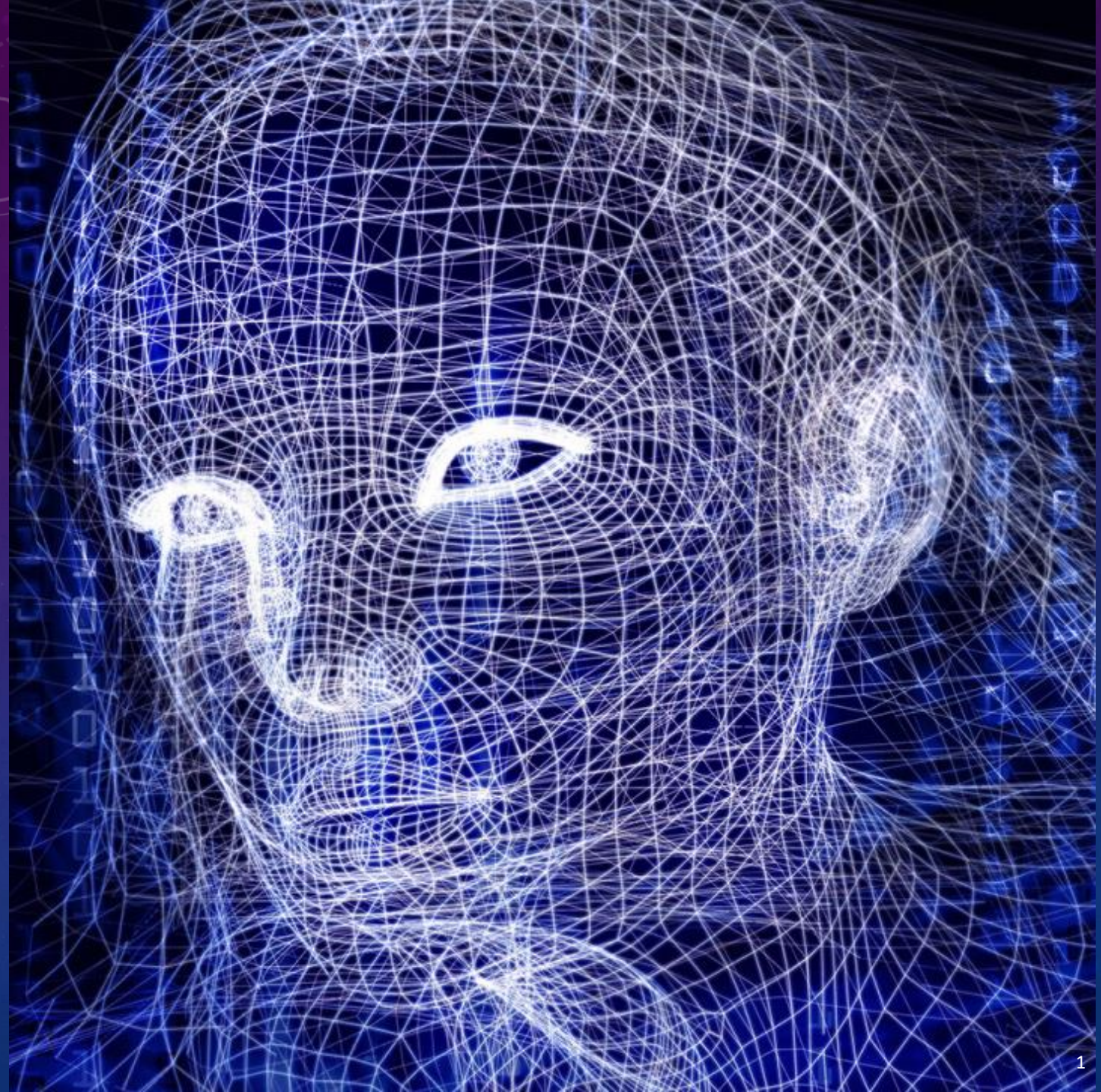
LECTURE SERIES FOR DIGITAL SURVEILLANCE SYSTEMS AND APPLICATION

INTRODUCTION TO DEEP LEARNING



徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology

Convolutional Neural Networks

1) Convolutions
2) Discrete convolutions
3) Pooling and Activation function
4) Convolution Layers
5) CNN Architectures
6) Loss Functions
7) Feature map visualization

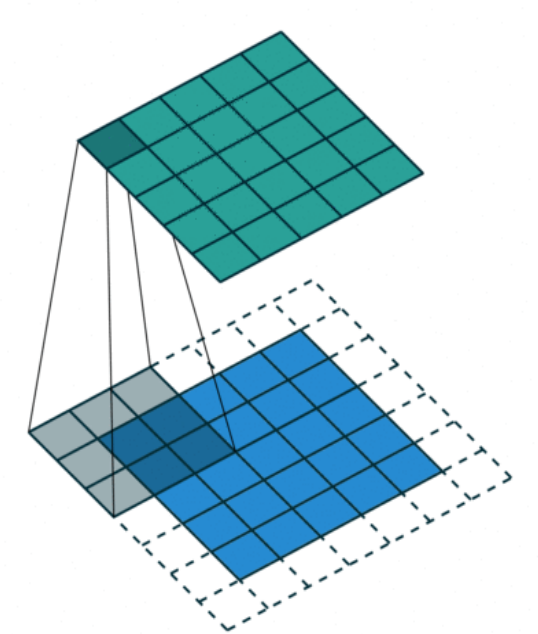
Convolutional Neural Networks (CNNs) Explained

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

Convolutional Operation

Convolutions

First we need to agree on a few parameters that define a convolutional layer.

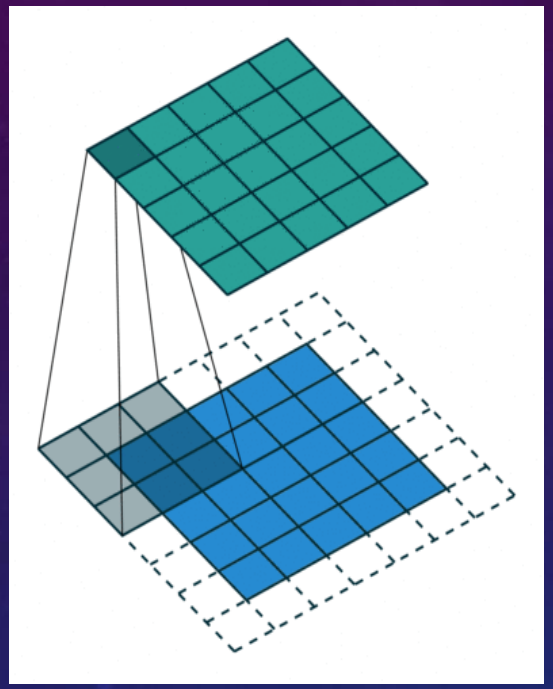


- **Kernel Size:** The kernel size defines the field of view of the convolution. A common choice for 2D is 3—that is 3x3 pixels.
- **Stride**: The stride defines the step size of the kernel when traversing the image. While its default is usually 1, but it can be 2 or other step sizes.

2D convolution using a kernel size of 3, stride of 1 and padding

Convolutions

First we need to agree on a few parameters that define a convolutional layer.



- **Padding**: The padding defines how the border of a sample is handled. A (half) padded convolution will keep the spatial output dimensions equal to the input, whereas unpadded convolutions will crop away some of the borders if the kernel is larger than 1.
- **Input & Output Channels**: A convolutional layer takes a certain number of input channels (I) and calculates a specific number of output channels (O). The needed parameters for such a layer can be calculated by I*O*K, where K equals the number of values in the kernel.

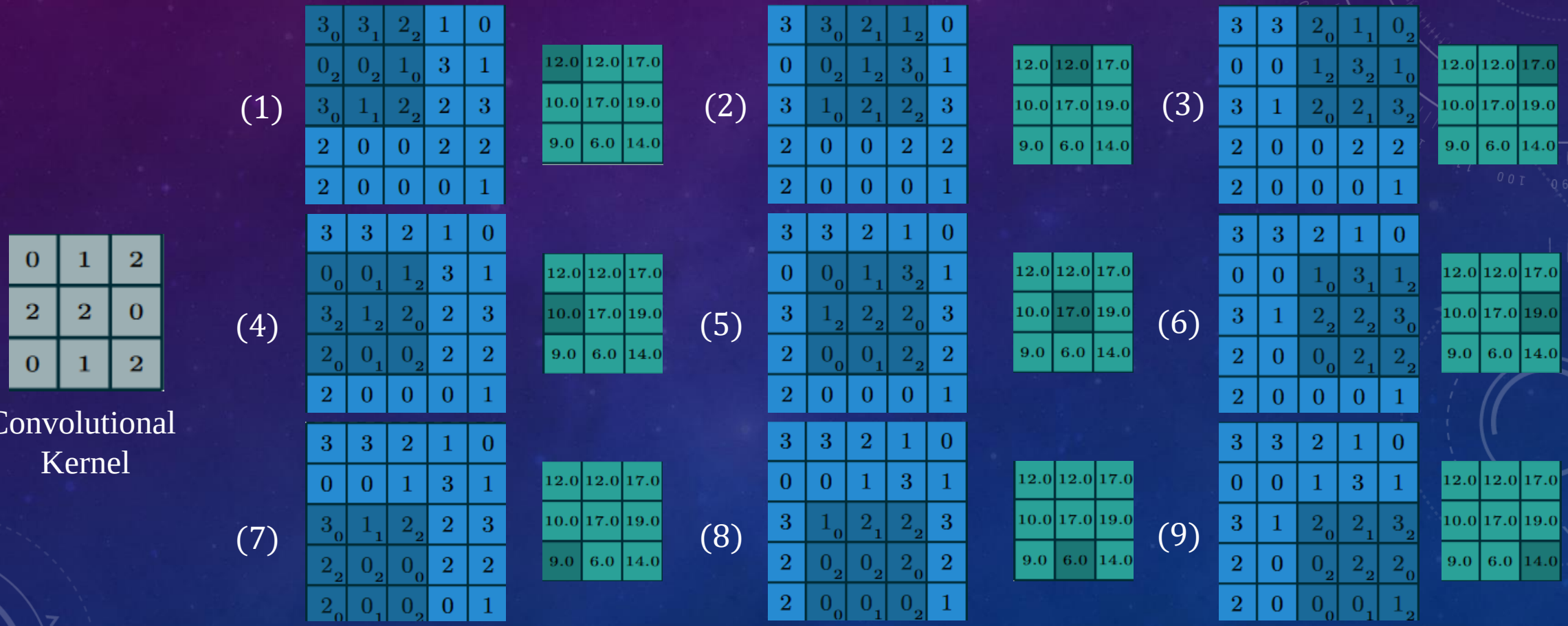
2D convolution using a kernel size of 3, stride of 1 and padding

Discrete Convolutions

- The bread and butter of neural networks is affine transformations：a vector is received as input and is multiplied with a matrix to produce an output.
- This is applicable to any type of input, be it *an image*, *a sound clip* or an *unordered collection of features*. They share these important properties：
  - They are stored as multi-dimensional arrays.
  - They feature one or more axes for which ordering matters (e.g., width and height axes for an image, time axis for a sound clip).
  - One axis, called the channel axis, is used to access different views of the data (e.g., the red, green and blue channels of a color image, or the left and right channels of a stereo audio track).

Discrete Convolutions

Matrix below provides an example of a discrete convolution. The light blue grid is called *the input feature map*, the green grid is called the *output feature map*, and the gray grid is *convolution kernel*.



Computing the output values of a discrete convolution

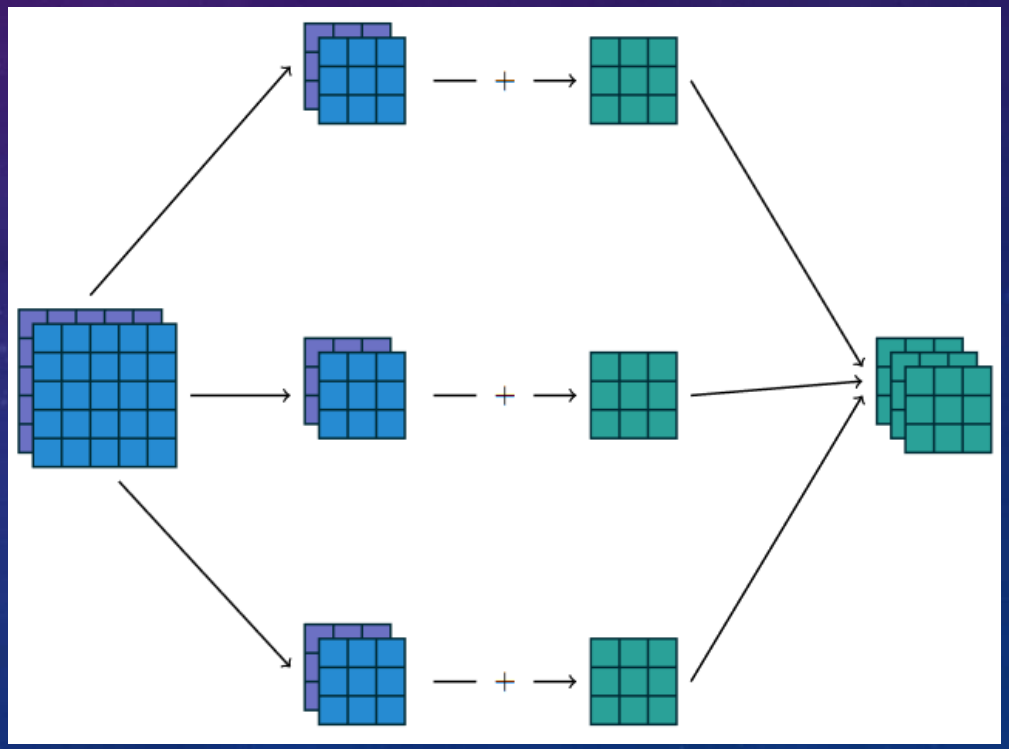
Discrete Convolutions

At each location, the product between each element of the kernel and the input element it overlaps is computed and the results are summed up to obtain the output in the current location. The procedure **can be repeated using different kernels** to form as many output feature maps (or layers) as desire.

A *kernel* (shaded area) of value slides across the input feature map.

0	1	2
2	2	0
0	1	2

Convolutional Kernel



Discrete Convolutions

The convolution depicted in first sample is an instance of a 2-D convolution, but it can be generalized to N-D convolutions. The collection of kernels defining a discrete convolution has a shape corresponding to some permutation of $(n, \mathrm{m}, k_1, \ldots, k_n)$, where

$$n : number\ of\ output\ feature\ maps,$$

$$m : number\ of\ input\ feature\ maps,$$

$$k_j : kernel\ size\ along\ axis\ j.$$

Discrete Convolutions

The following properties affect the output size o_j of a convolutional layer along axis j ：

i_j ：input size along axis j,

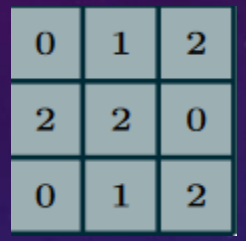
k_j ：kernel size along axis j,

s_j ：stride (distance between two consecutive position of the kernel) along axis j,

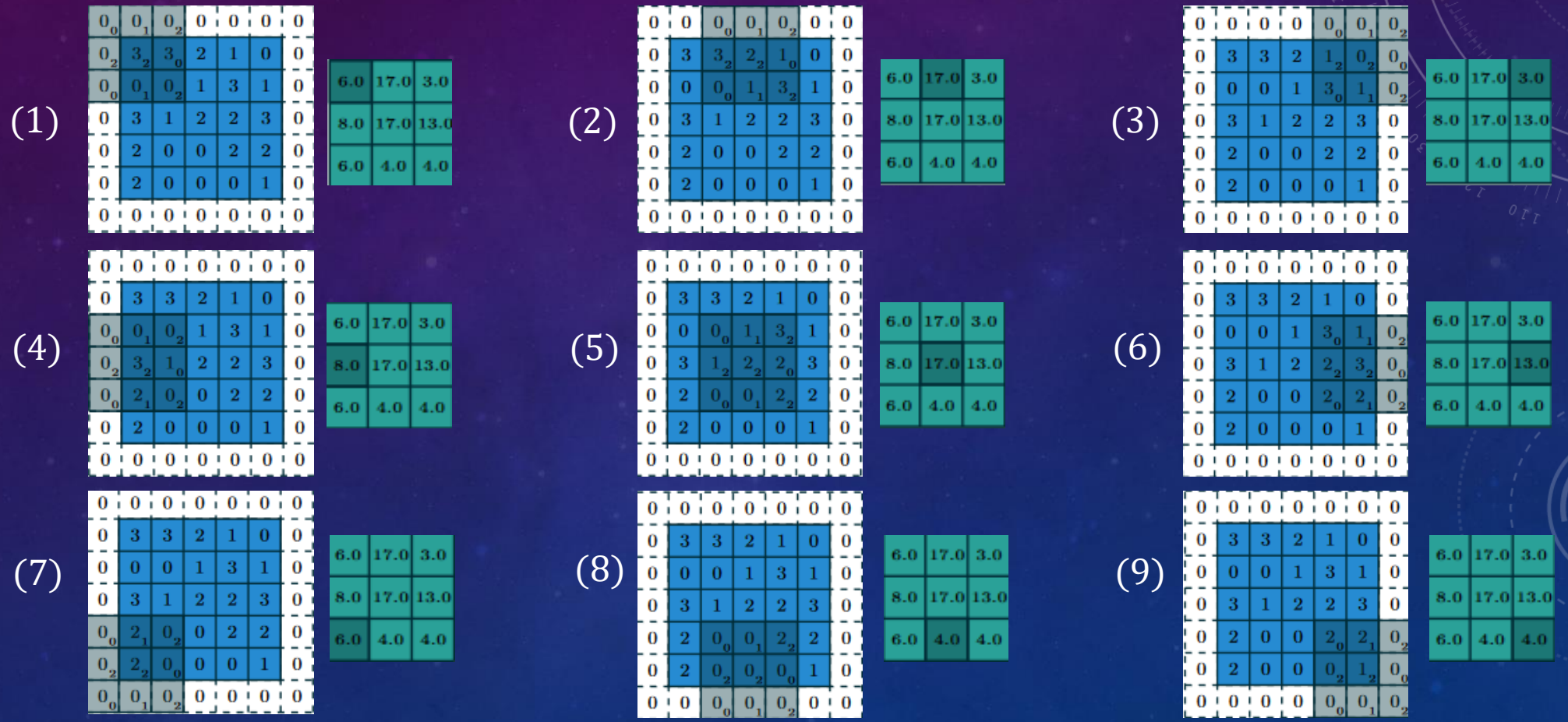
p_j ：zero padding (number of zeros concatenated at the beginning and at the end of an axis) along axis j,

Discrete Convolutions

The example below shows a 3 × 3 kernel applied to a 5 × 5 input padded with a 1 × 1 border of zeros using 2 × 2 strides.



Convolutional Kernel



Computing the output values of a discrete convolution for $N = 2$, $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, and $p_1 = p_2 = 1$.

Discrete Convolutions

Note that ： strides constitute a form of *subsampling*. As an alternative to being interpreted as a measure of how much the kernel is translated, strides can also be viewed as how much of the output is retained. For instance, moving the kernel by hops of two is equivalent to moving the kernel by hops. of one but retaining only odd output elements.

An alternative way of viewing strides. Instead of translating the 3×3 kernel by increments of s= 2(left), the kernel is translated by increments of 1 and only one in s=2 output elements is retained (right).

See Coding Manual, Chapter 2, Page 2.

Convolution Arithmetic

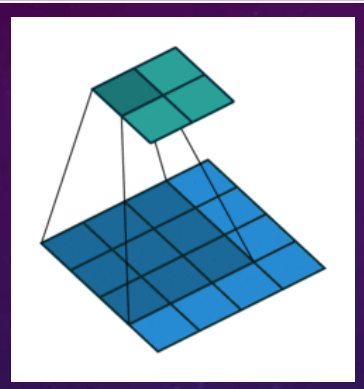
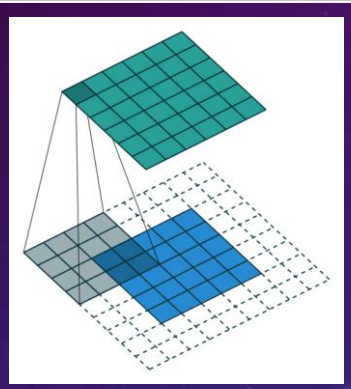
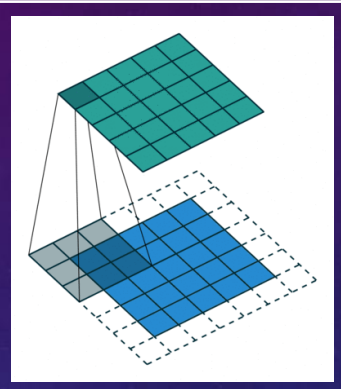
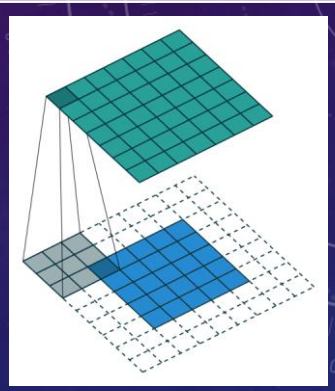
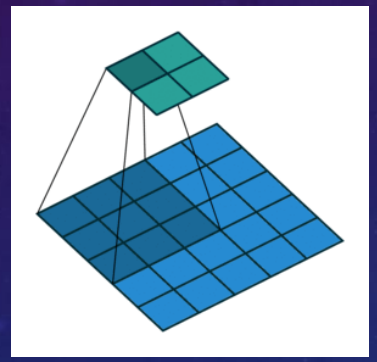
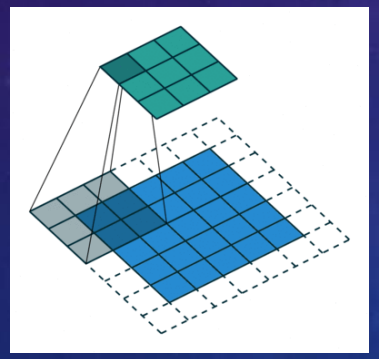
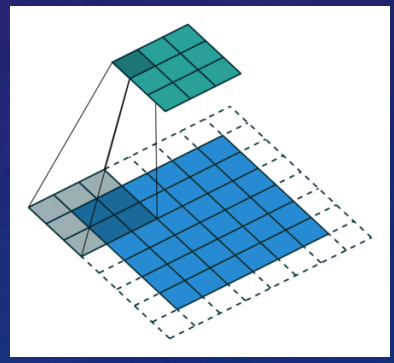
The analysis of the relationship between convolutional layer properties is eased by the fact that they don't interact across axes, i.e., the choice of kernel size, stride and zero padding along axis j only affects the output size of axis j. Because of that, we'll focus on the following simplified setting:

- 2-D discrete convolutions ($N = 2$)
- square inputs ($i_1 = i_2 = i$)
- square kernel size ($k_1 = k_2 = k$)
- same strides along both axes ($s_1 = s_2 = s$)
- same zero padding along both axes ($p_1 = p_2 = p$)

This facilitates the analysis and the visualization, but keep in mind that the results outlined here also generalize to the N-D and non-square cases.

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)	

https://github.com/vdumoulin/conv_arithmetic

Exercise 2-1 – Design Convolution Filter

Please download 2-1_Convolution_Example.py and design your own convolution filter with a

(1) 3x3 convolution filter

(2) 5x5 convolution filter

(3) 7x7 convolution filter

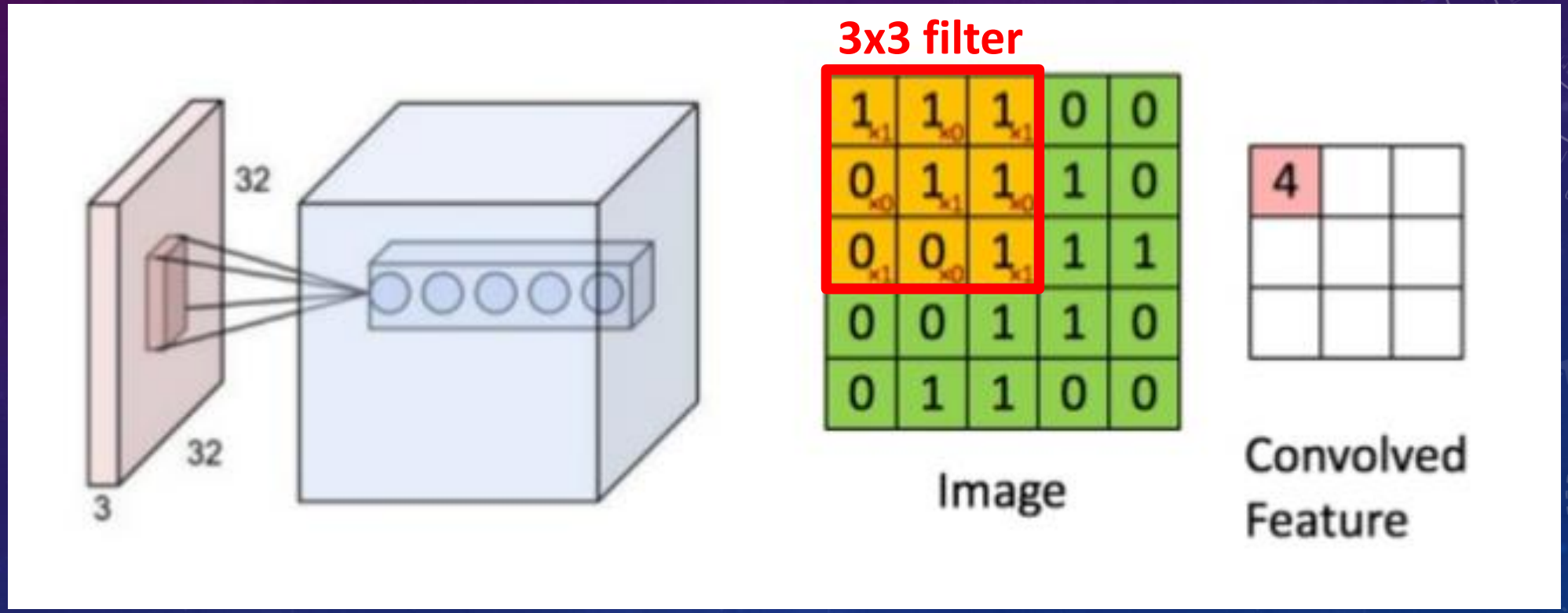
with random coefficients in normal distribution (std=1, mean=5), and compute the output by convolving I_{mg} with F_{conv}. Then, calculate the information loss with original image. Please write down result and your code in MS Word to Moodle

Change it in to normal distribution

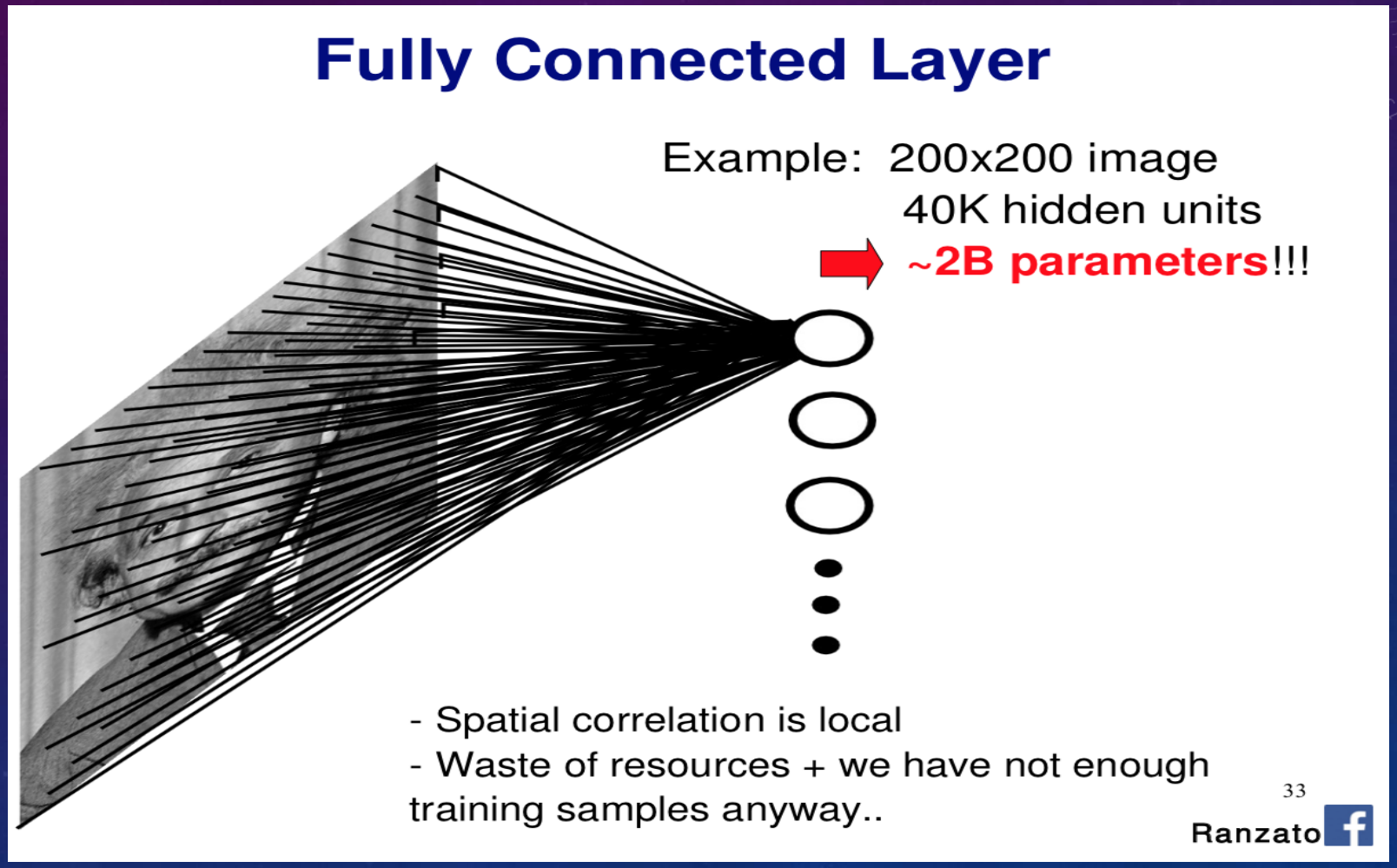
```
KERNAL_SIZE = 3
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

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
```

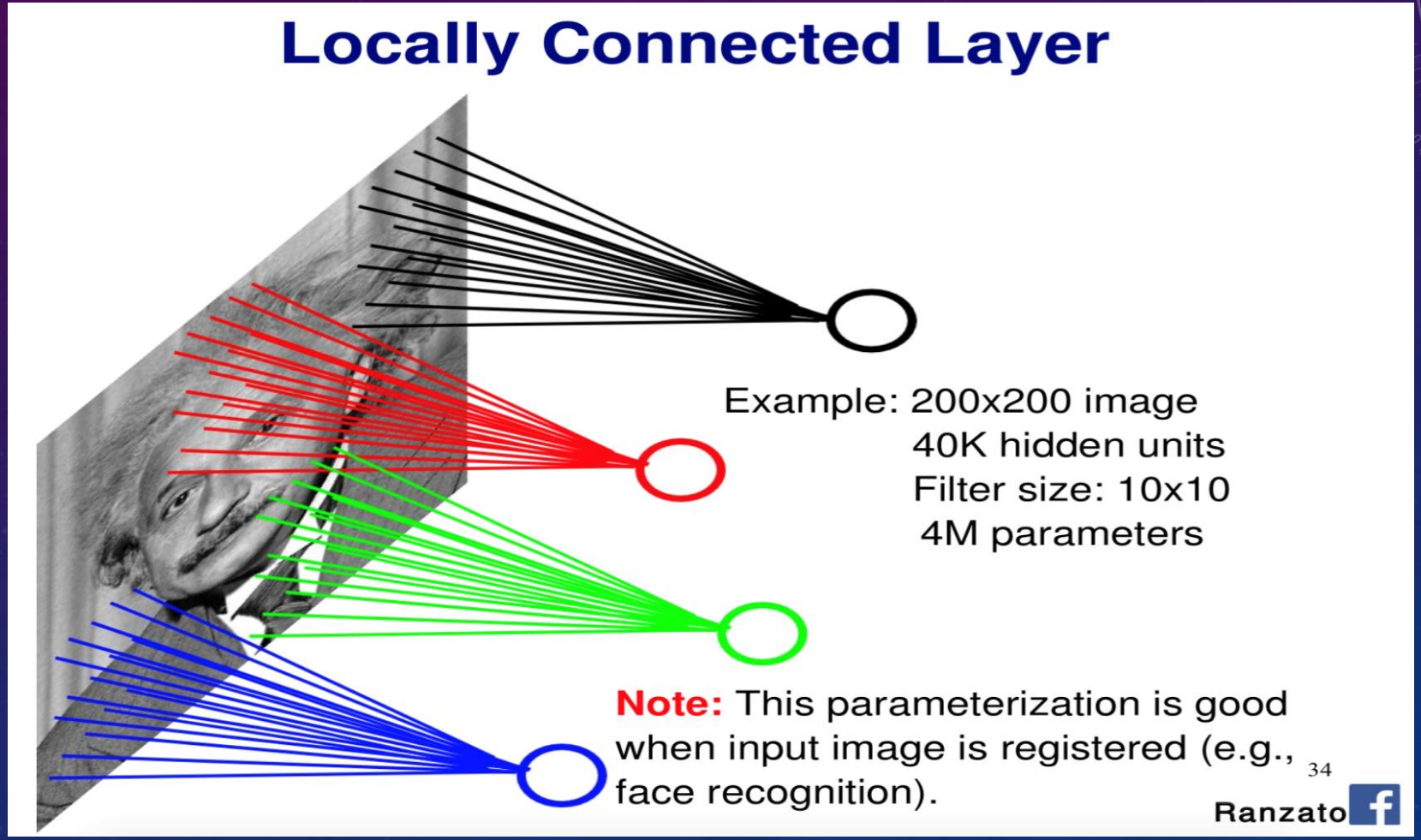
Convolutional Layer



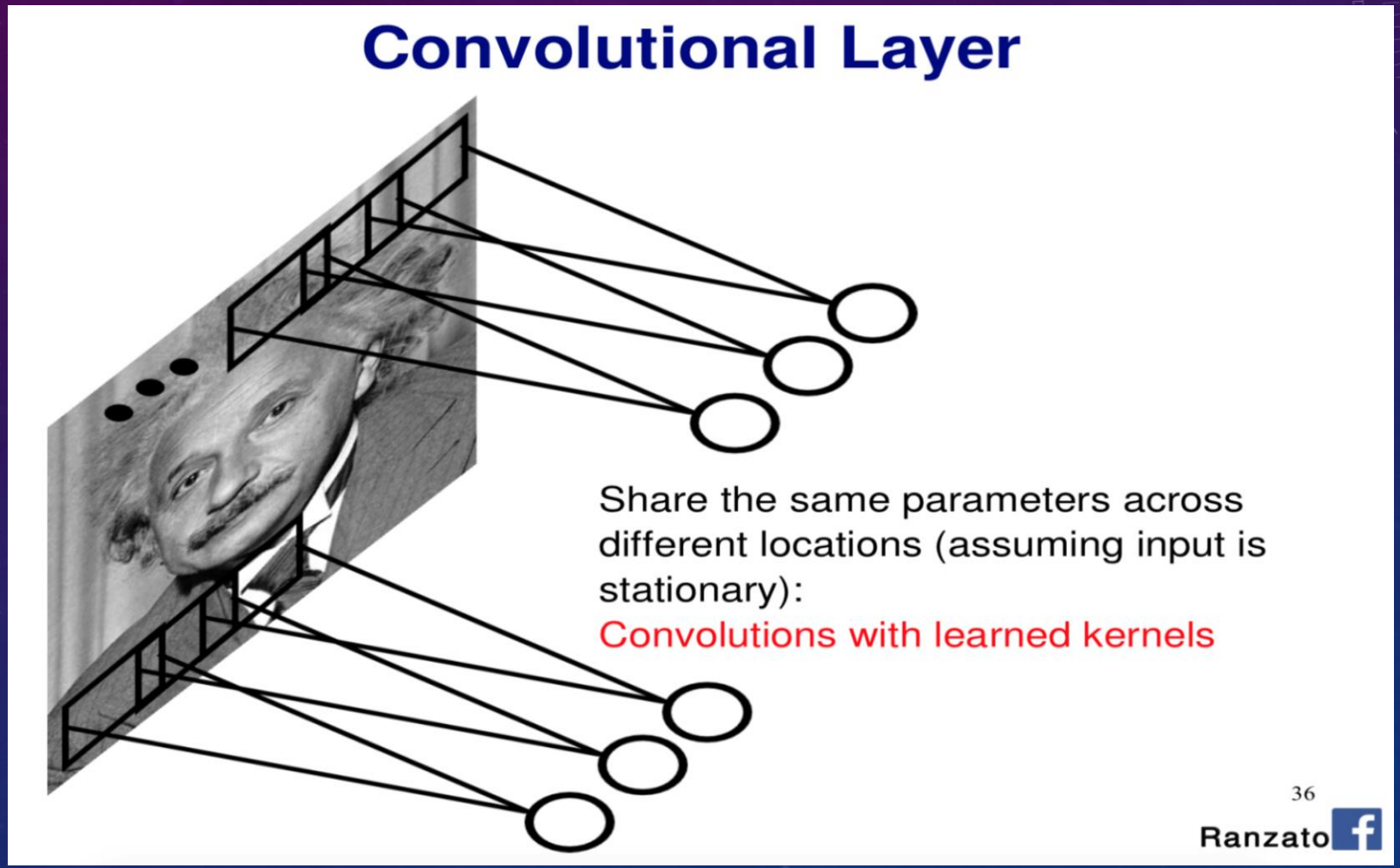
Fully Connected Layer



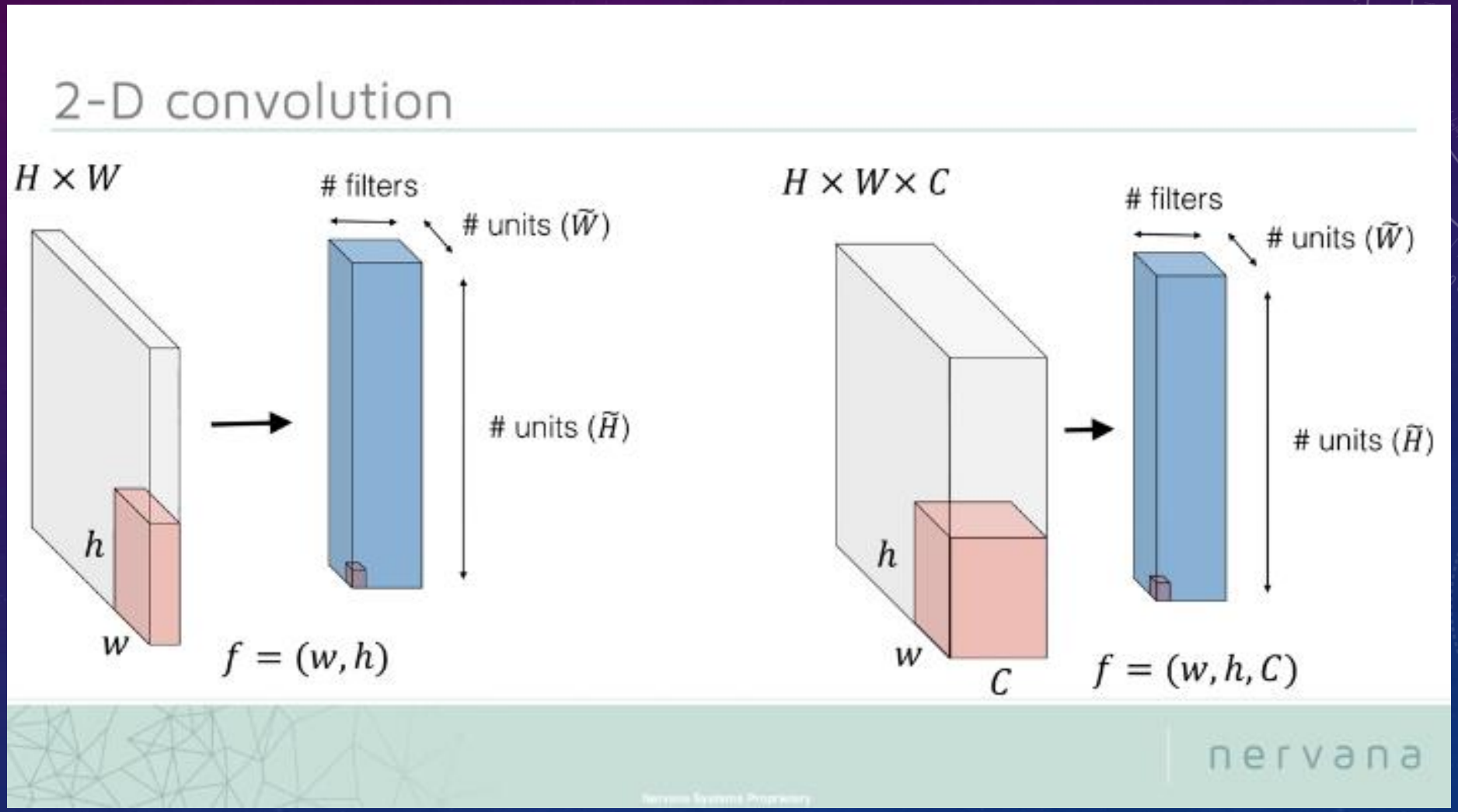
Locally Connected Layer



Convolutional Layer

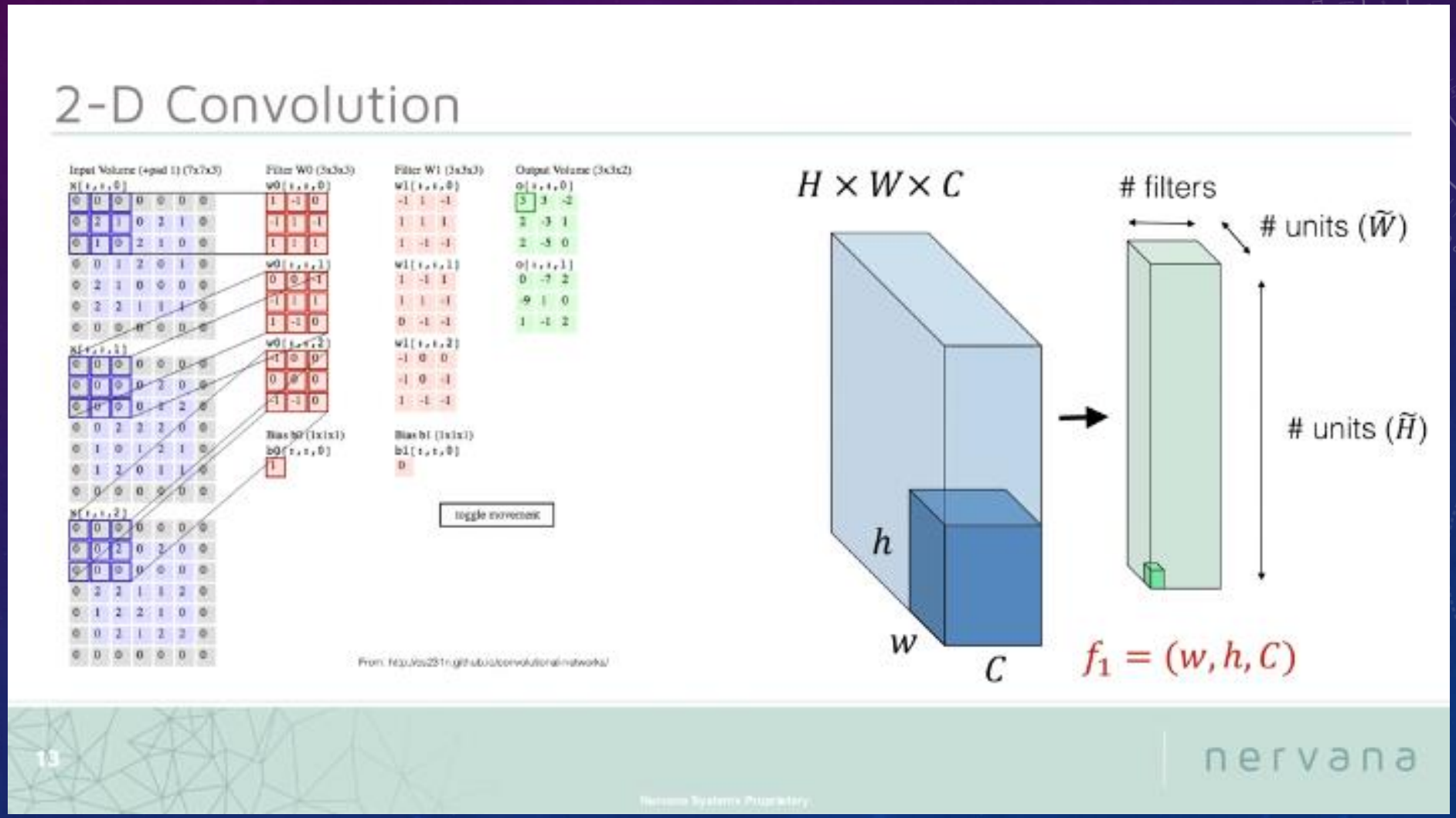


2-D Convolution

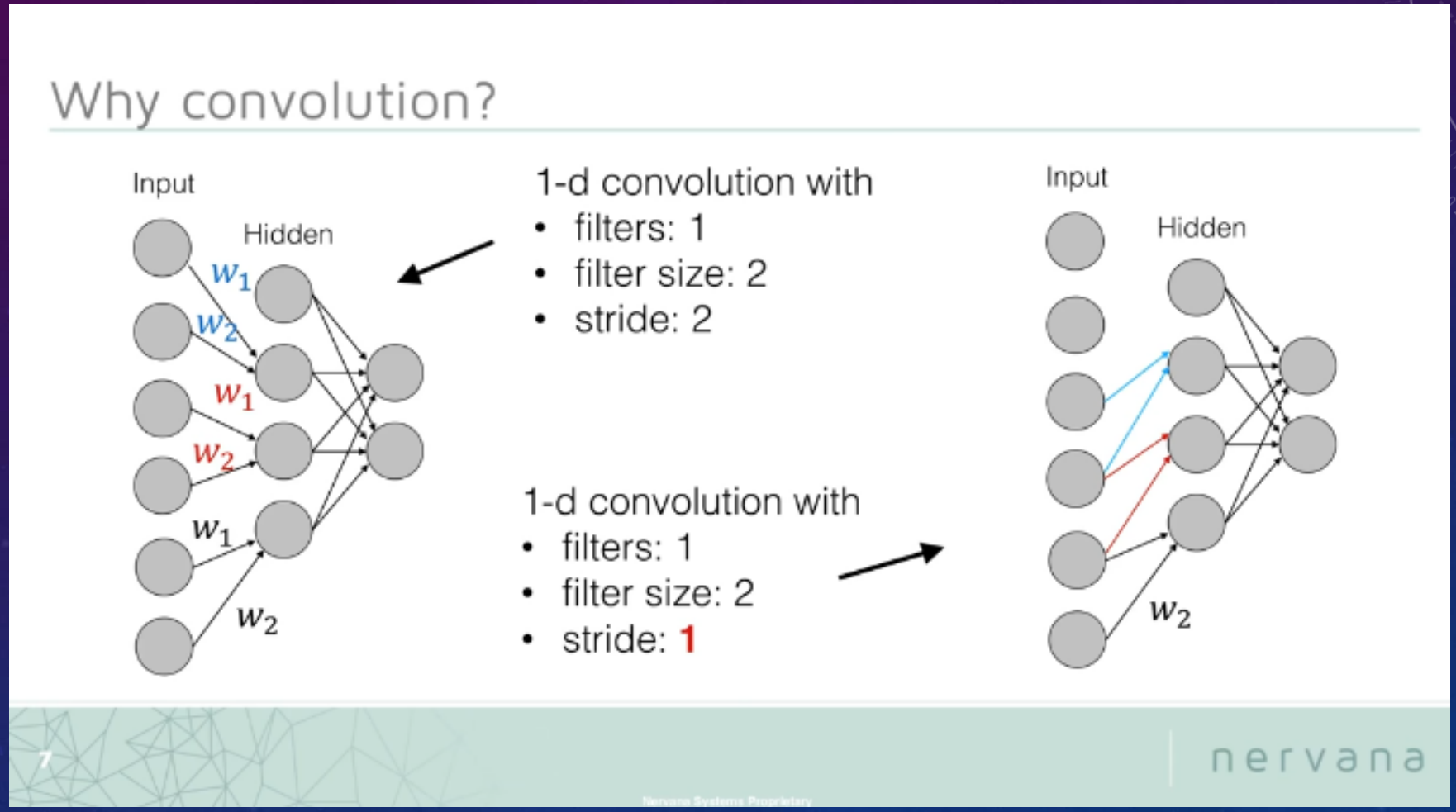


https://www.youtube.com/watch?v=SQ67NBCLV98 [27:48]

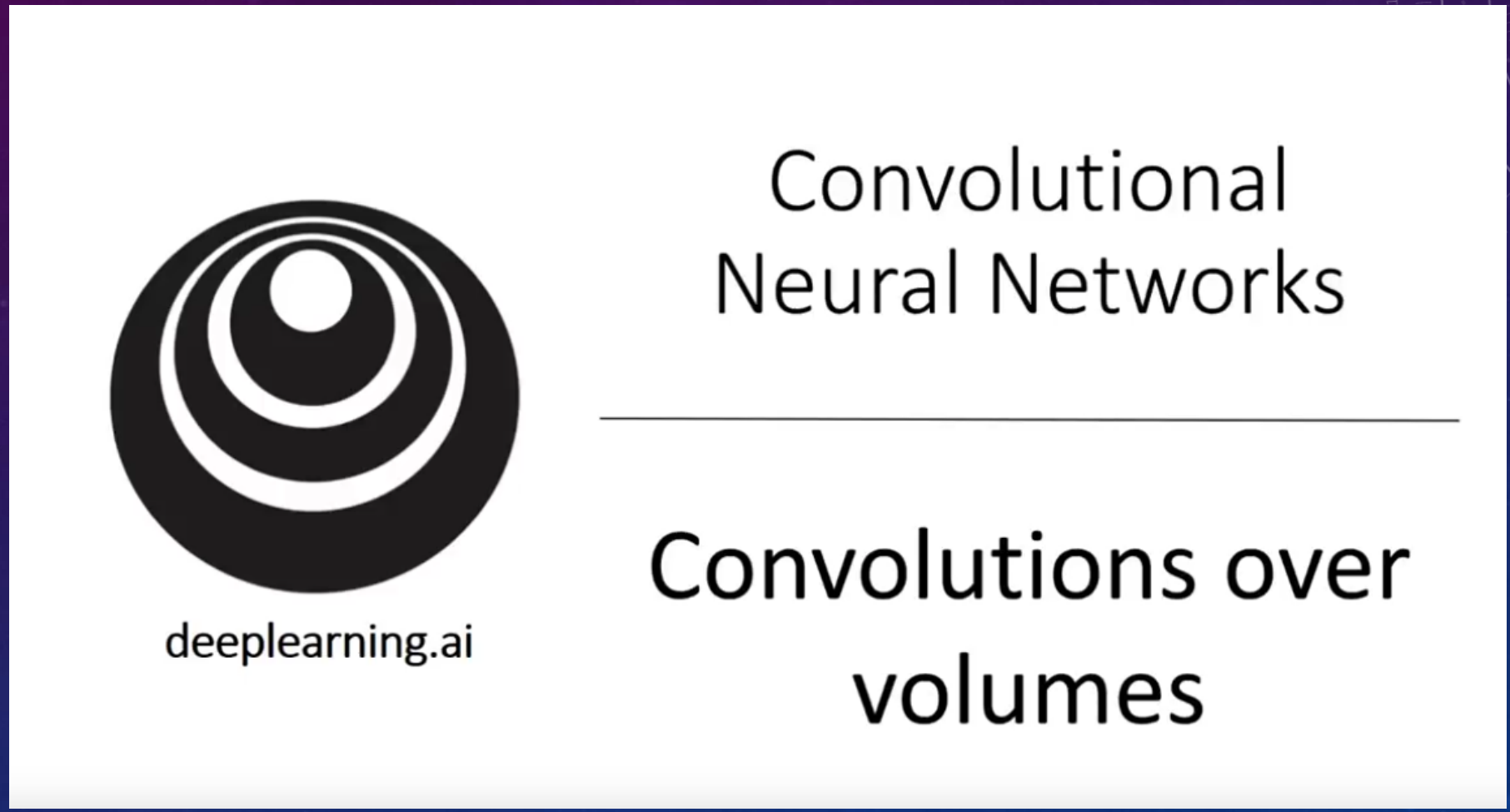
2-D Convolution



Why Convolution?



Strided Convolutions – Andrew Ng

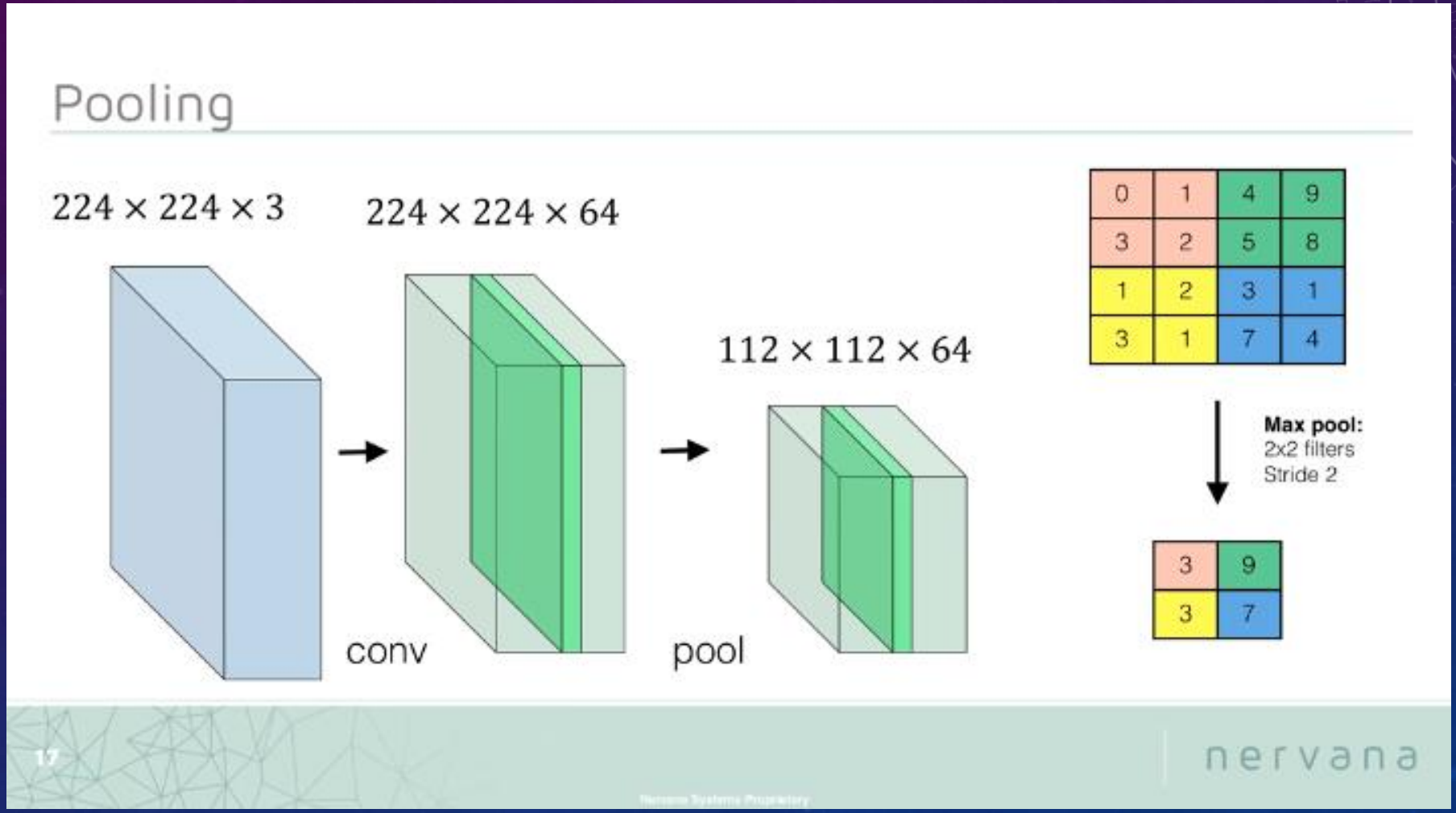


https://www.youtube.com/watch?v=KTB_OFoAQcc [10:44]

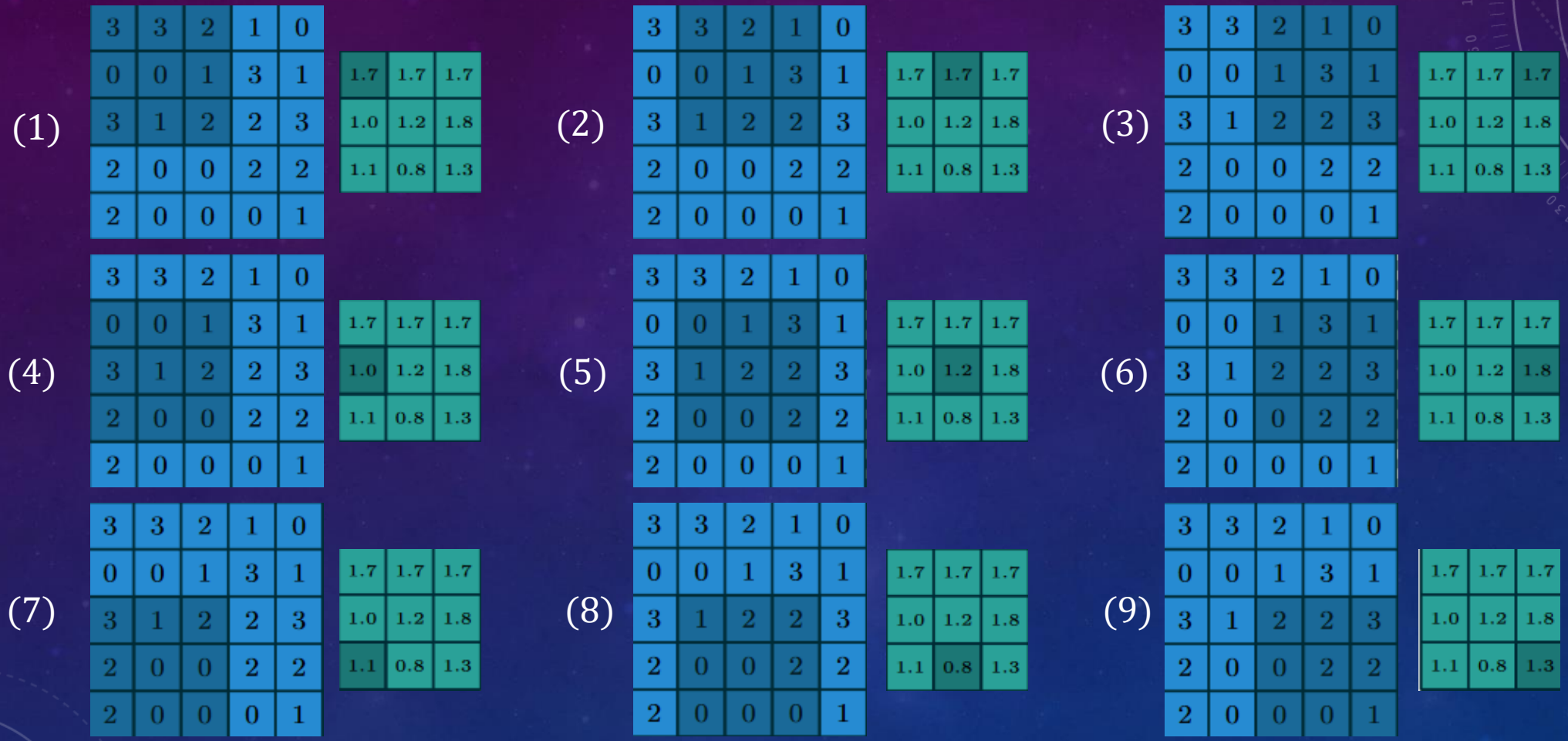
Max Pooling in Convolutional Neural Networks Explained

https://www.youtube.com/watch?v=ZjM_XQa5s6s [10:49]

Pooling

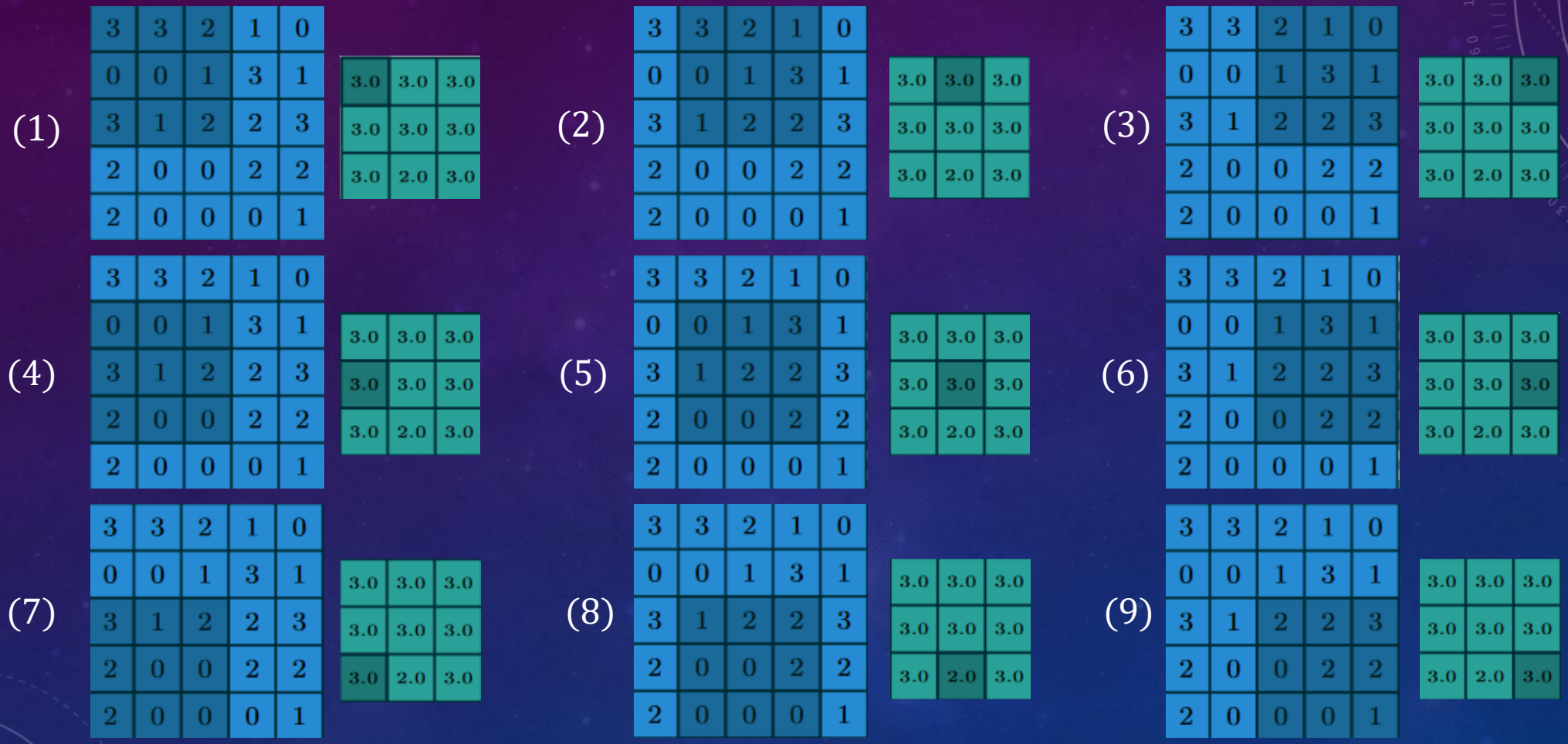

Pooling
224 × 224 × 3
224 × 224 × 64
112 × 112 × 64
conv
pool
0 1 4 9
3 2 5 8
1 2 3 1
3 1 7 4
Max pool:
2x2 filters
Stride 2
3 9
3 7
17
nervana

Pooling : Average Pooling



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.

See Coding Manual, Chapter 2, Page 6.

Exercise 2-2 – Pooling Practice

Please download 2-2_Convolution_Pooling_Example.py and design your own convolution filter with a 3x3 convolution filter with random coefficients in normal distribution (mean=2, std=0), and compute the output by convolving I_{mg} with F_{conv}. Then, upload the report in MS Word or PDF to Moodle

Change it in to normal distribution

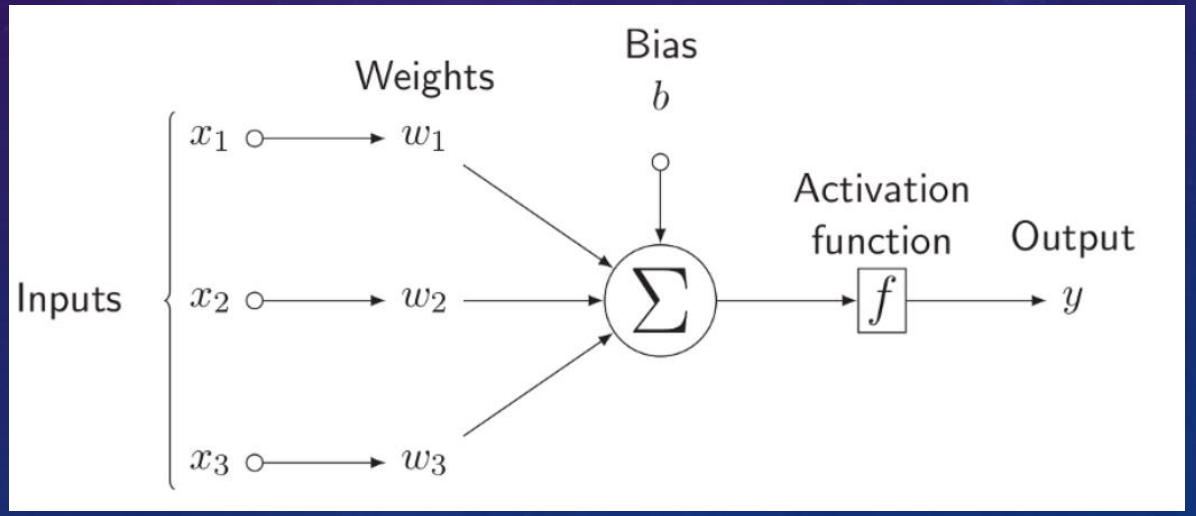
```
KERNAL_SIZE = 3
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

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
```

Please save result and your code in MS Word to Moodle

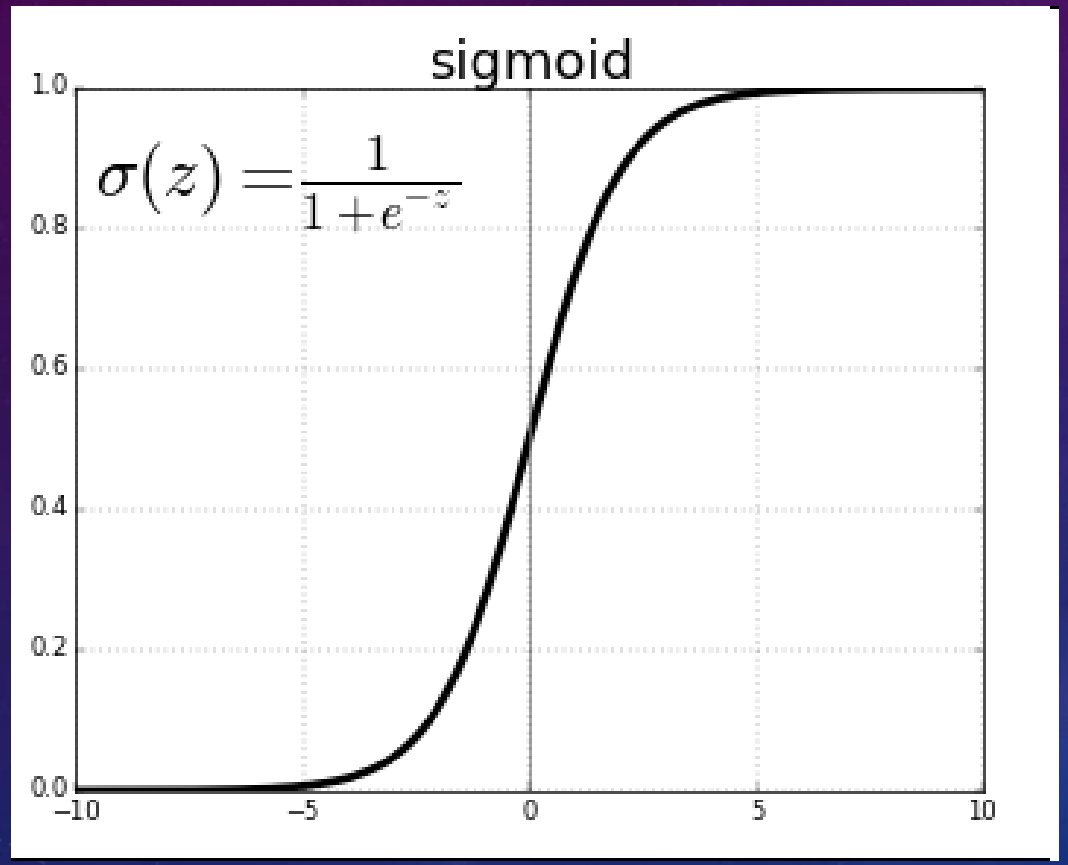
Activation Function

- 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.



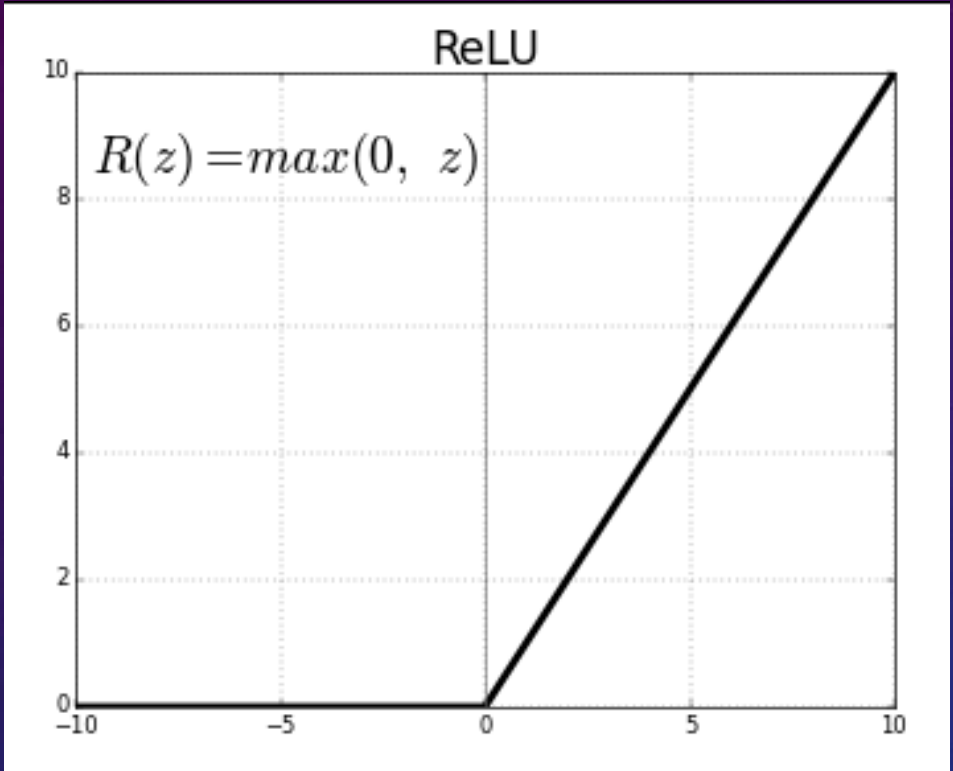
https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network

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

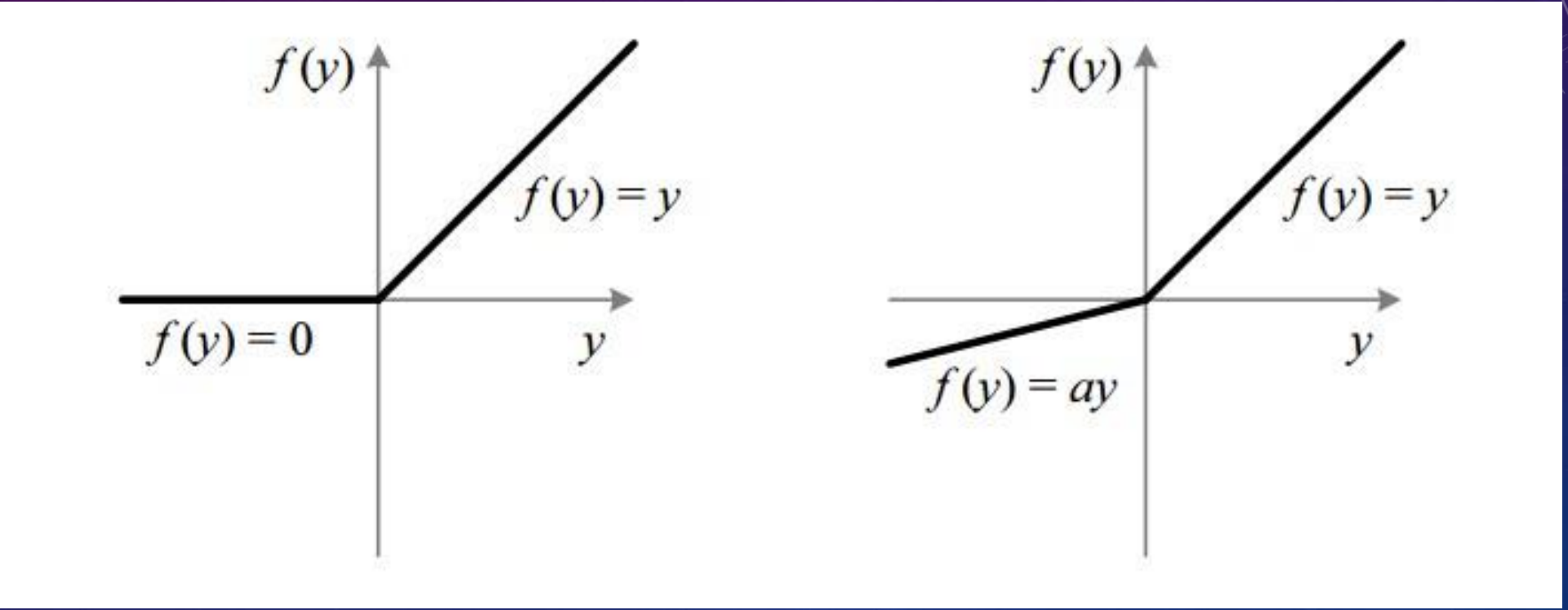


- The ReLU is half rectified (from bottom). *f(z)* is zero when *z* is less than zero and *f(z)* is equal to *z* when *z* is above or equal to zero.

- 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



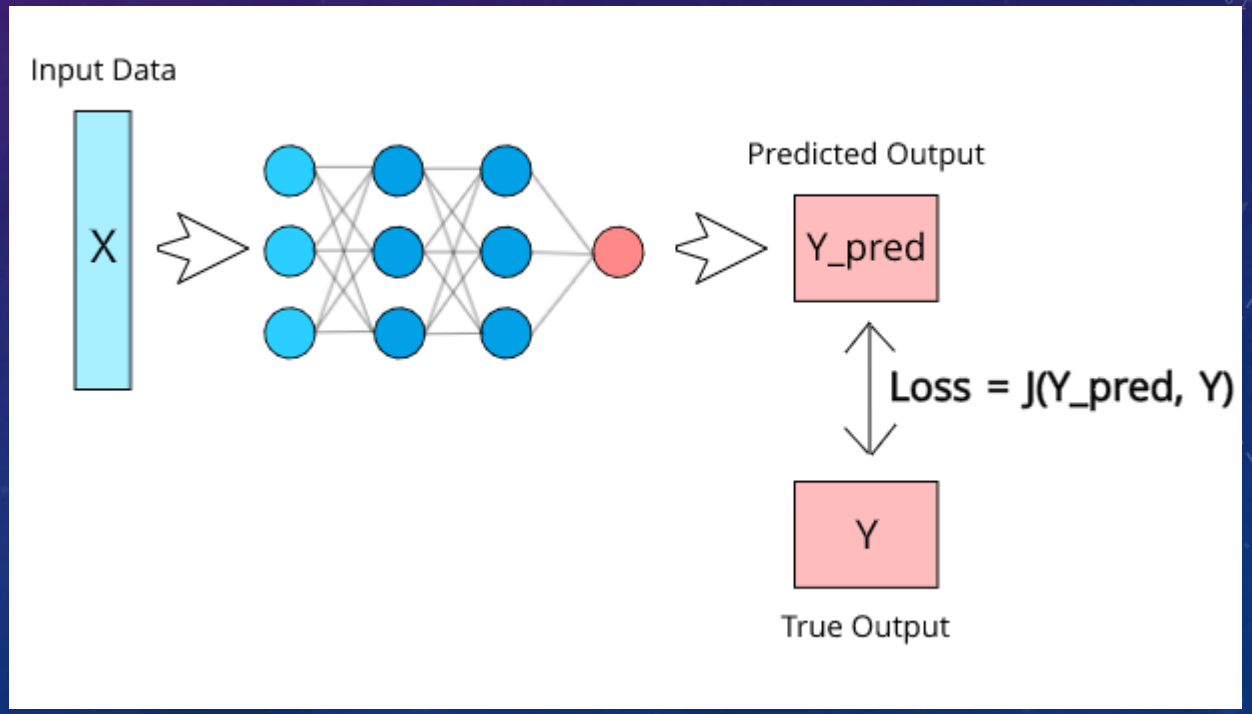
The leak helps to increase the range of the ReLU function.
When a is not 0.01 then it is called Randomized ReLU.

Which Activation Function Should I Use?

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

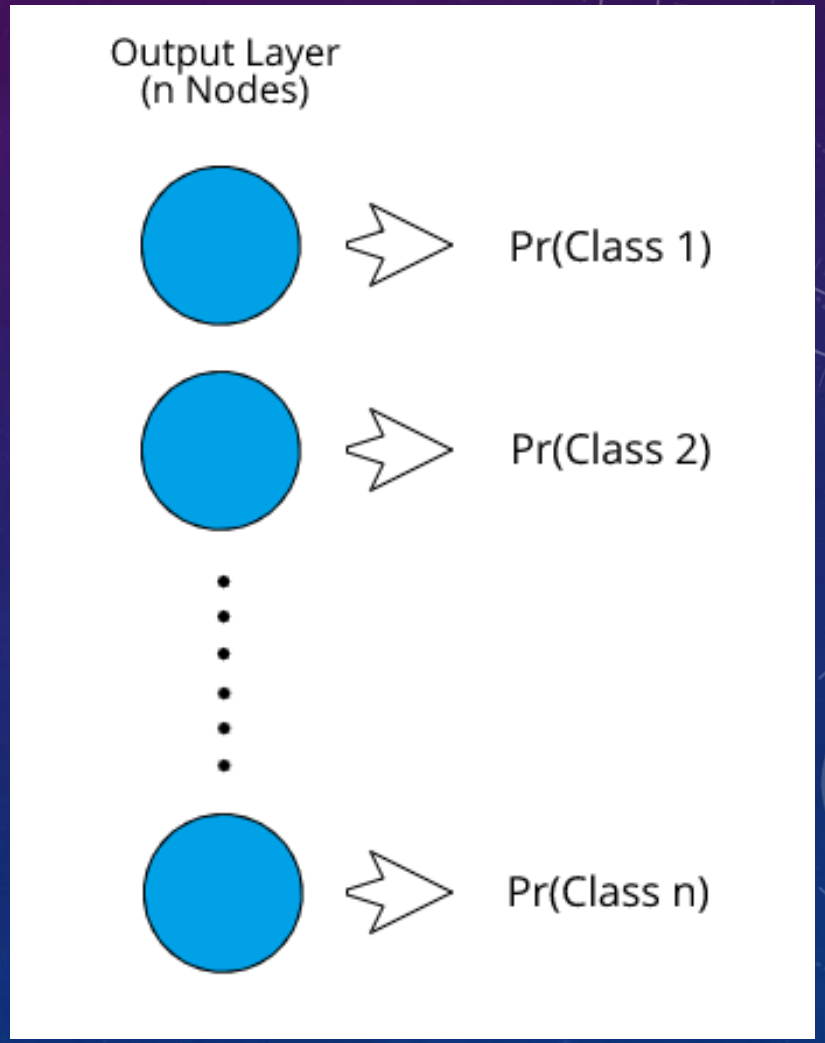
Loss Functions

- Loss function is used as measurement of how good a prediction model does in terms of being able to predict the expected outcome.
- From a very simplified perspective, the loss function (J) can be defined as a function which takes in two parameters:
    1. Predicted Output
    2. True Output



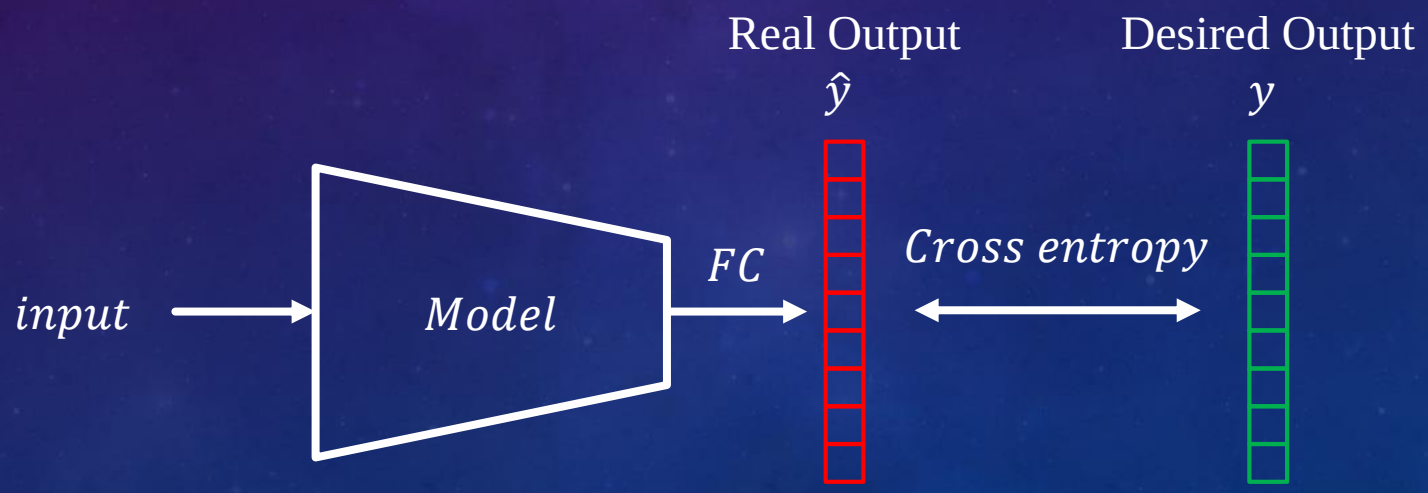
Classification Loss

- When a neural network is trying to predict a discrete value, we can consider it to be a classification model.
- The number of nodes of the output layer will depend on the number of classes present in the data. Each node will represent a single class. The value of each output node essentially represents the probability of that class being the correct class.



Classification Loss

- During training, we put in an image of a landscape, and we hope that our model produces predictions that are close to the ground-truth class 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 might be most reasonable way for the task of classification.



Softmax Function

- The **softmax function**, also known as **softargmax** or **normalized exponential function**, 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 $\sigma: \mathbb{R}^K \to \mathbb{R}^K$ is define as follows:

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

Softmax Function

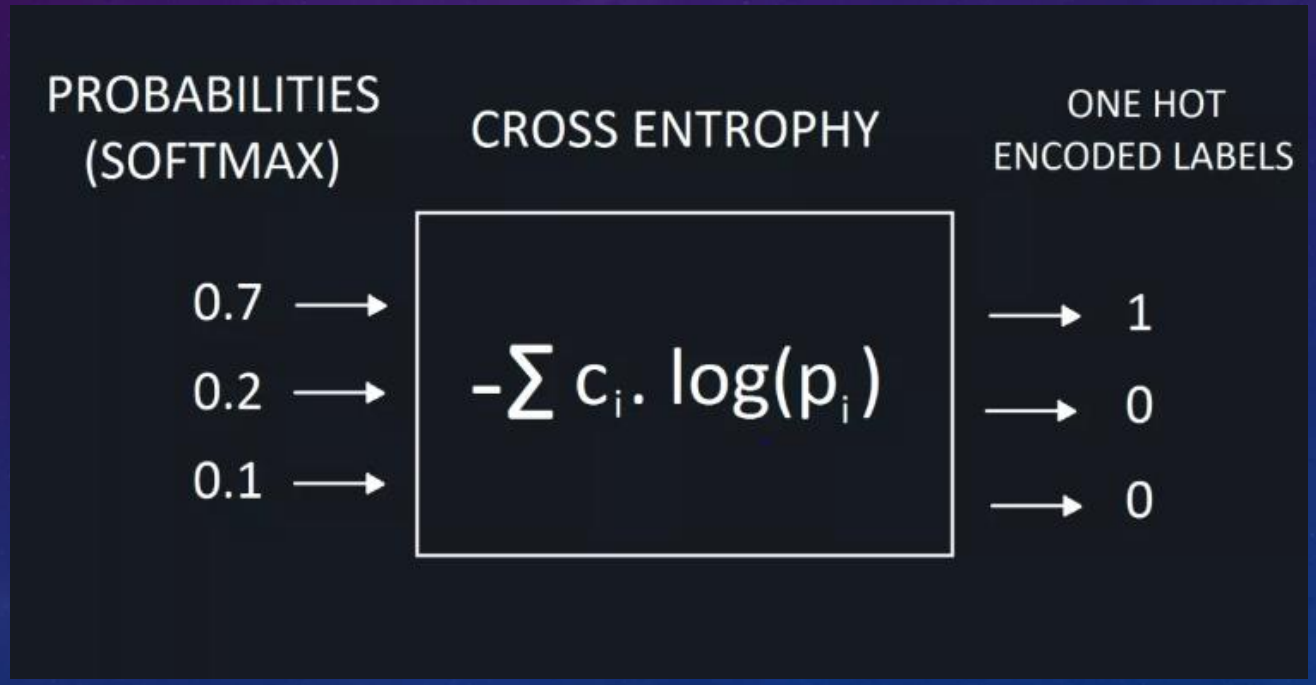
- We apply the standard exponential function to each element z_i of the input vector z and normalize these values by dividing by the sum of all these exponentials; this normalization ensures that the sum of the components of the output vector $\sigma(z)$ is 1.
- Instead of e, a different base $b > 0$ 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 β) yields the expressions:

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

Cross Entropy

Applying softmax function normalizes outputs in scale of [0, 1]. Also, sum of outputs will always be equal to 1 when softmax is applied. After that, applying one hot encoding transforms outputs in binary form.

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

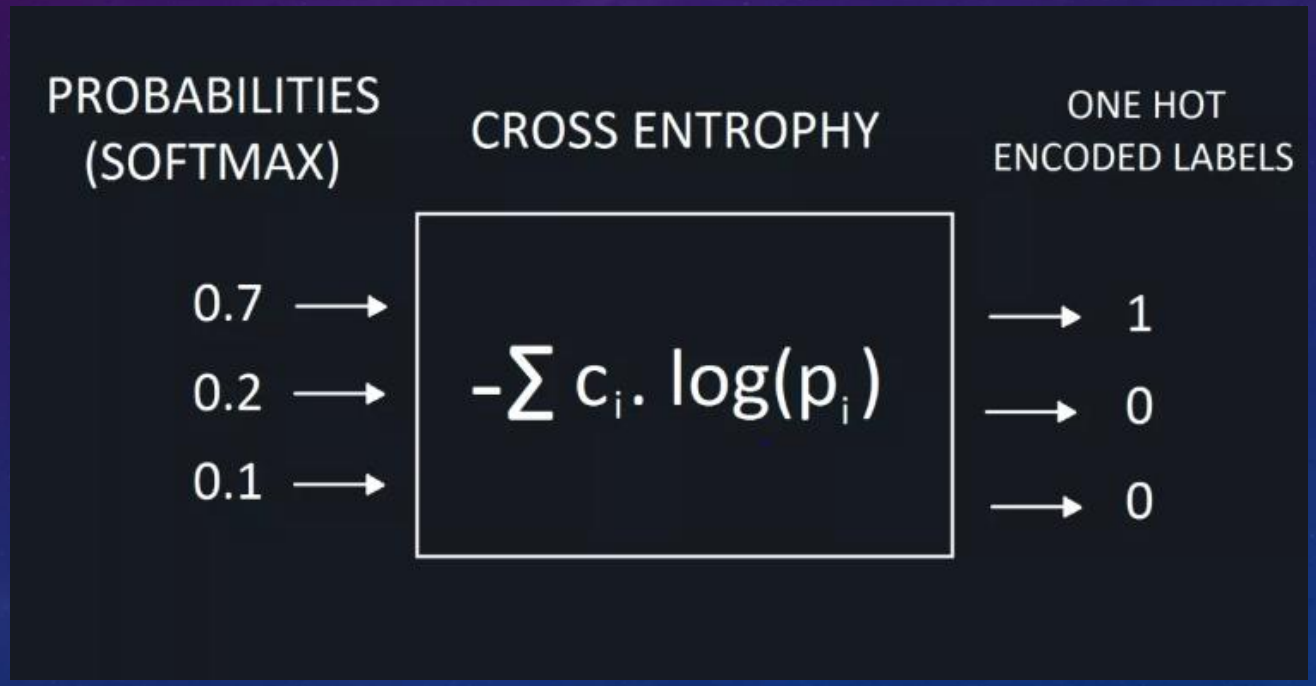


y
(Ground-Truth)

Cross Entropy

That's why, softmax and one hot encoding would be 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

We need to know 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.

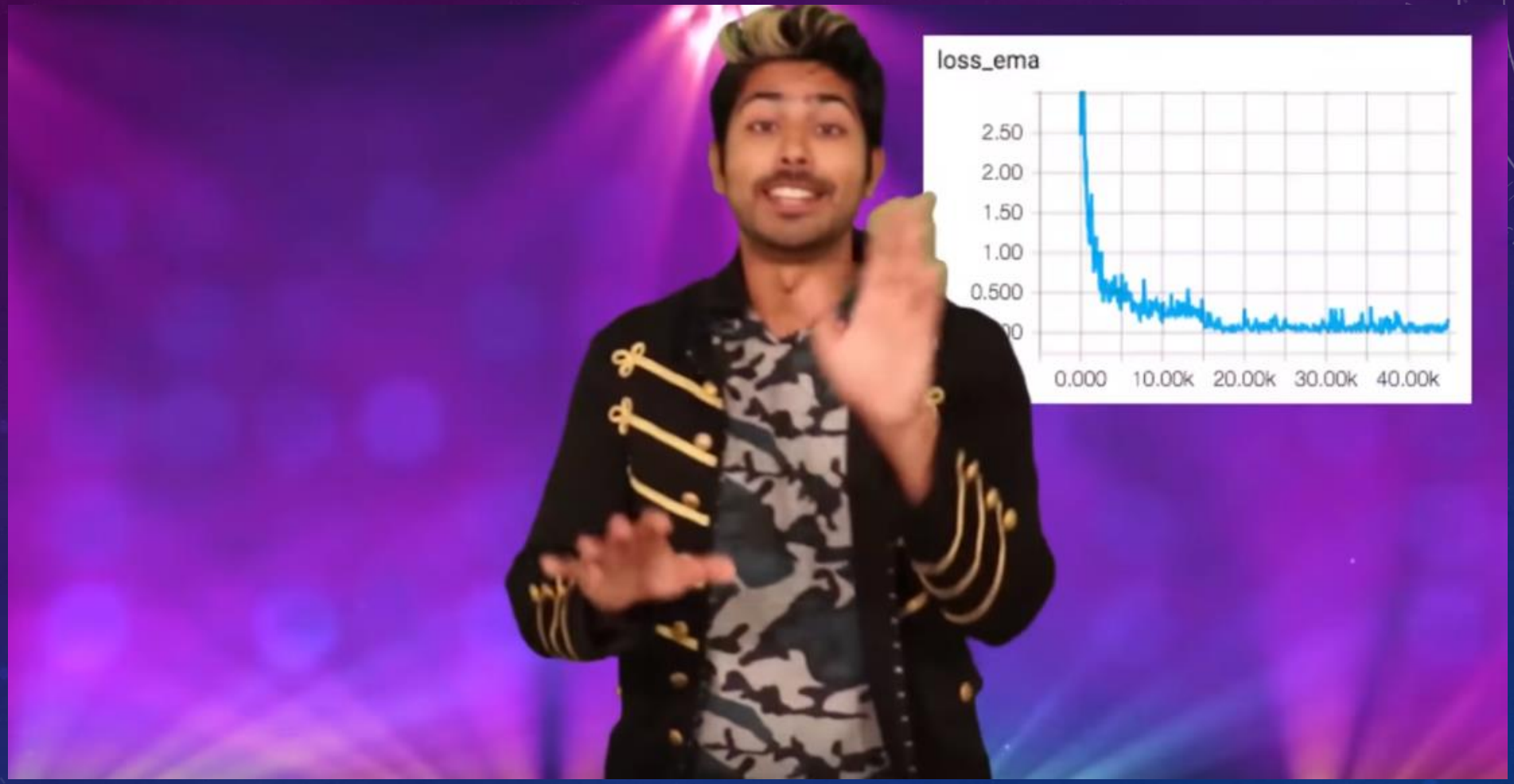
$$E(y, \hat{y}) = -\sum_i y_i log\hat{y}_i - (1 - y_i)\log(1 - \hat{y}_i)$$

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

PS: some sources might define the function a $E(y, \hat{y}) = -\sum_i y_i log\hat{y}_i$, *Resource :*

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

Loss Functions Explained



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

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

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

Classification Error

1/3 = 0.33

1/3 = 0.33

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

What is Backpropagation Really Doing? Deep Learning, Chapter 3

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

Convolutional Neural Networks – Stanford University School of Engineering

Space Dimension Calculation

(31:18 - 47:09)

Pooling

(54:37 – 1:00:15)

Fully Connected Layer

(1:00:15 -1:03:22)

https://www.youtube.com/watch?v=bNb2fEVKeEo&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=5&ab_channel=StanfordUniversitySchoolofEngineering

Training Neural Networks – Stanford University School of Engineering

Activation Functions
(04:51 - 27:11)

Weight Initialization
(34:25 - 48:51)

Batch Normalization
(48:51 -1:04:38)

https://www.youtube.com/watch?v=wEoyxE0GP2M&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=6&ab_channel=StanfordUniversitySchoolofEngineering

CNN Architectures – Stanford University School of Engineering

https://www.youtube.com/watch?v=DAOcjicFr1Y&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=9 [1:17:39]

CNN Architectures – Stanford University School of Engineering

LeNet (02:47 - 03:15)

AlexNet (03:16 – 05:26)

(05:48 – 06:39)

(06:56 – 07:33)

(08:14 – 11:54)

(12:08 – 12:53)

(12:56 – 13:48)

(14:02 – 15:29)

VGGNet (15:30 – 17:10)

(18:15 – 20:58)

(25:05 – 27:36)

(28:28 – 28:37)

GoogLeNet (28:37 – 33:17)

(36:50 – 40:11)

(41:20 – 43:55)

(47:07 – 47:23)

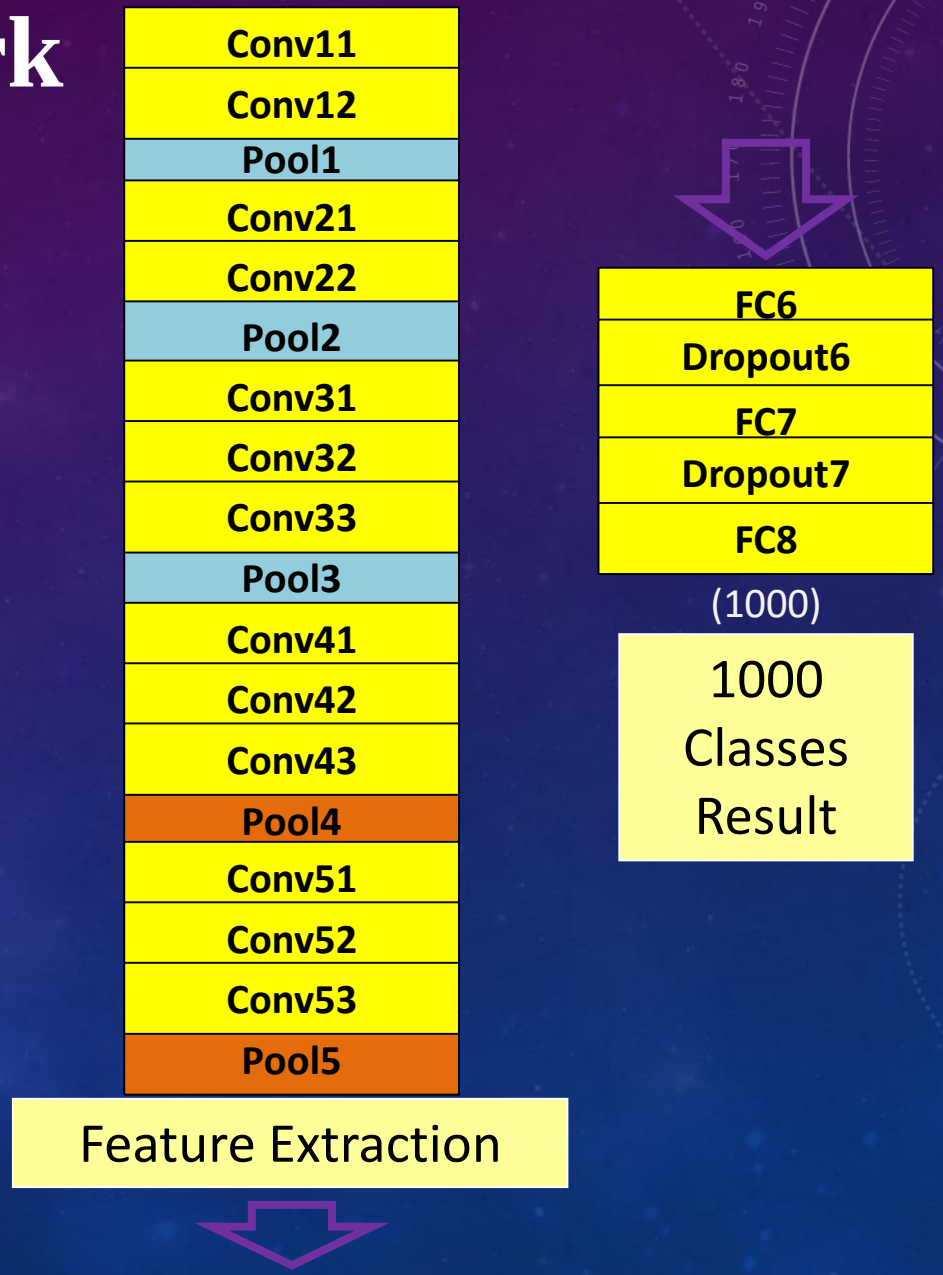
ResNet (47:24 – 53:08)

(53:56 – 55:01)

(58:16 – 1:02:33)

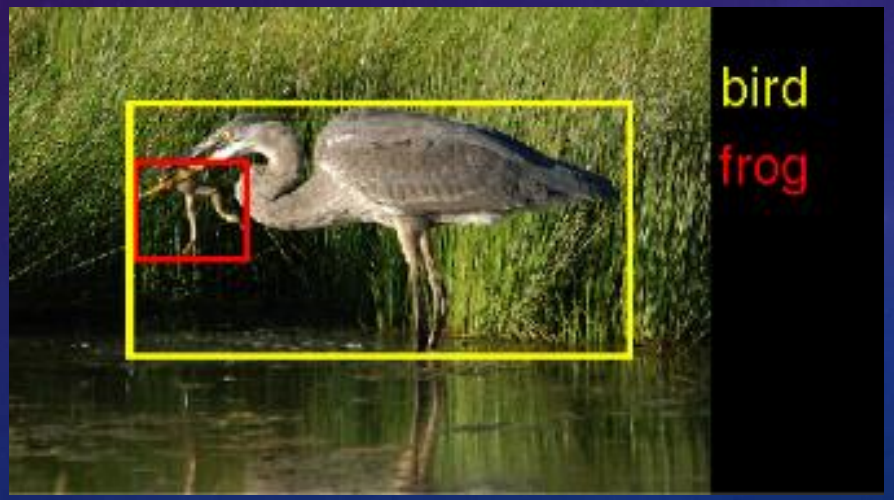
Convolution Neural Network

- ILSVRC-2014 VGG Model
  - Task: 1000 Objects
  - Layer
    - Convolutional layer
    - Max pooling layer
    - Dropout layer
    - Fully connected layer



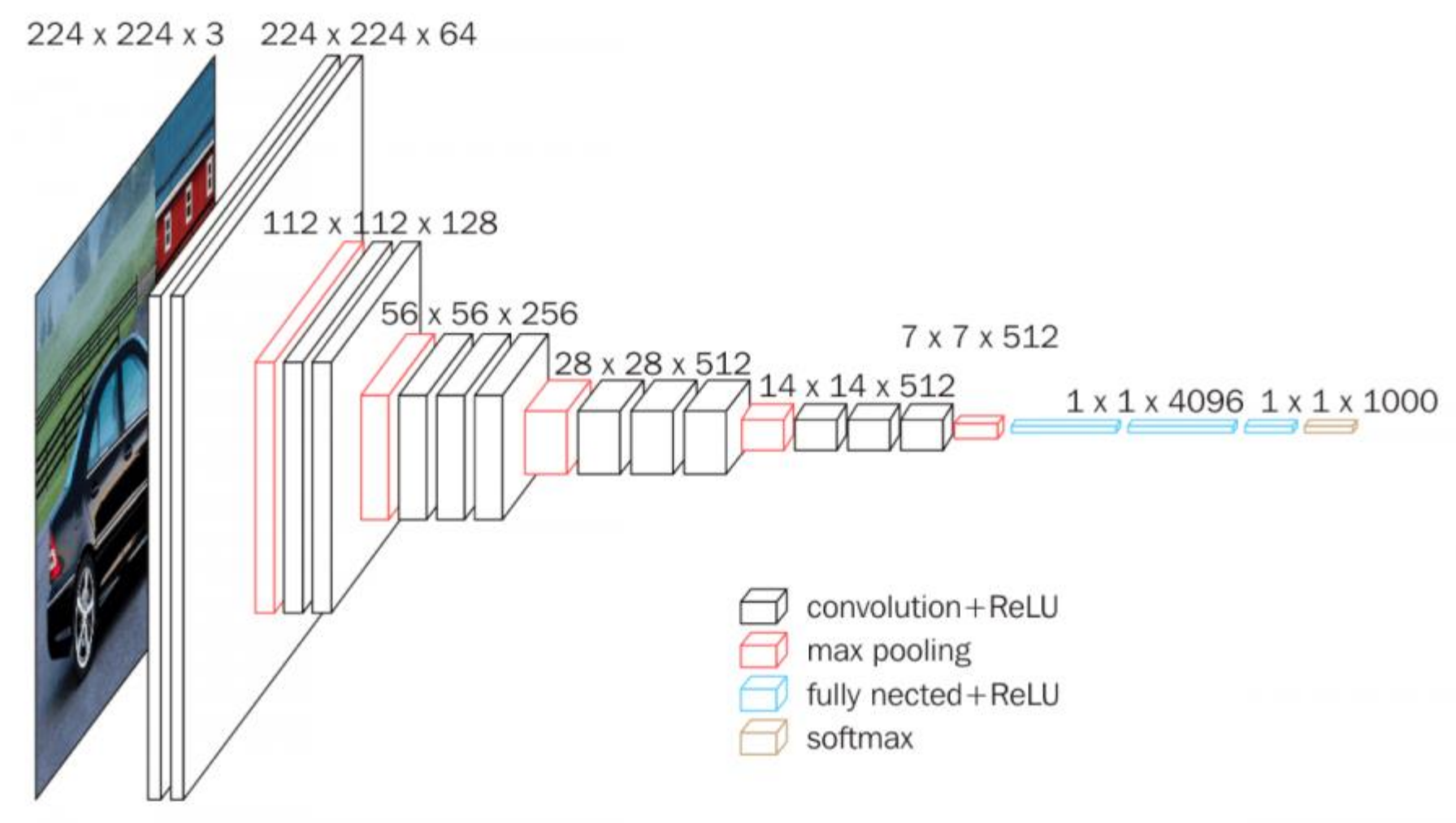
ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014 in the Classification Task

- This challenge evaluates algorithms for object detection and image classification at large scale. This year there will be two competitions:
  - A detection challenge on fully labeled data for 200 categories of objects
  - An image classification plus object localization challenge with 1000 categories.

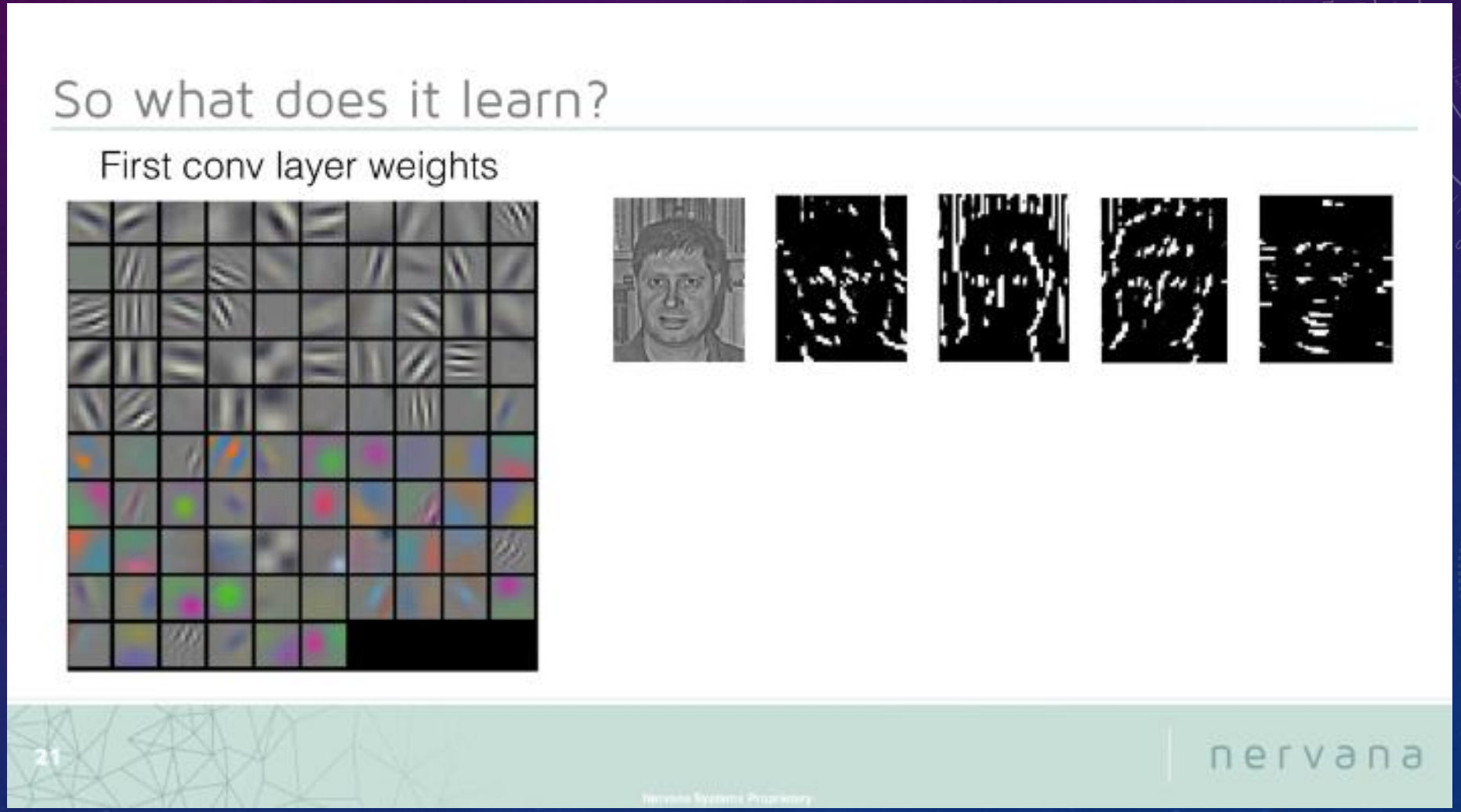


http://image-net.org/challenges/LSVRC/2014/

ILSVRC-2014 VGG Model



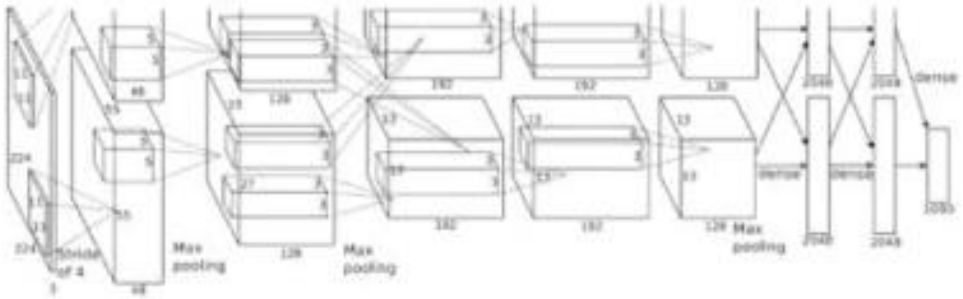
What does Filter Learn?

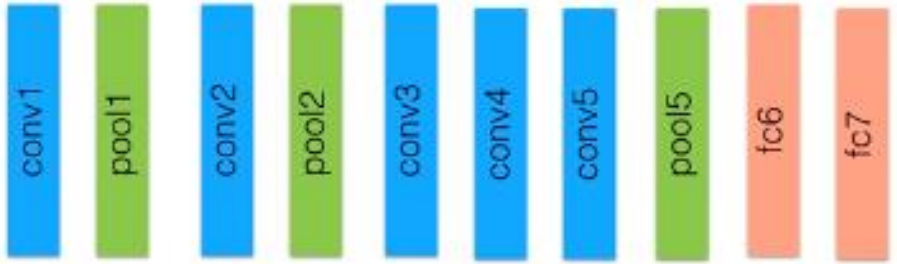


Well-Known Networks

Common Models

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





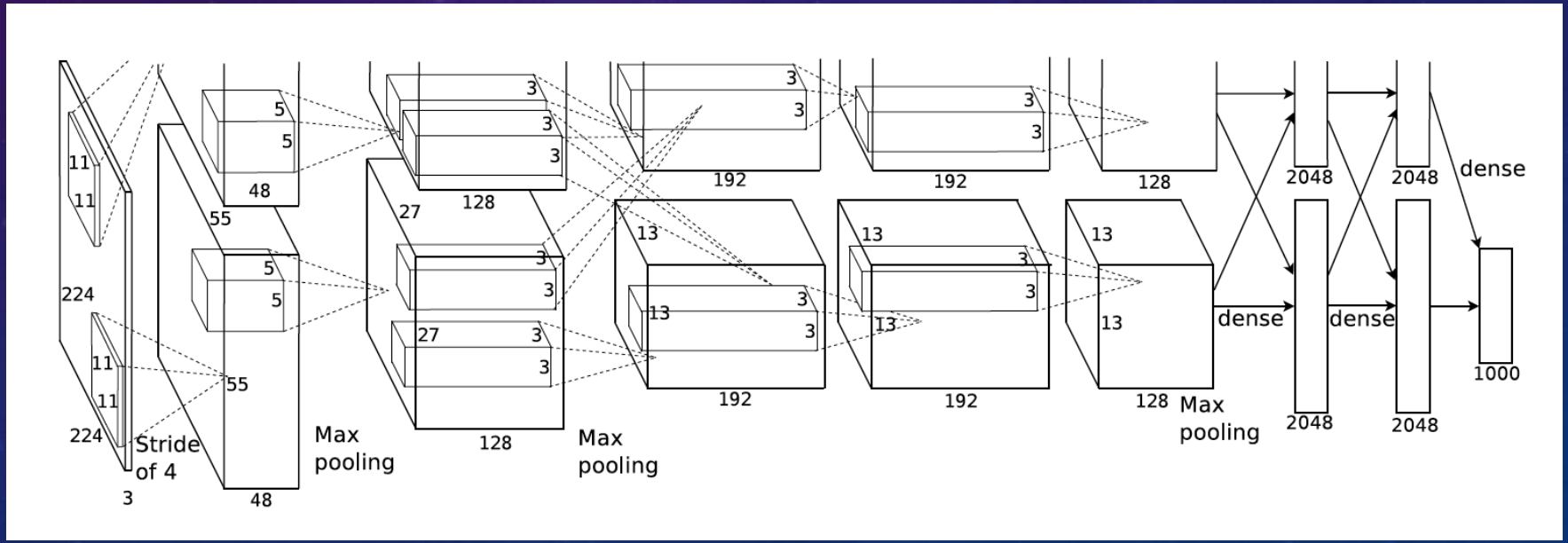
24

nervana

https://www.youtube.com/watch?v=SQ67NBCLV98 [27:48]

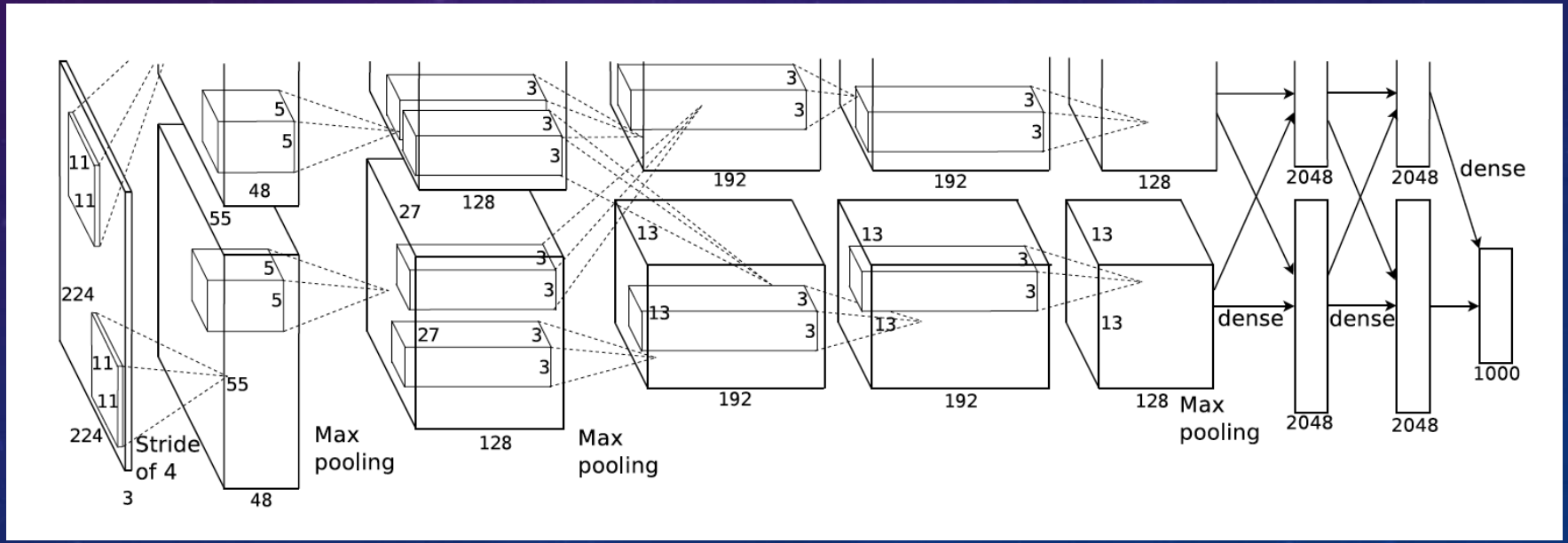
AlexNet

- Use Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
- Use dropout instead of regularization to deal with overfitting. However the training time is doubled with the dropout rate of 0.5.
- Overlap pooling to reduce the size of network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively.



AlexNet

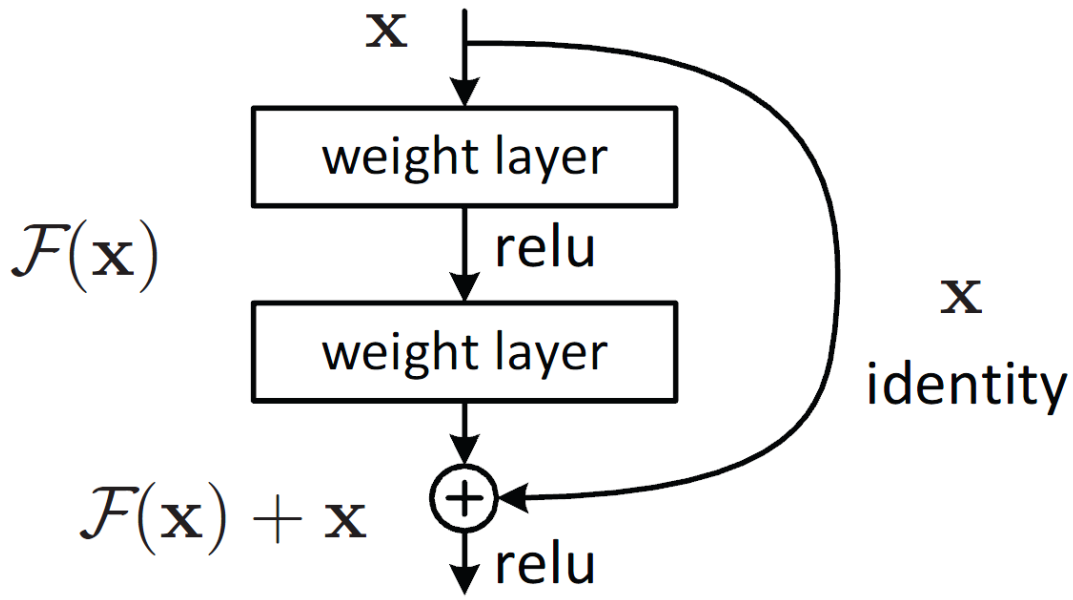
- Copy convolution layers into different GPUs; Distribute the fully connected layers into different GPUs.
- Feed one batch of training data into convolutional layers for every GPU (Data Parallel).
- Feed the results of convolutional layers into the distributed fully connected layers batch by batch (Model Parallel) When the last step is done for every GPU. Backpropagate gradients batch by batch and synchronize the weights of the convolutional layers.



Well-Known Networks

Common Models

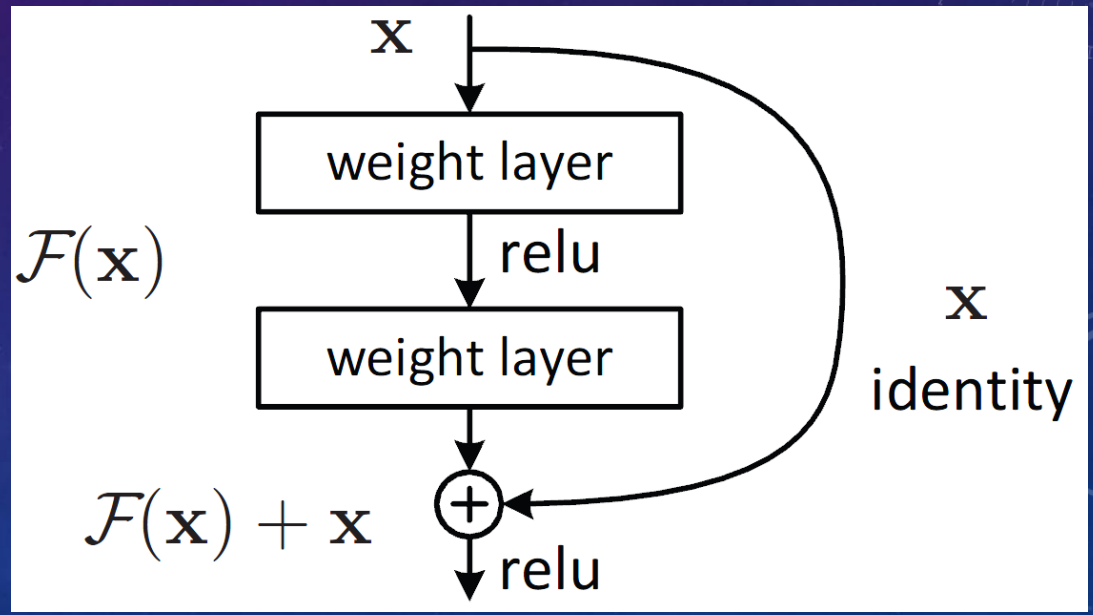
- AlexNet (ILSVRC 2012 winner)
- ZF Net (2013 winner)
- GoogLeNet (2014 winner)
- VGG (2014 runner-up)
- **ResNet (2015 winner)**



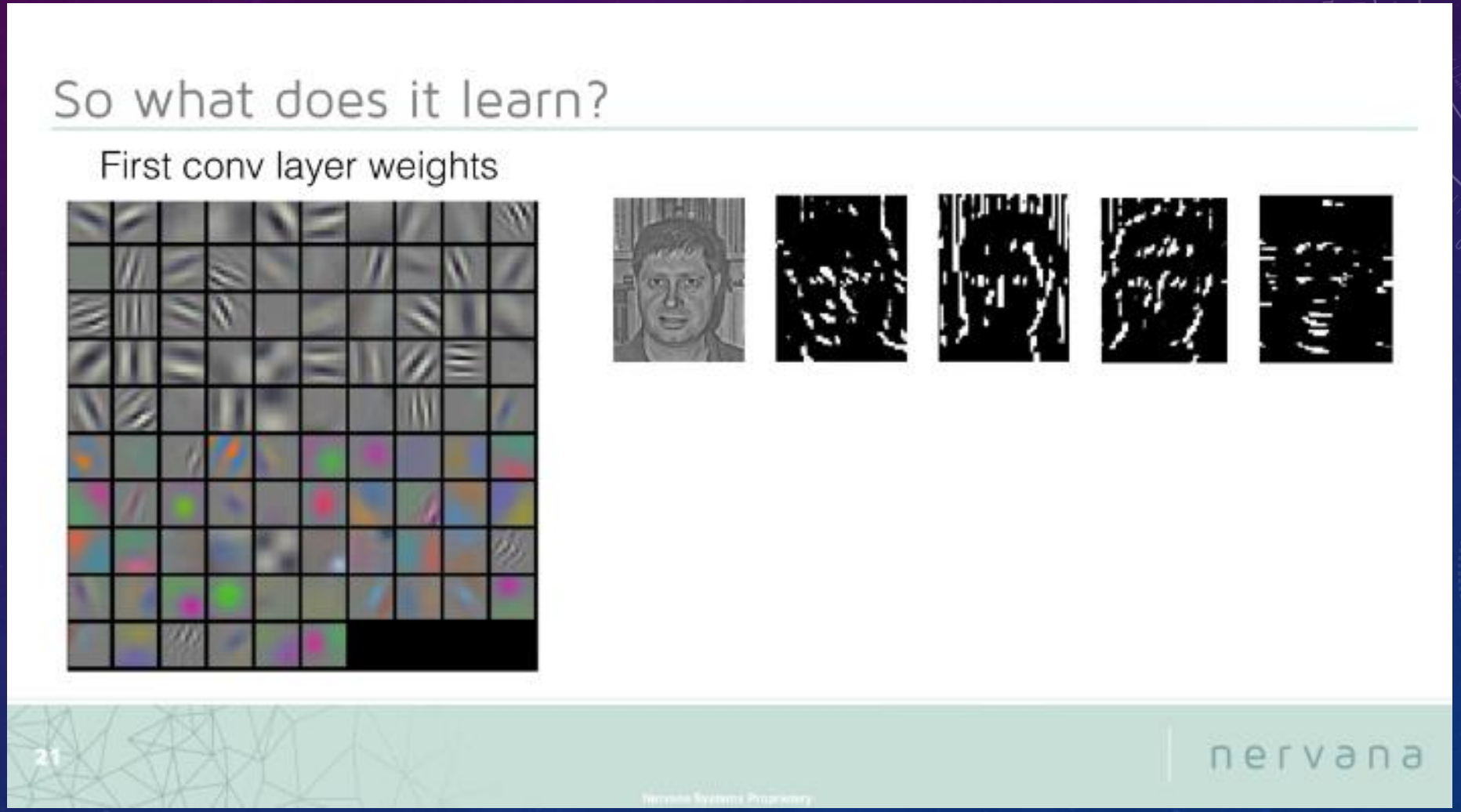
nervana

ResNet

- Since AlexNet, the state-of-the-art CNN architecture is going deeper and deeper.
- However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem
- The core idea of ResNet is introducing a so-called “identity shortcut connection” that skips one or more layers, as shown in the following figure:



What does Filter Learn?



See Coding Manual, Chapter 2, Page 7.

Exercise 2-3: Feature Map Visualization

- Pip install opencv-python in your pytorch environment.
- Please download the “2-3_Feature_map_visualization” from the Clouds, which is built on the VGG-16 trained on the ImageNet.
- Choose your own images from Internet.
- Compare the feature maps extracted from layer 5 and observe the size and dimension of the feature maps.

Please write down result and your code in MS Word and upload to the Moodle

See Coding Manual, Chapter 2, Page 11.

Exercise 2-4: Feature Map Visualization

- Pip install opencv-python in your pytorch environment.
- Please download the “2-3_Feature_map_visualization” on Clouds and choose your own images from Internet.
- Compare the probability of the images that contain two classes and different variations (pose, occlusion, age).
- Please write down result and your code in MS Word to Moodle




Verification - Euclidean Distance

- In mathematics, the Euclidean distance or Euclidean metric is the "ordinary" straight-line distance between two points in Euclidean space.

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

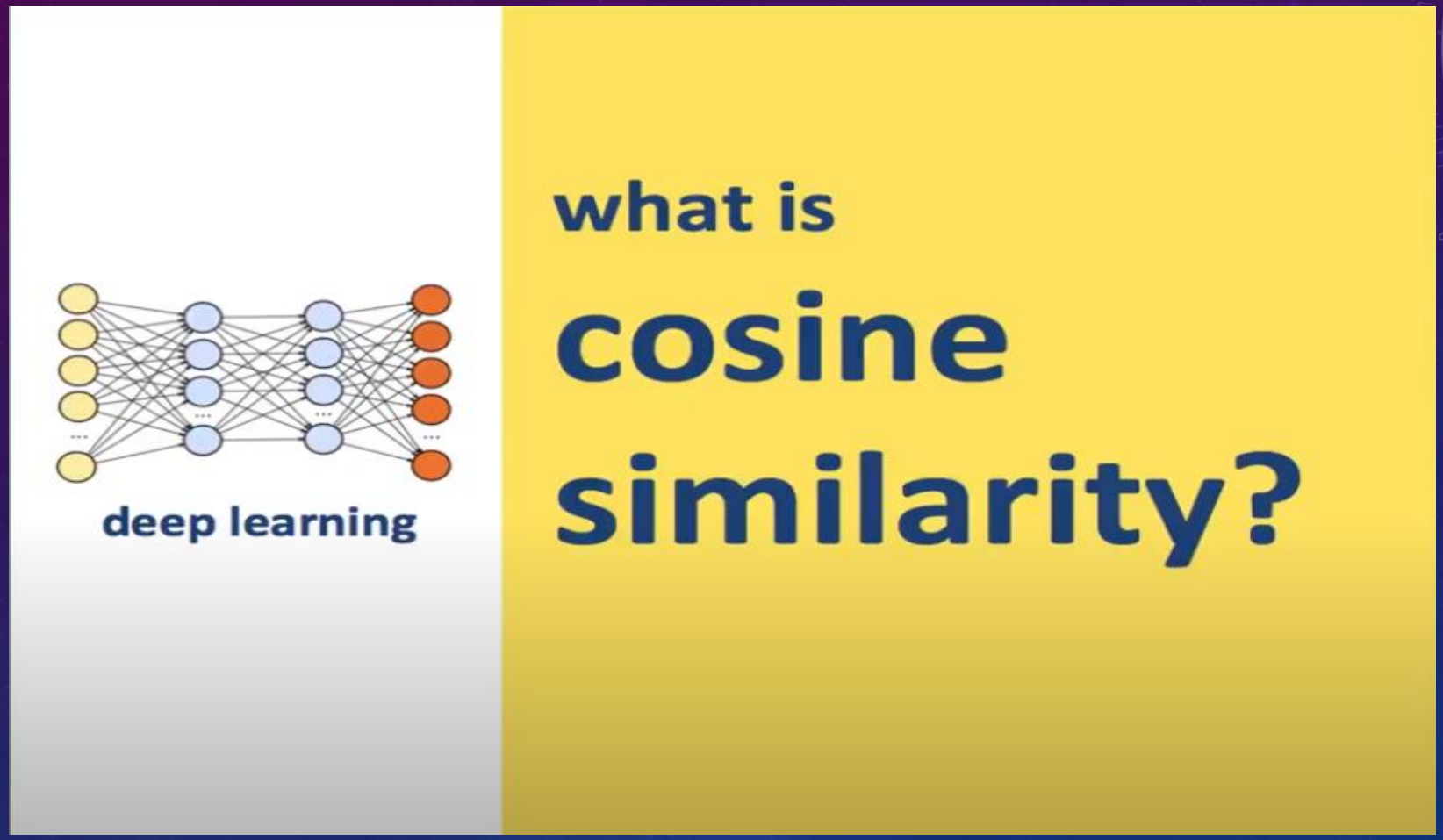
$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Verification - Cosine Similarity

- Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space.
- The cosine value of a 0 degree angle is 1, and the cosine value of any other angle is not greater than 1; and its minimum value is -1.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

How to measure similarity in vector space (cosine similarity)



https://www.youtube.com/watch?v=xY3jrJdpuQg&t=34s&ab_channel=MinsukHeo%ED%97%88%EB%AF%BC%EC%84%9D [4:16]

See Coding Manual, Chapter 2, Page 12.

Exercise 2-5: Feature Overview

- Pip install opencv-python in your pytorch environment.
- Please download the “2-5_Feature_Overview” on Clouds and choose your own images from Internet.
- Step 1: Compare different objects and the same objects by choosing images from Internet.
- Step 2: Compare the results of different crop sizes of the same image(e.g. Dog image only crop the tail)
- Upload your observations, comments and your code to Moodle in a docx.

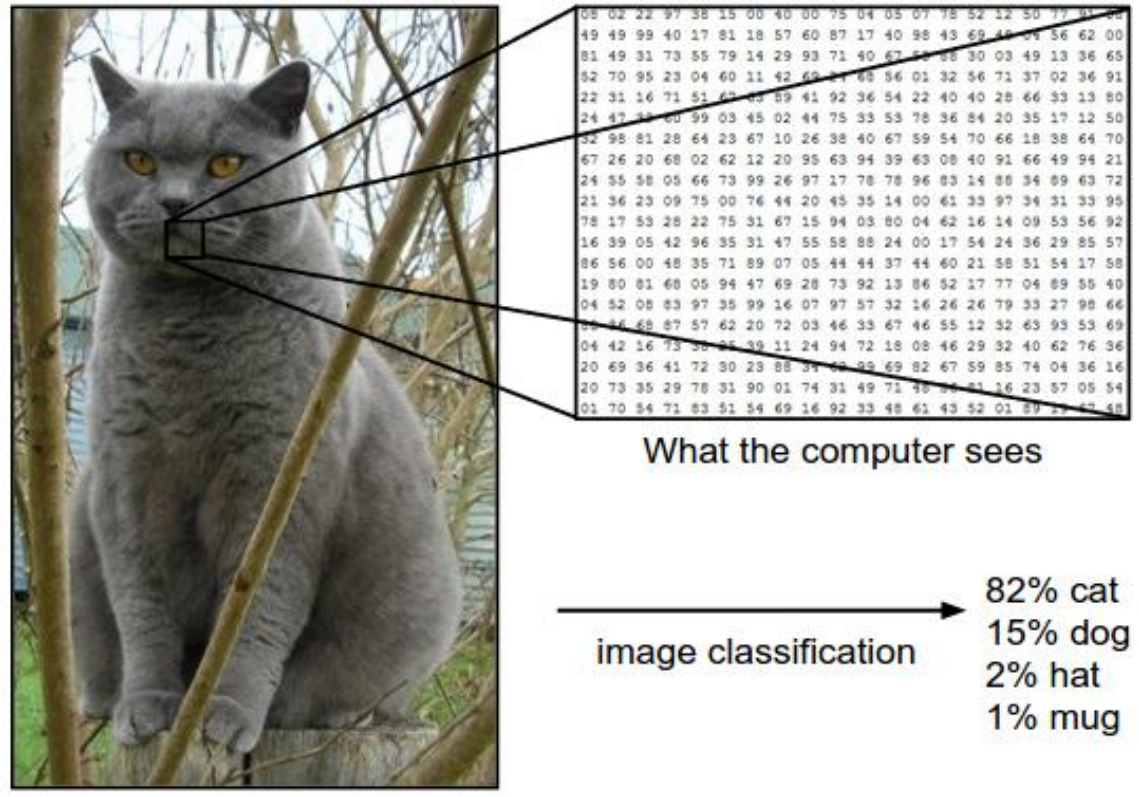
Image Classification

How to Make an Image Classifier – Intro to Deep Learning #6

https://www.youtube.com/watch?v=cAICT4Al5Ow[8:44]

Image Classification

The task in Image Classification is to predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image.



Images are 3-dimensional arrays of integers from 0 to 255, of size Width x Height x 3.
(The 3 represents the three color channels Red, Green, Blue.)

Image Classification

Challenges:

1. Viewpoint variation.
2. Scale variation.
3. Deformation.
4. Occlusion.
5. Illumination conditions.
6. Background clutter.
7. Intra-class variation.

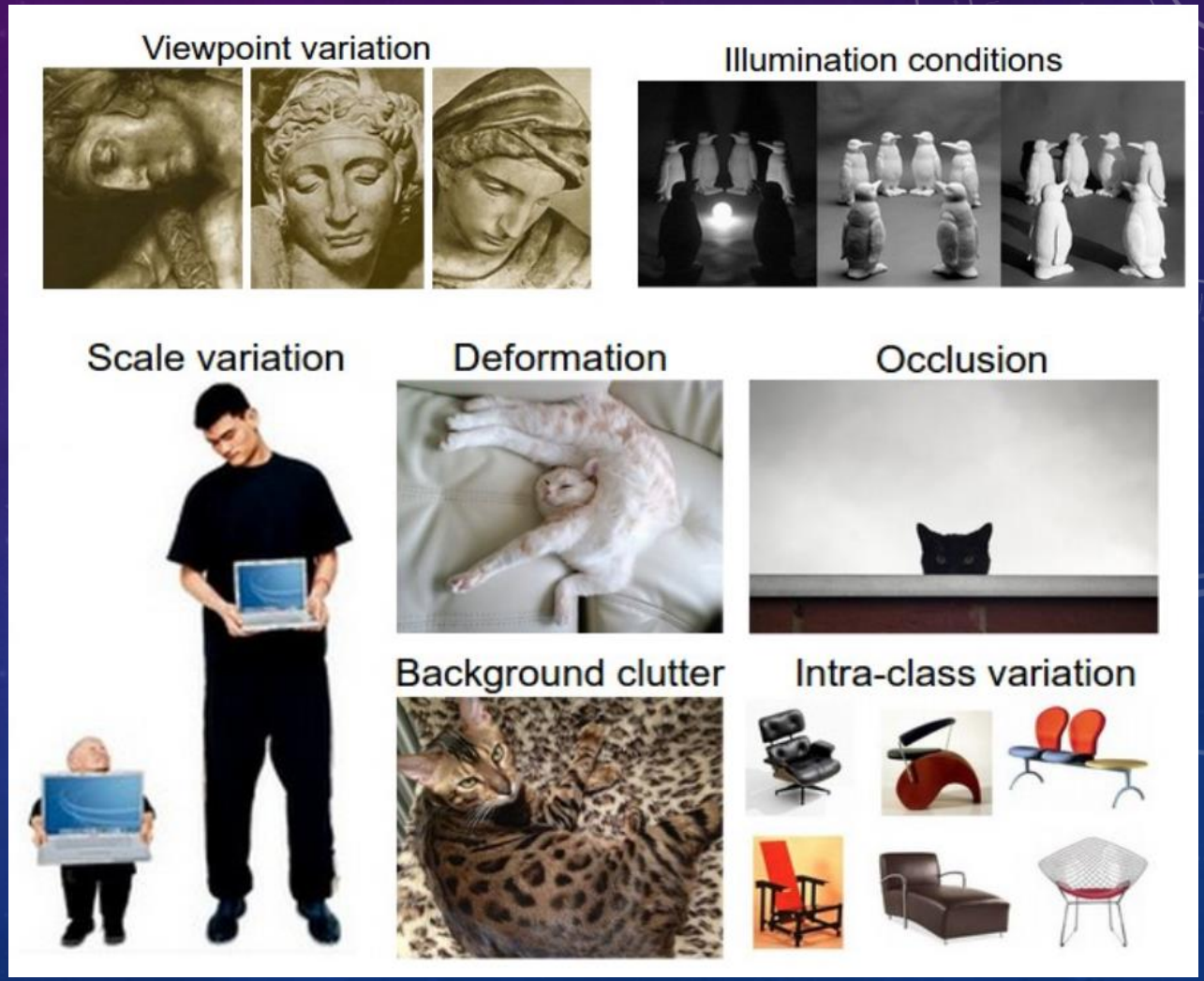


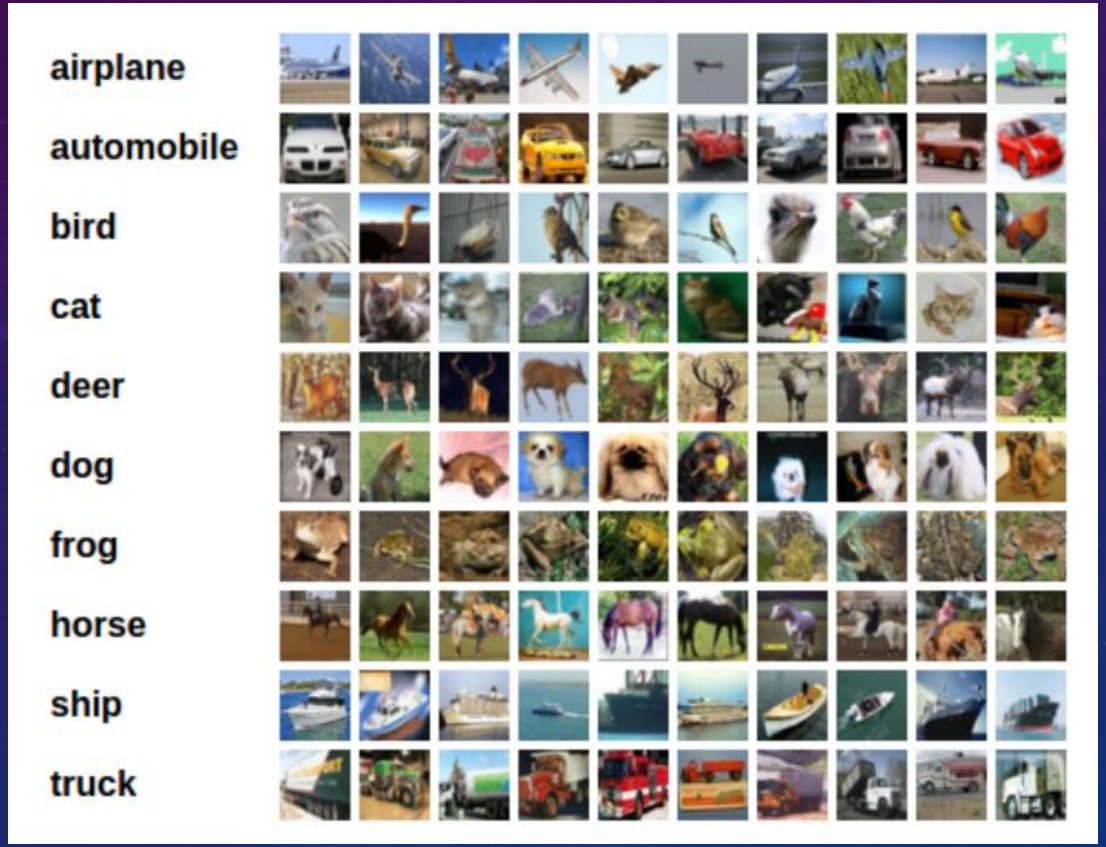
Image Classification

Data-driven approach :

- We’re going to provide the computer with many examples of each class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class.
- This approach is referred to as a **data-driven approach**, since it relies on first accumulating a training dataset of labeled images.

Training A Classifier

For this section, we will use the CIFAR10 dataset.



The images in CIFAR-10 are of size 3x32x32

Training A Classifier

We will do the following steps:

1. Load and normalizing the CIFAR10 training and test datasets using torchvision
2. Define a Convolutional Neural Network
3. Define a loss function
4. Train the network on the training data
5. Test the network on the test data

See Coding Manual, Chapter 2, Page 17.

Exercise 2-6 – Build A Classifier

- Please install matplotlib with the command “pip install matplotlib”
- Download 2-6_CIFAR10.py and do the following steps:
- Step 1: Epoch
- Step 2: Optimizer learning rate
- Please upload your result in MS Word to Moodle

Exercise 2-7

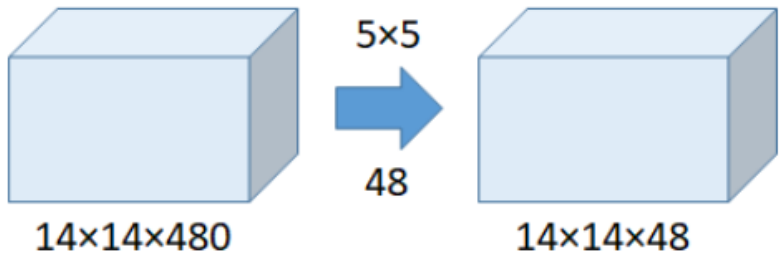
- Download and modify 2-6_CIFAR10.py to train MNIST dataset with following steps:
  1. Download the MNIST Dataset from the following website
     Hint: https://pytorch.org/docs/stable/torchvision/datasets.html
  2. Since MNIST is a handwritten digits dataset, please change the name of classes.
  3. Since the size of images in MNIST dataset are (1,28,28), please check the following points (Input size of first convolution layer and fully-connected layer)
- Please write down the differences in the settings and the results to a Words doc, and paste your code, and upload to the Moodle.

GoogleNet

- 1×1 Convolution at the middle of the network
  - In GoogLeNet, 1×1 convolution is used as a dimension reduction module to reduce the computation.
- Global average pooling is used at the end of the network instead of using fully connected layers
- Inception module

GoogleNet

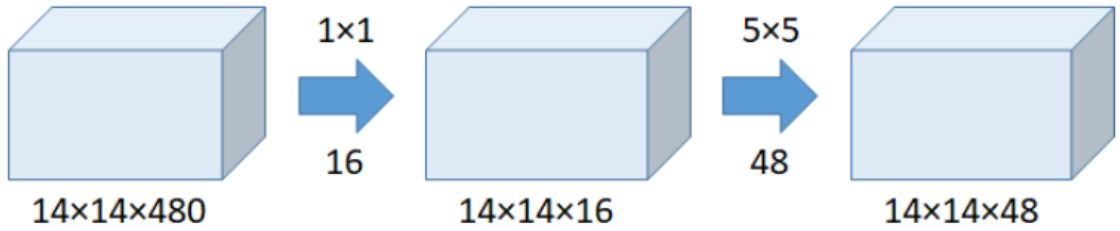
- 1×1 Convolution at the middle of the network
  - In GoogLeNet, 1×1 convolution is used as a dimension reduction module to reduce the computation.



Without the Use of 1×1 Convolution

Number of operations = $(14\times14\times48)\times(5\times5\times480) = 112.9M$

With the use of 1×1 convolution:



With the Use of 1×1 Convolution

Number of operations for $1\times1 = (14\times14\times16)\times(1\times1\times480) = 1.5M$

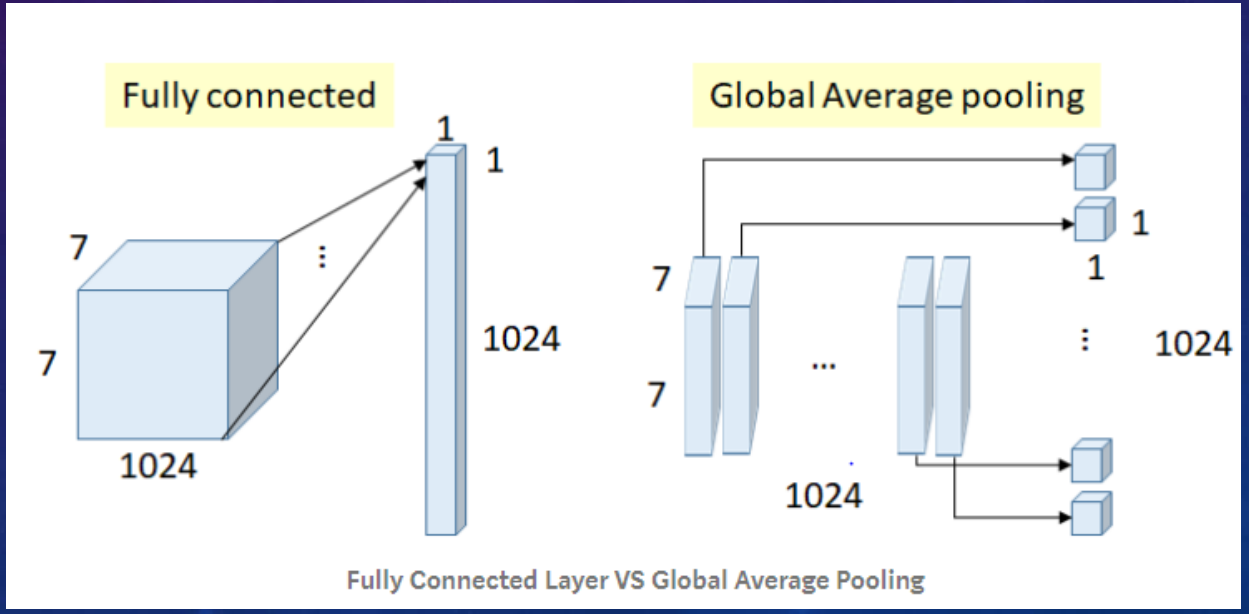
Number of operations for $5\times5 = (14\times14\times48)\times(5\times5\times16) = 3.8M$

Total number of operations = 1.5M + 3.8M = 5.3M

which is much much smaller than 112.9M !!!!!!!!!!!!!!

GoogleNet

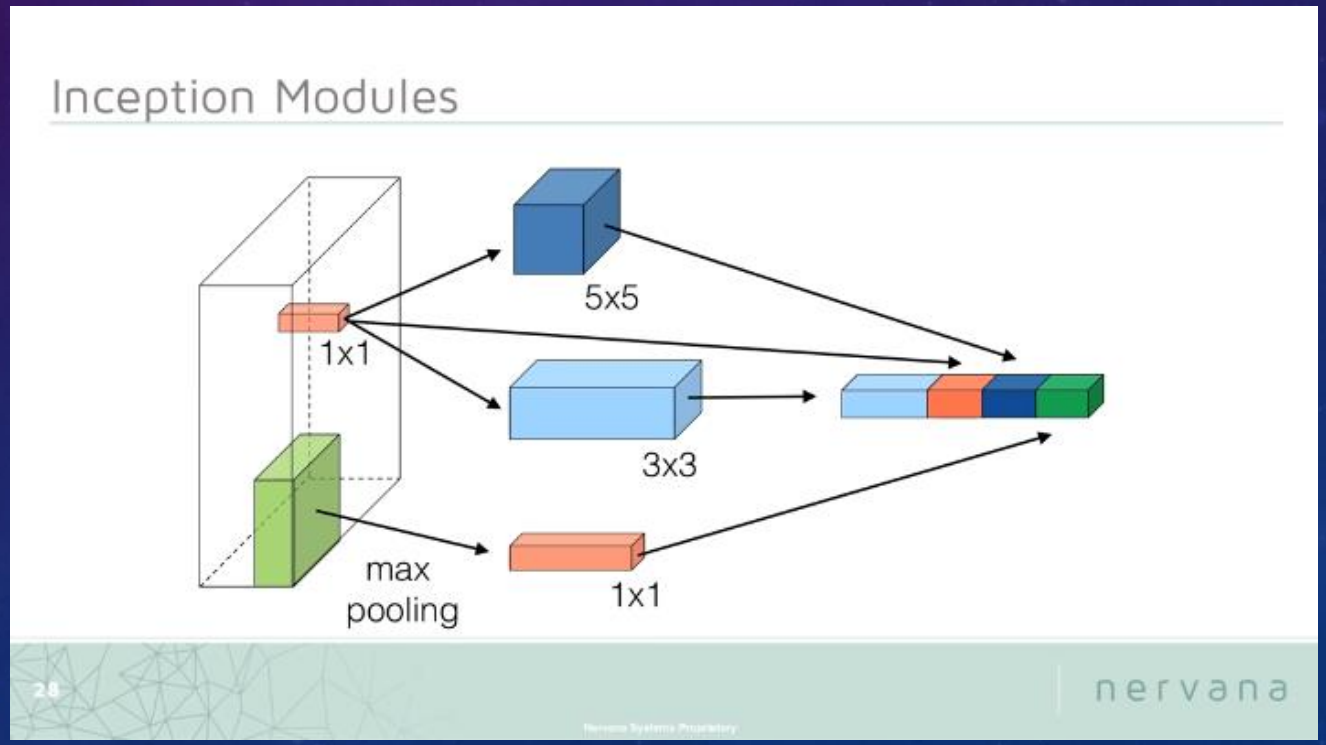
- Number of weights (connections) above = 7×7×1024×1024 = 51.3M
- In GoogleNet, global average pooling is used nearly at the end of network by averaging each feature map from 7×7 to 1×1, as in the figure above.
- Number of weights = 0
- And authors found that a move from FC layers to average pooling improved the top-1 accuracy by about 0.6%.



Fully Connected Layer VS Global Average Pooling

GoogleNet

- 1×1 conv, 3×3 conv, 5×5 conv, and 3×3 max pooling are done altogether for the previous input, and stack together again at output. When image's coming in, different sizes of convolutions as well as max pooling are tried. Then different kinds of features are extracted.
- After that, all feature maps at different paths are concatenated together as the input of the next module.



See Coding Manual, Chapter 2, Page 28.

Exercise 2-8: Result Compare on CIFAR10

- Please download 2-8 CIFAR10_compare.zip from moodle
- Modify test.py to use ResNet18, GoogleNet and VGG16 for testing CIFAR10 Dataset
- Modify test.py to use 2-6_CIFAR10.py checkpoint for testing CIFAR10 Dataset

Upload your observations, comments and your code to Moodle in a docx.