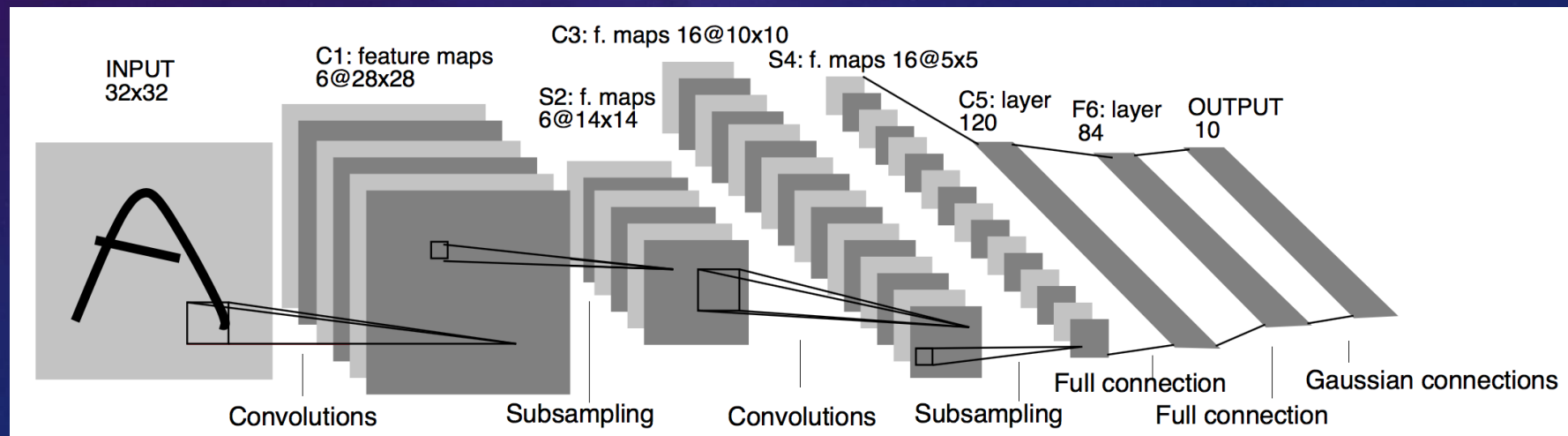
<https://datascience.stackexchange.com/questions/9302/the-cross-entropy-error-function-in-neural-networks>



# Example 2.6 Train the LeNet Network

Please download [2-6\\_CNN\\_MNIST.zip](#) and unzip it.

- Given a CNN base network architecture in the figure below.
- Consider the MNIST dataset as the input data: 90% for training and 10% for testing.
- Use the “Softmax with Cross Entropy” as the objective function.
- Please follow the instruction to train the CNN base handwritten digits classifier.
- With the predicted results and its ground-truth, please compute the confusion matrix for each class.



LeNet (1998)

# Common CNN Models

## Common Models

---

- AlexNet (ILSVRC 2012 winner)
- ZF Net (2013 winner)
- GoogLeNet (2014 winner)
- VGG (2014 runner-up)
- ResNet (2015 winner)