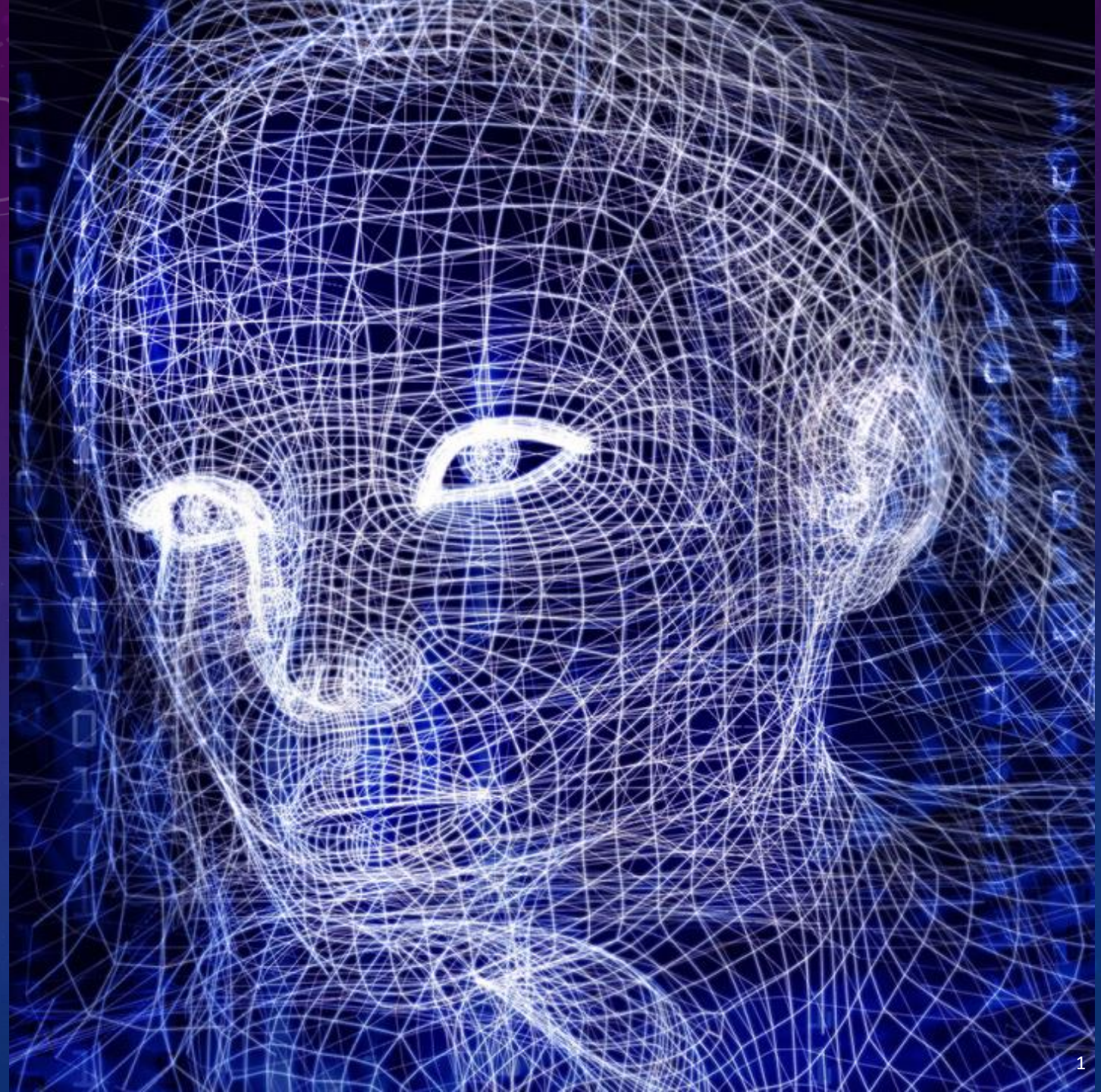
LECTURE SERIES FOR COMPUTER VISION

INTRODUCTION TO DEEP LEARNING

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology



Convolutional Neural Networks

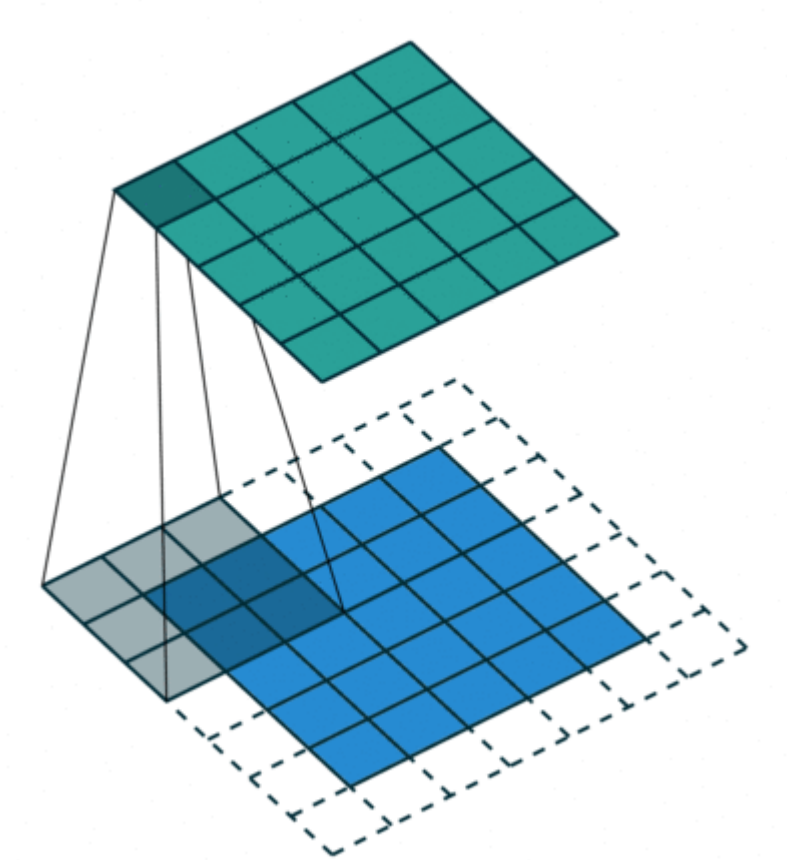
1) Convolution、Padding
2) Pooling
3) Activation Function
4) Weight Initialization
5) Loss Function
6) Back Propagation
7) Architecture Introduction
8) Feature Visualization
9) Image Classification

Convolutional Neural Networks (CNNs) Explained

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

Convolutions

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

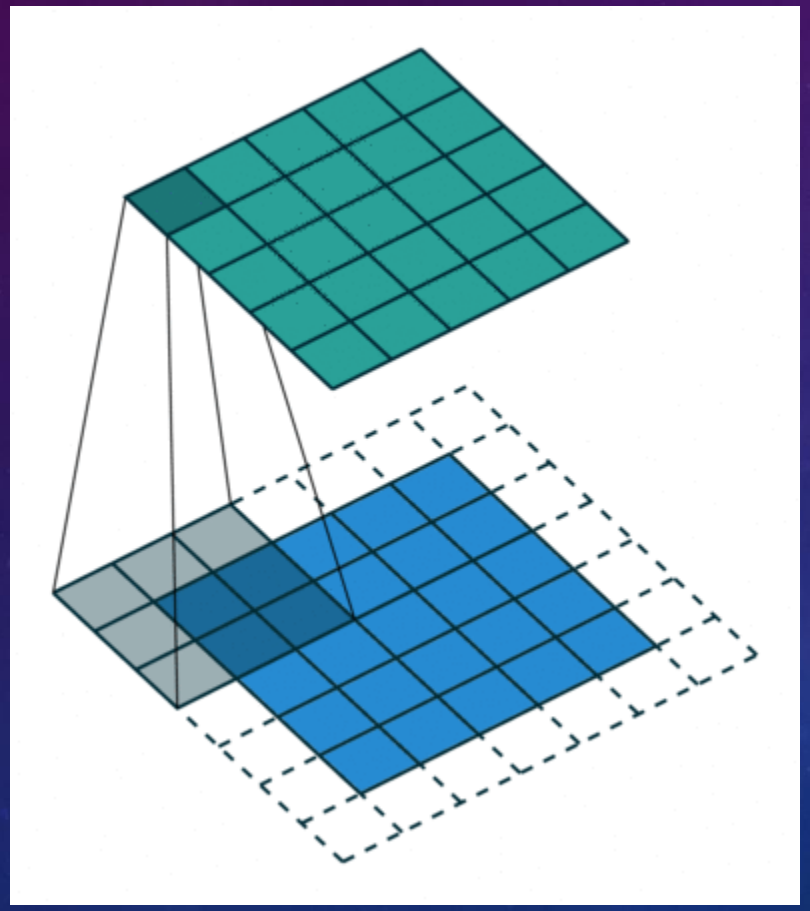


- **Kernel Size:** The kernel size defines the field of view of the convolution. A common choice for 2D is 3—that is 3x3 pixels.
- **Stride**: The stride defines the step size of the kernel when traversing the image. While its default is usually 1, but it can be 2 or other step sizes.

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.



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

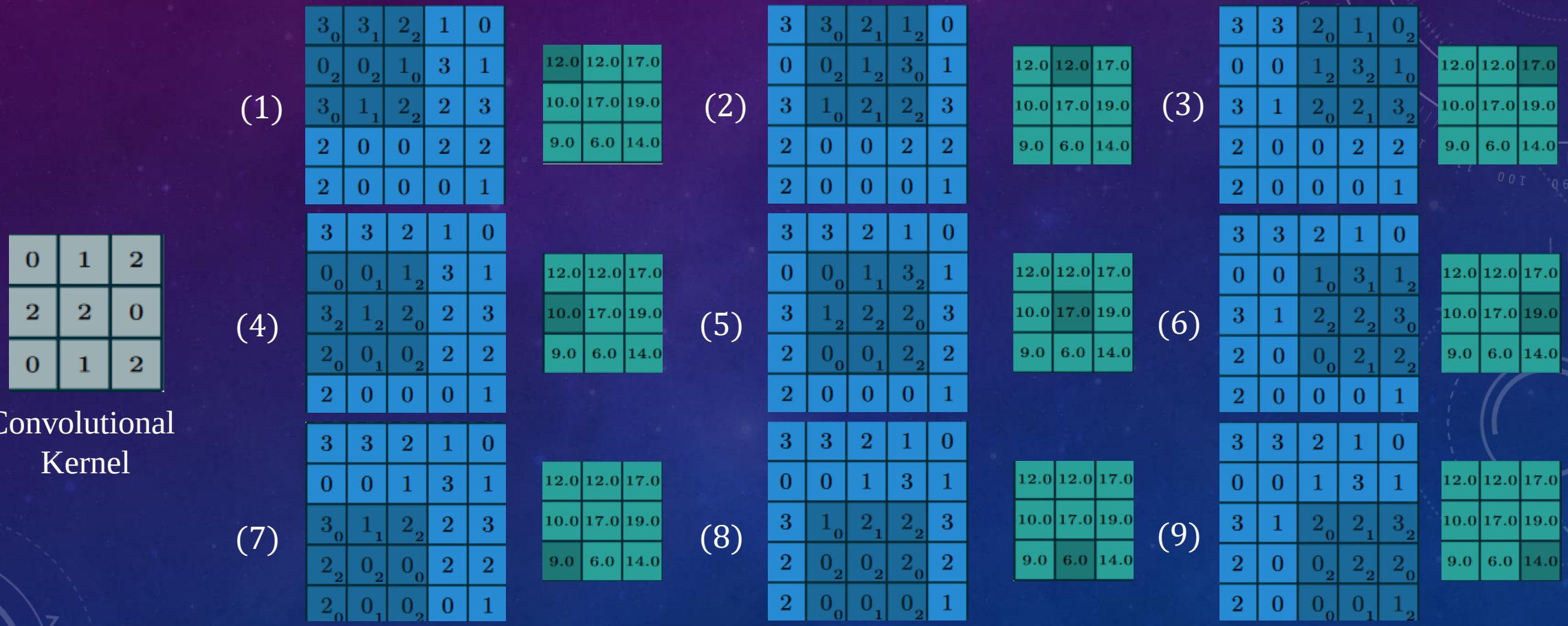
- **Padding**: The padding defines how the border of a sample is handled. A (half) padded convolution will keep the spatial output dimensions equal to the input, whereas unpadded convolutions will crop away some of the borders if the kernel is larger than 1.
- **Input & Output Channels**: A convolutional 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.

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

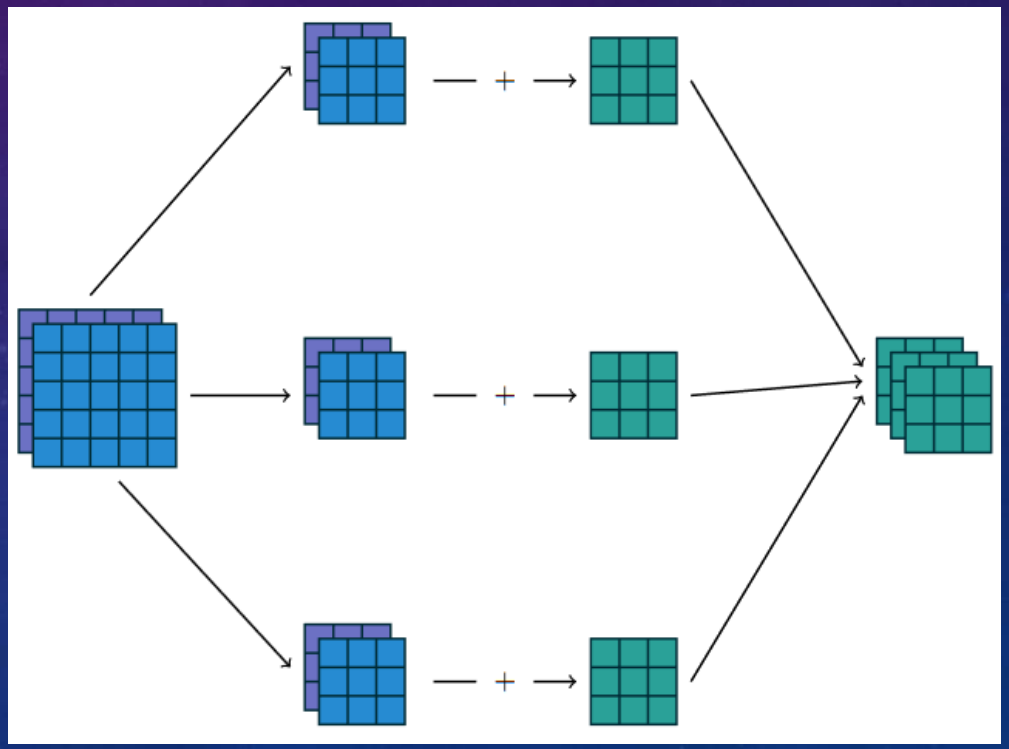
Discrete Convolutions

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 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

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).

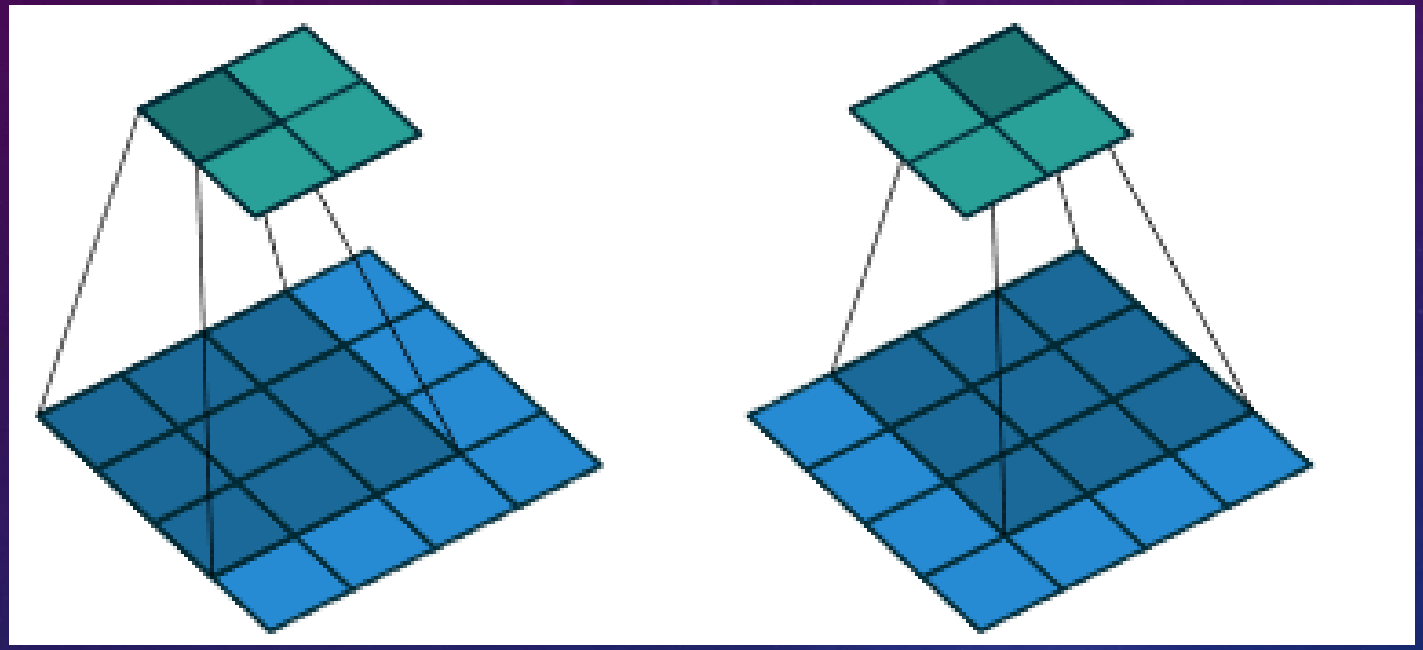
Convolution Arithmetic

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 there sults outlined here also generalize to the N-D and non-square cases.

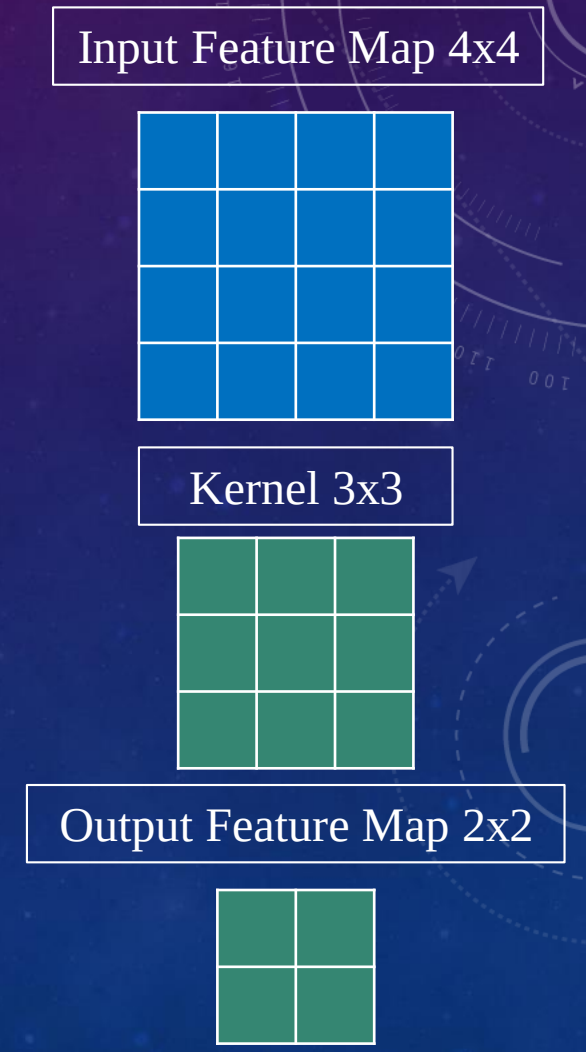
Example: No Zero Padding, Unit Strides



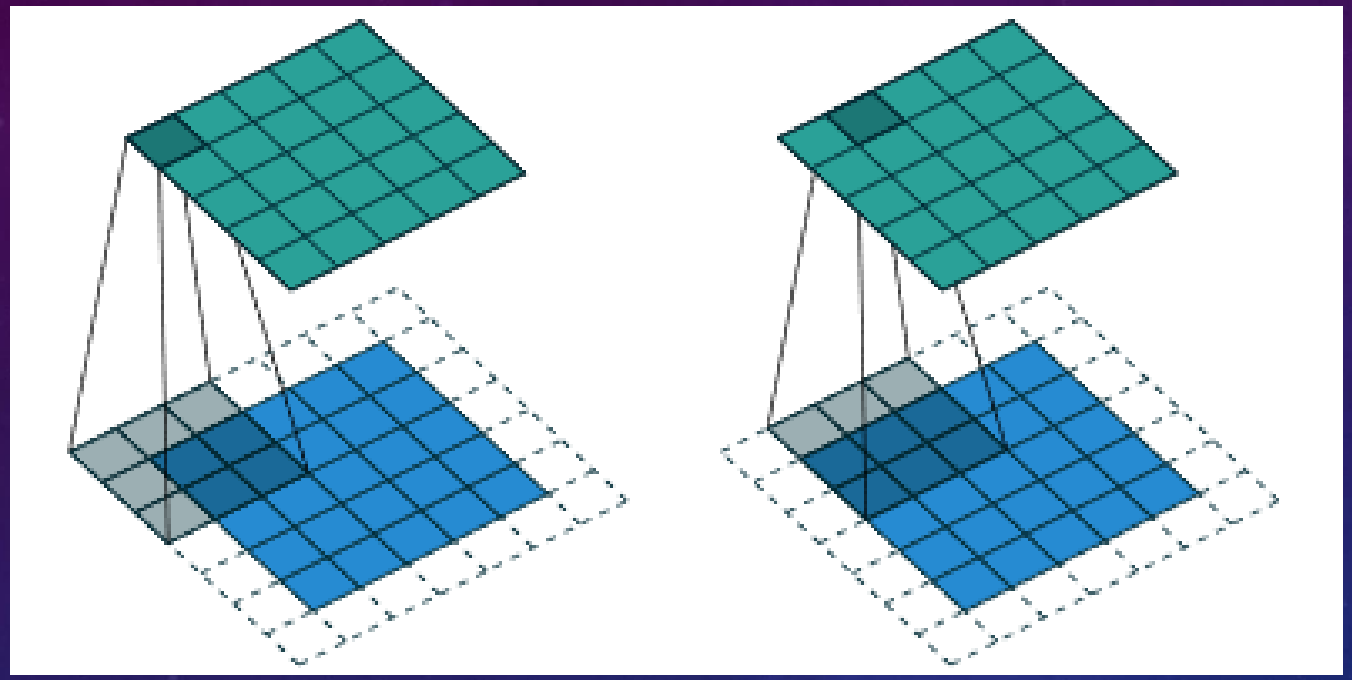
(i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$)

Relationship 1. *For any i and k, and for $s = 1$ and $p = 0$,*

$$o = (i - k) + 1$$



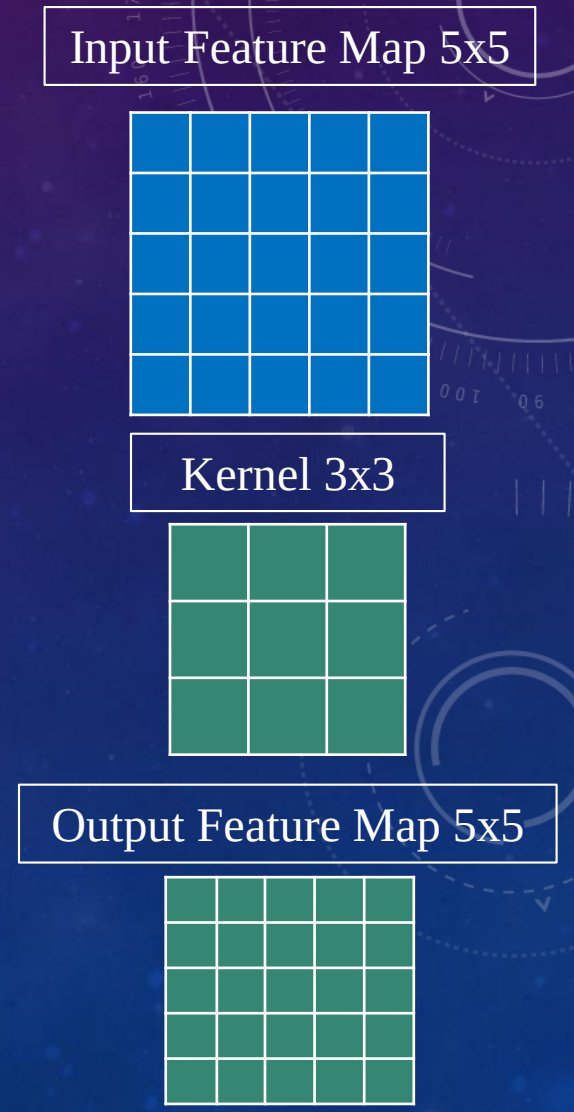
Example: Zero Padding, Unit Strides



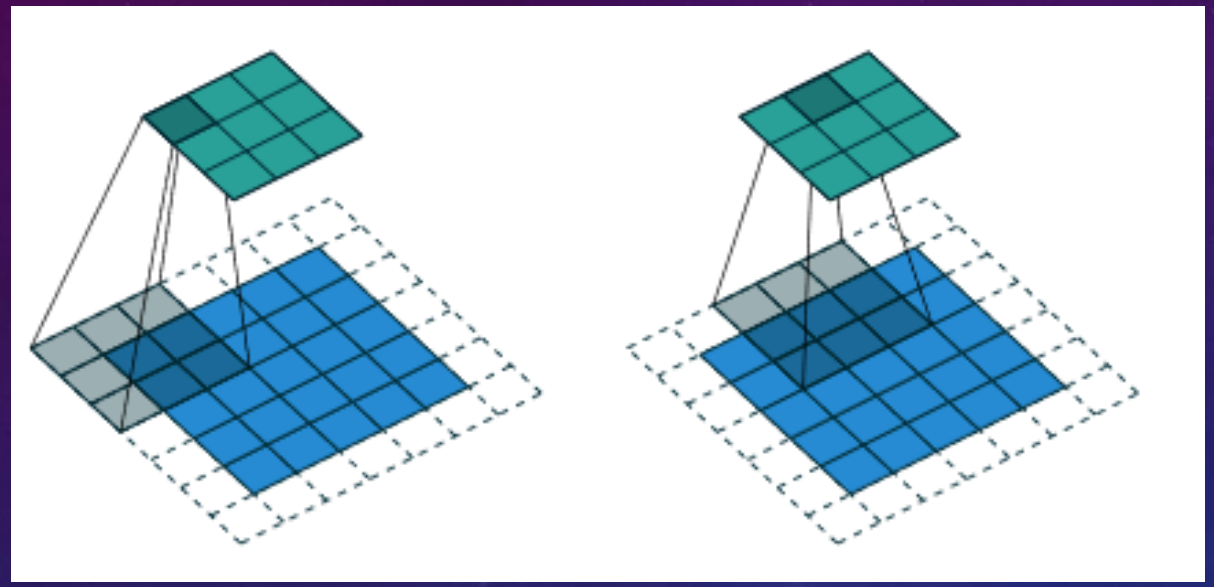
(i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$)

Relationship 3. *For any i and k odd* $(k = 2n + 1, n \in \mathbb{N})$, $s = 1$ *and* $p = [k/2] = n$

$$o = i + 2[k/2] - (k - 1)$$
$$= i + 2n - 2n$$
$$= i$$



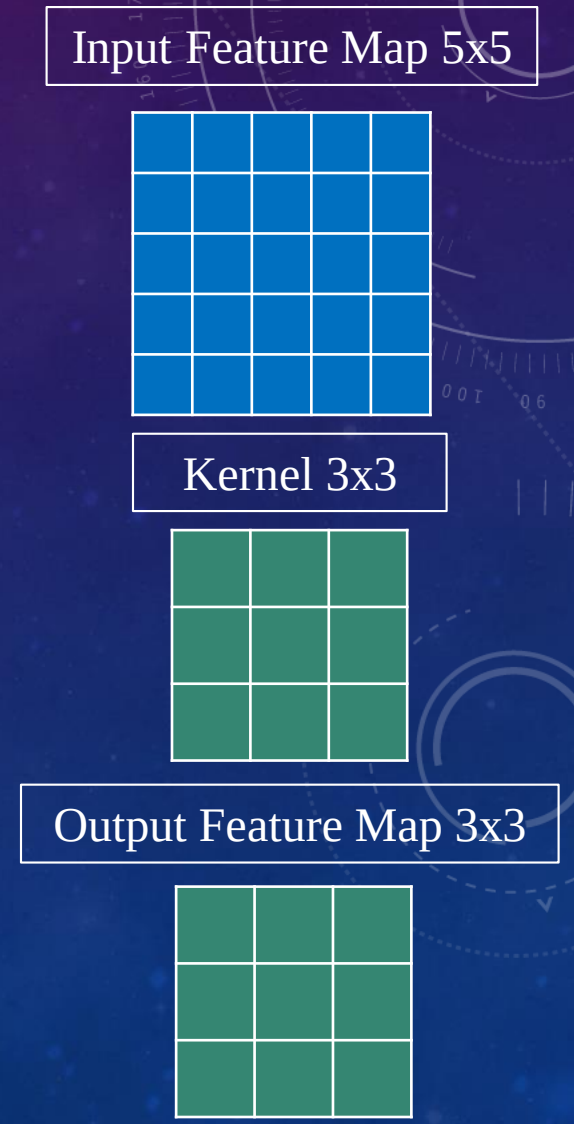
Example: Zero Padding, Non-unit Strides



(i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$)

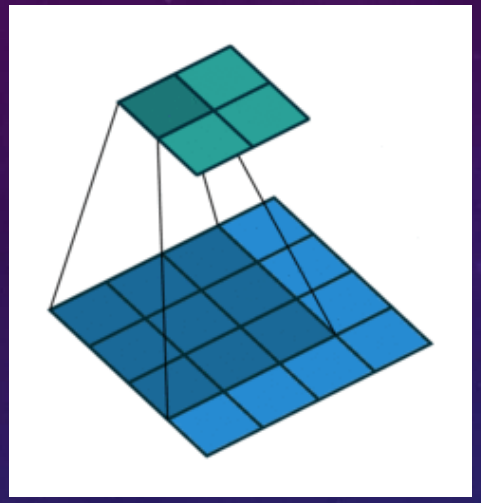
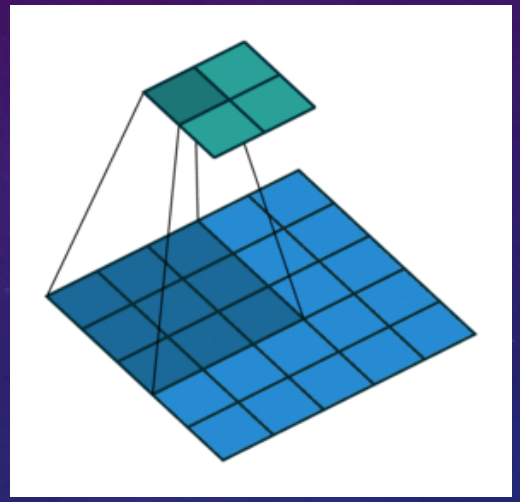
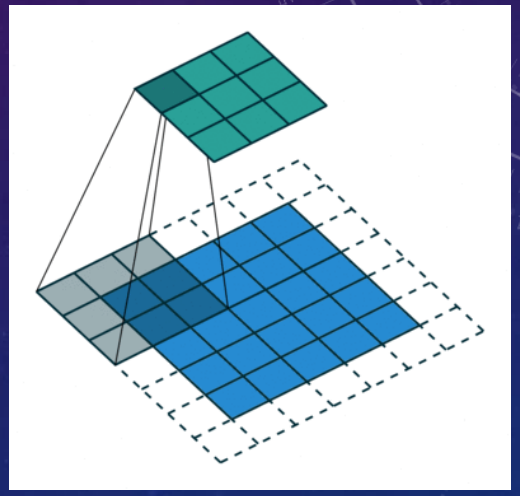
Relationship 6. *For any i, k, p and s,*

$$o = \left[\frac{i + 2p - k}{s}\right] + 1$$



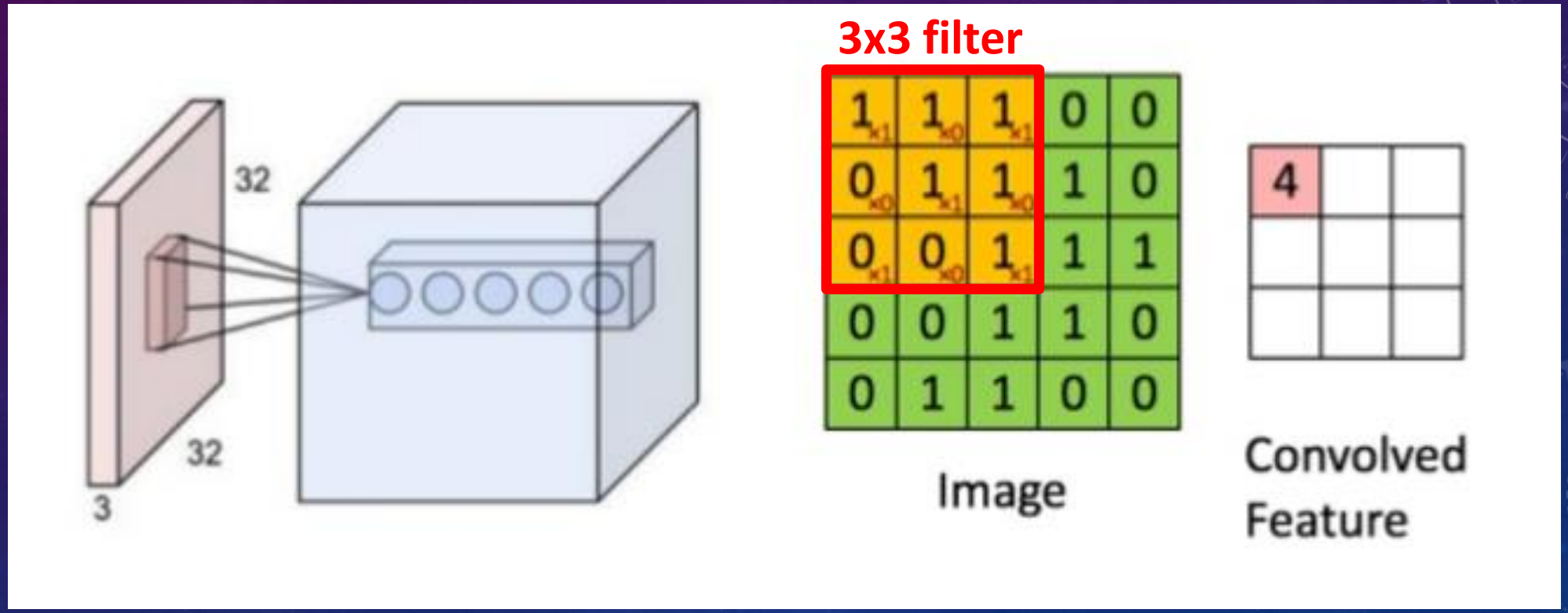
Convolution Animations

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

		
No padding, unit stride	No padding, two strides	Padding, two strides

https://github.com/vdumoulin/conv_arithmetic

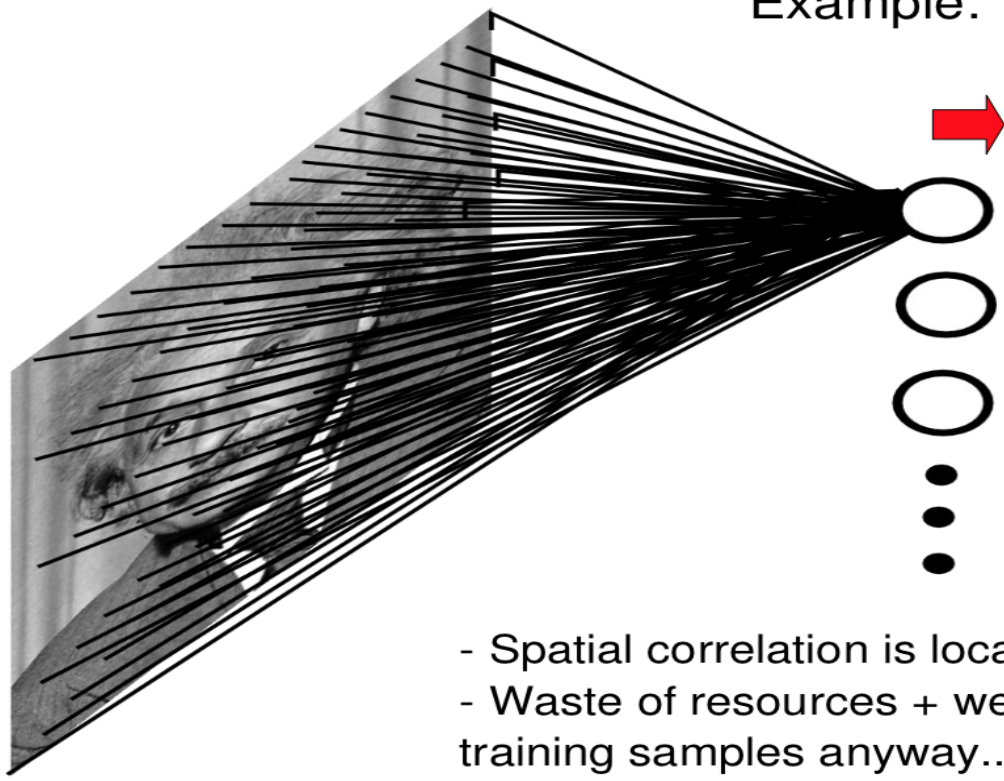
Convolutional Layer



Fully Connected Layer

Fully Connected Layer

Example: 200x200 image
40K hidden units
~2B parameters!!!

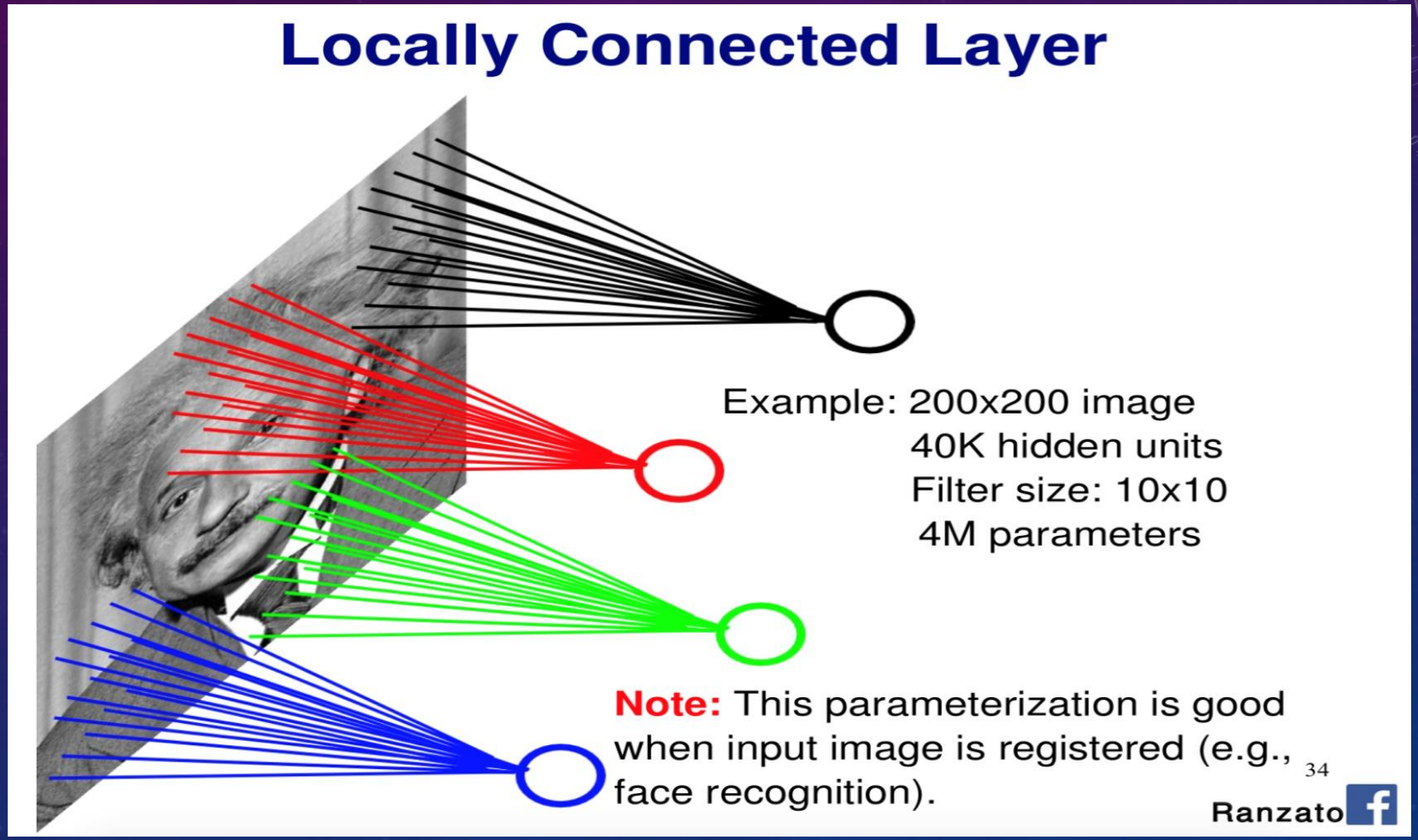


- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

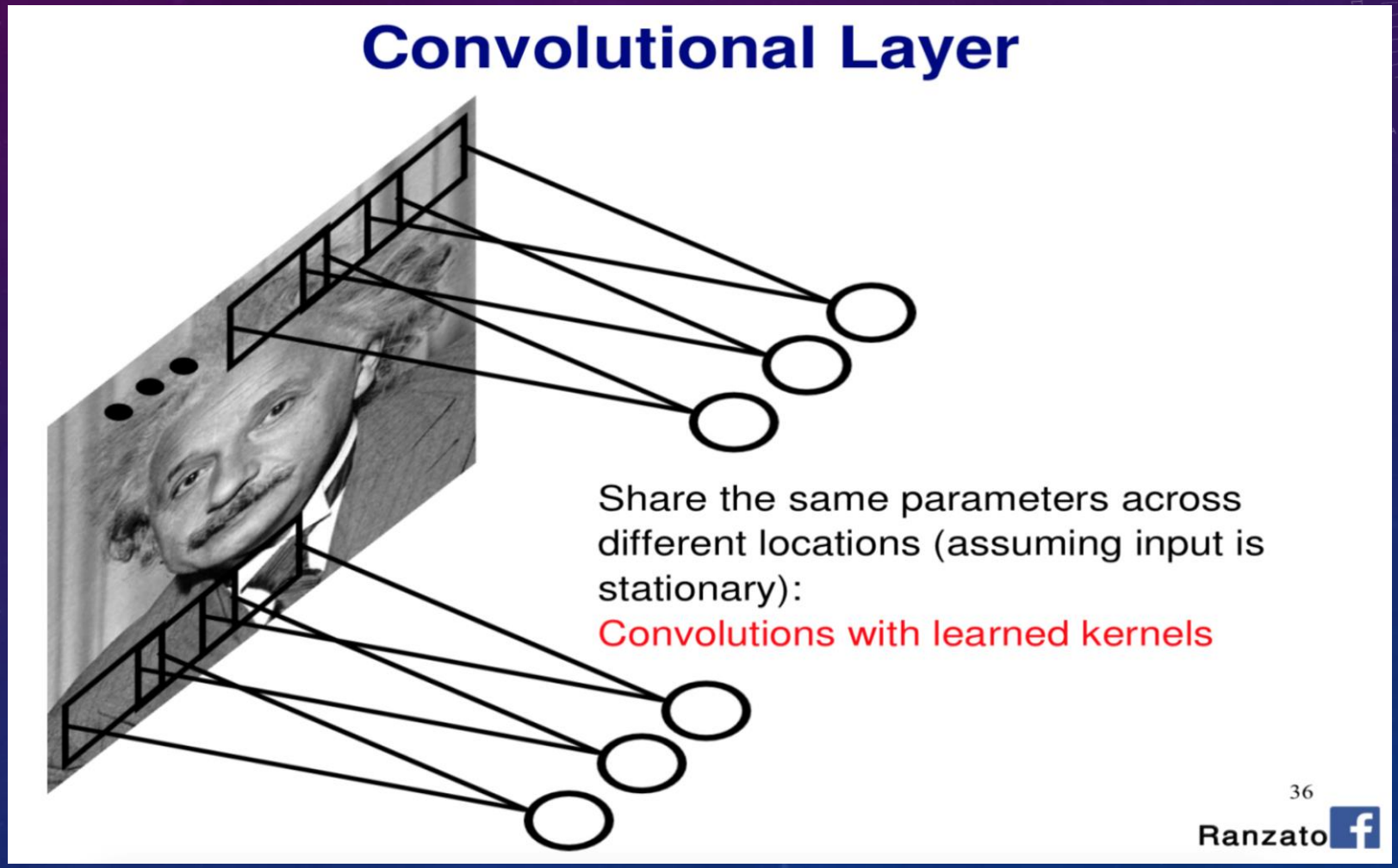
33

Ranzato

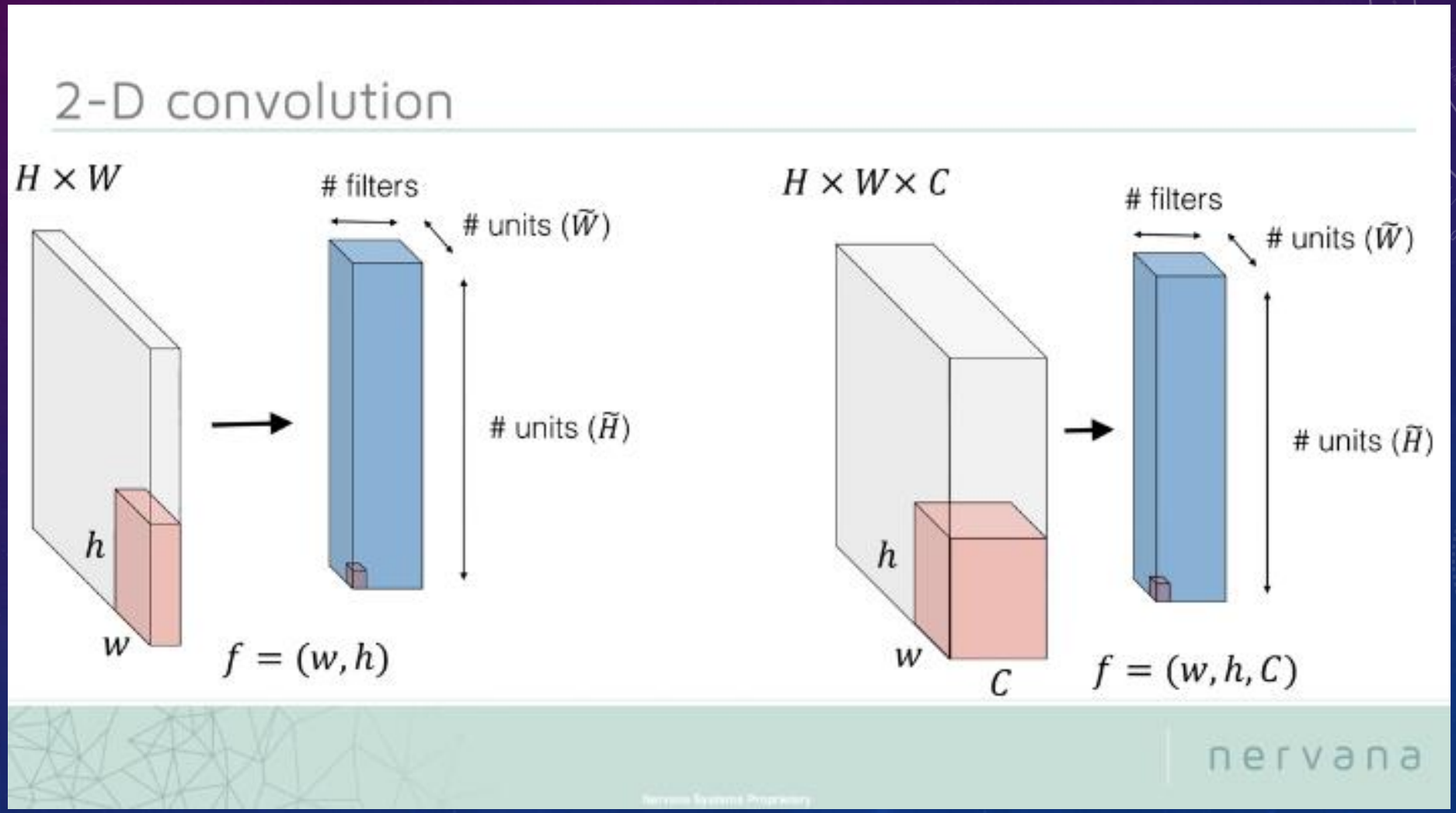
Locally Connected Layer



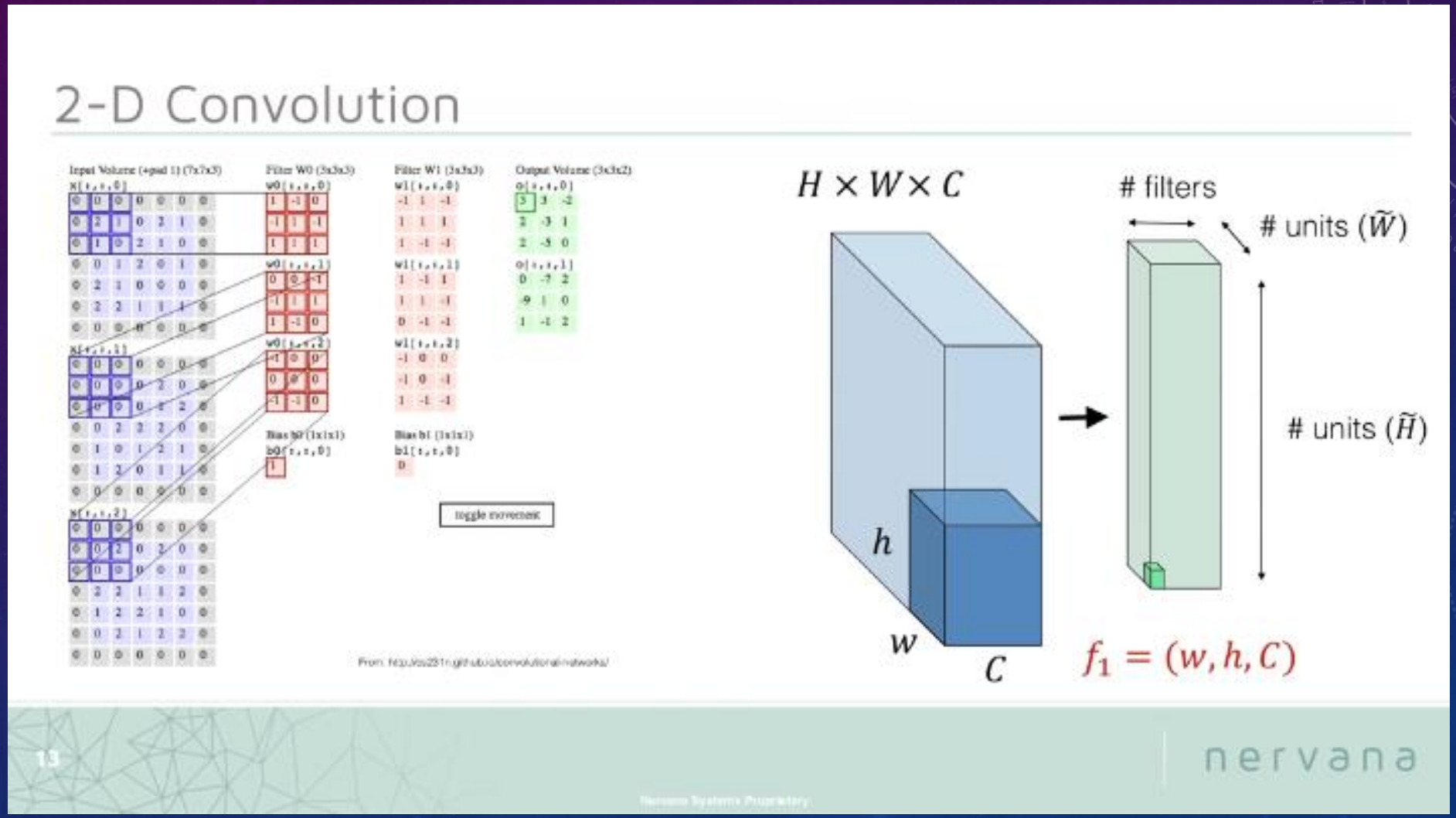
Convolutional Layer



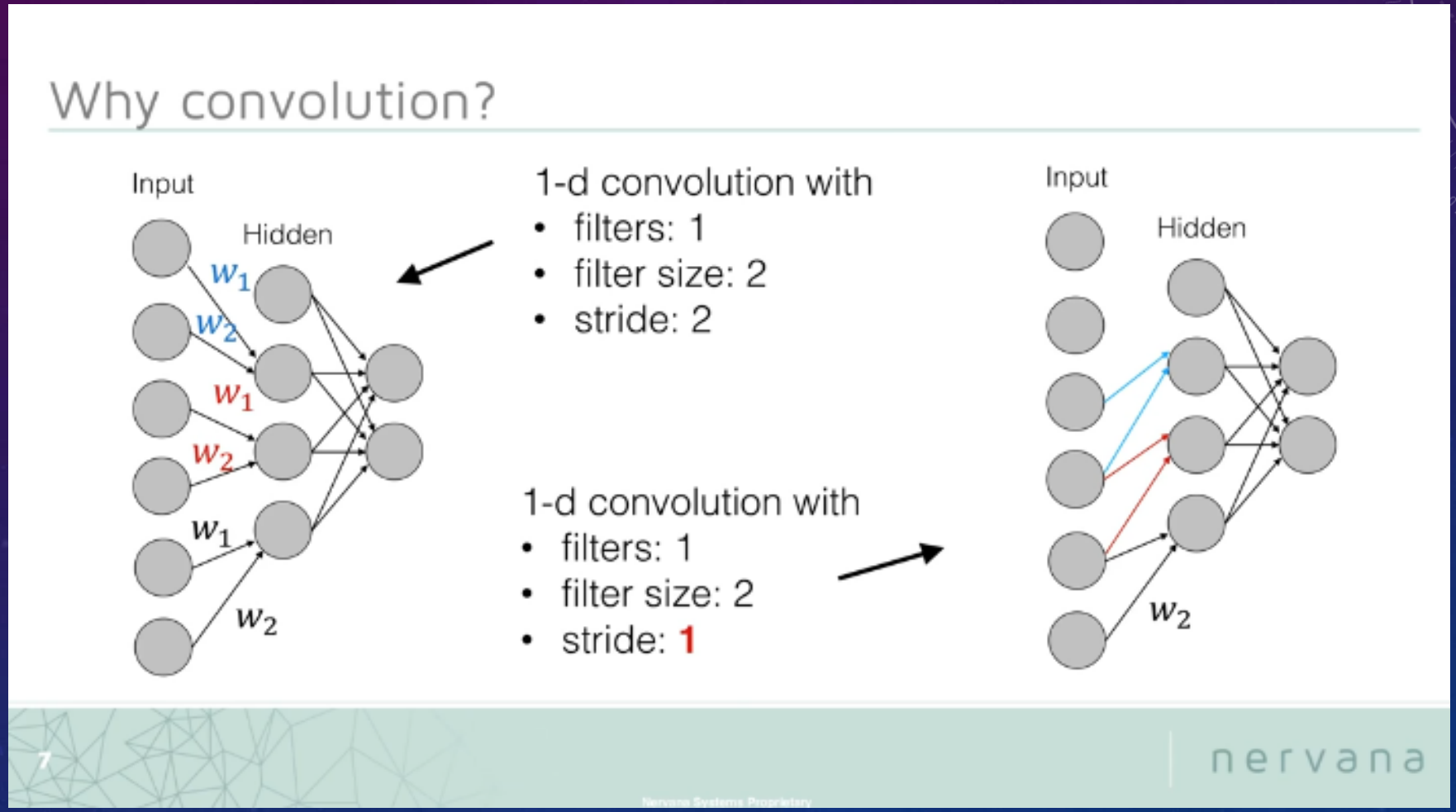
2-D Convolution



2-D Convolution

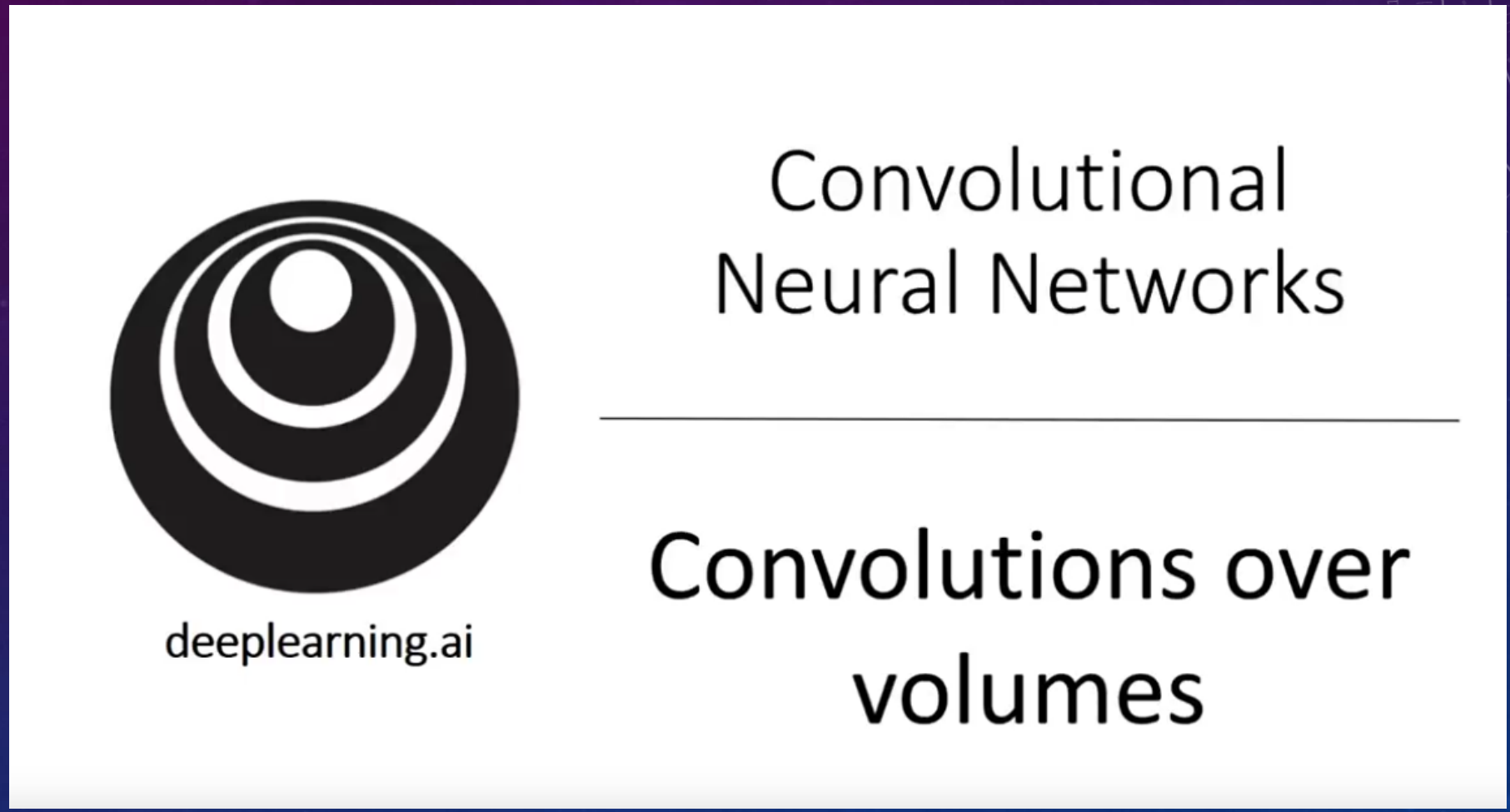


Why Convolution?



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

Strided Convolutions – Andrew Ng

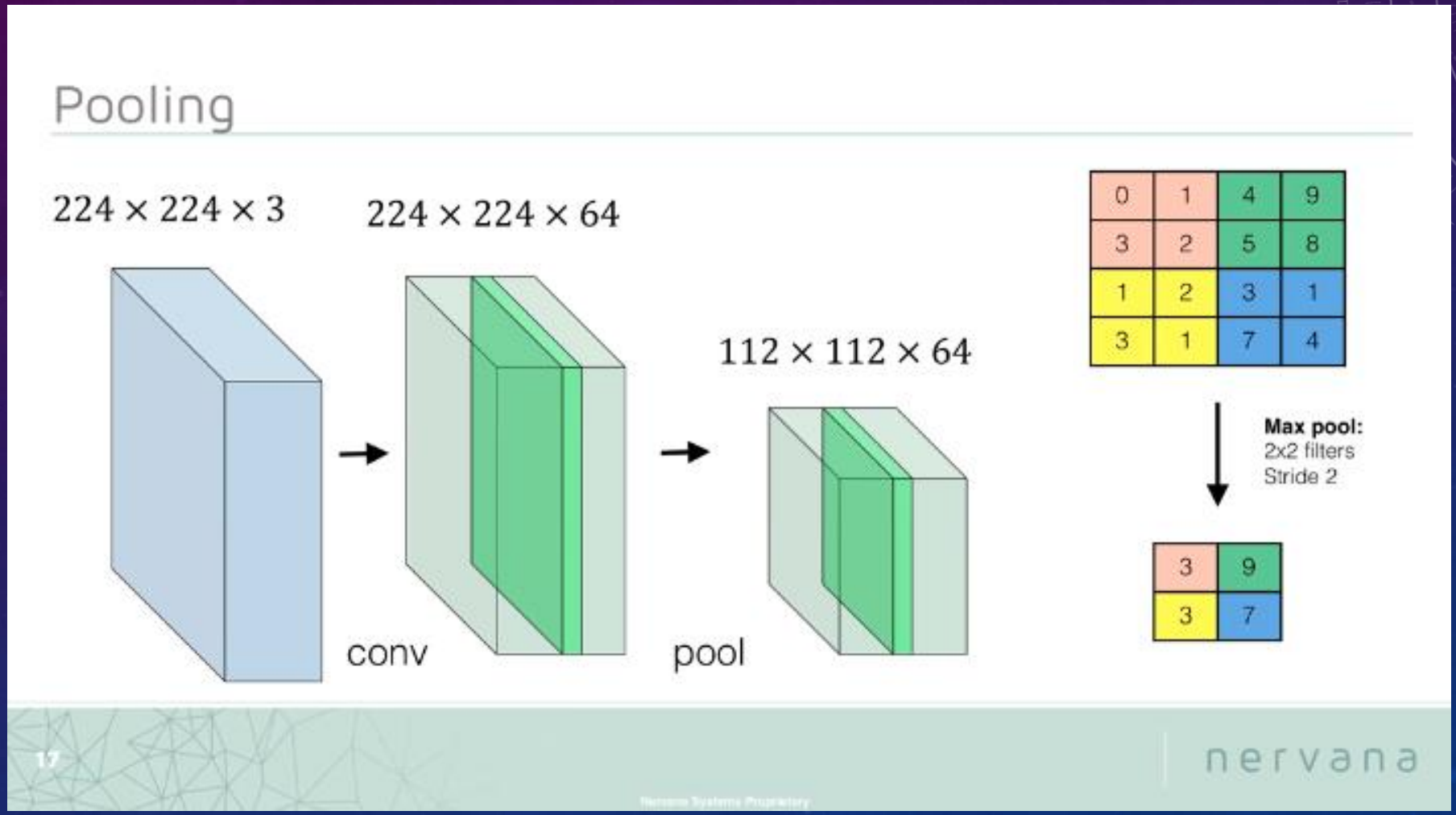


https://www.youtube.com/watch?v=KTB_OFoAQcc [10:44]

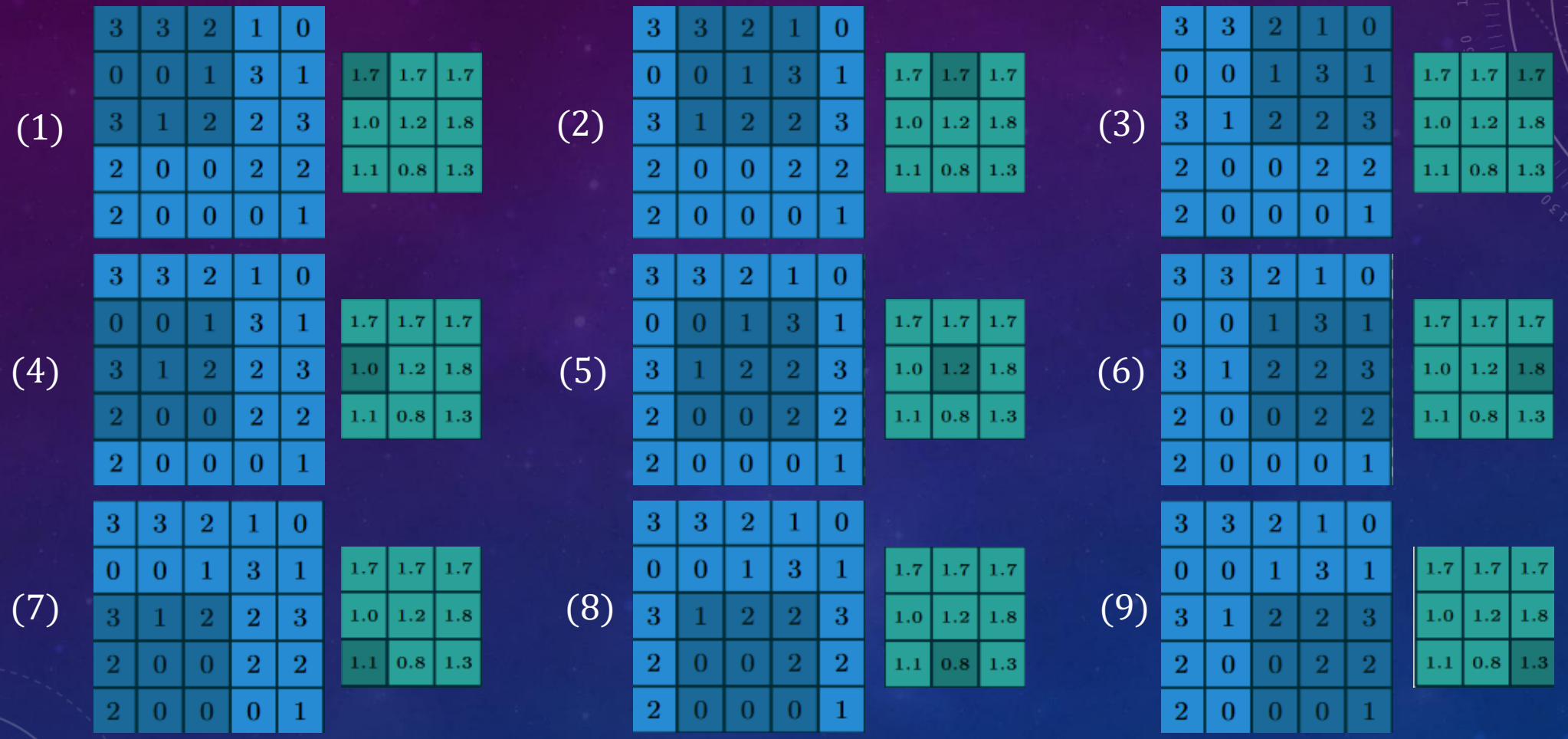
Max Pooling in Convolutional Neural Networks Explained

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

Pooling

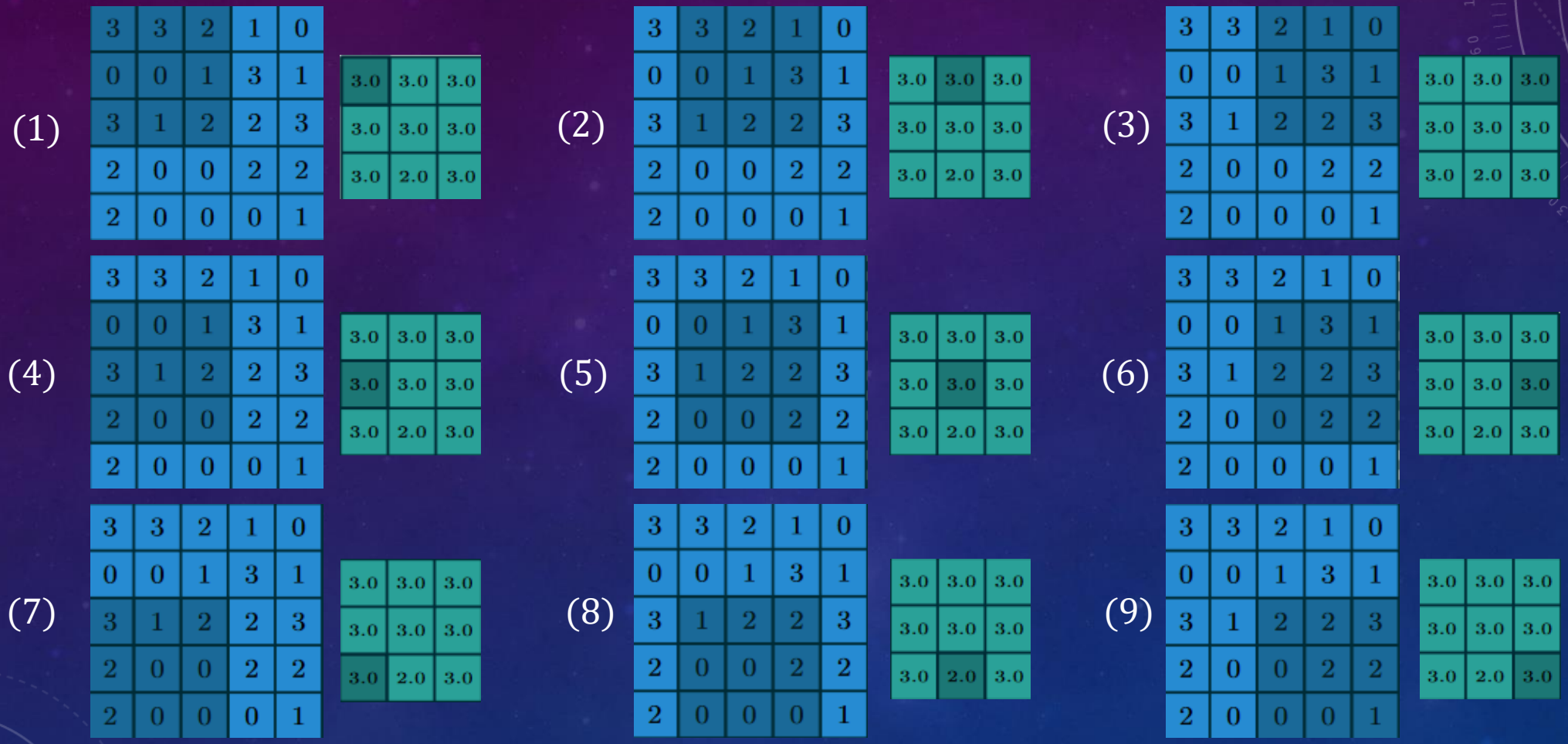

Pooling
224 × 224 × 3
224 × 224 × 64
112 × 112 × 64
conv
pool
0 1 4 9
3 2 5 8
1 2 3 1
3 1 7 4
Max pool:
2x2 filters
Stride 2
3 9
3 7
17
nervana

Pooling : Average Pooling



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

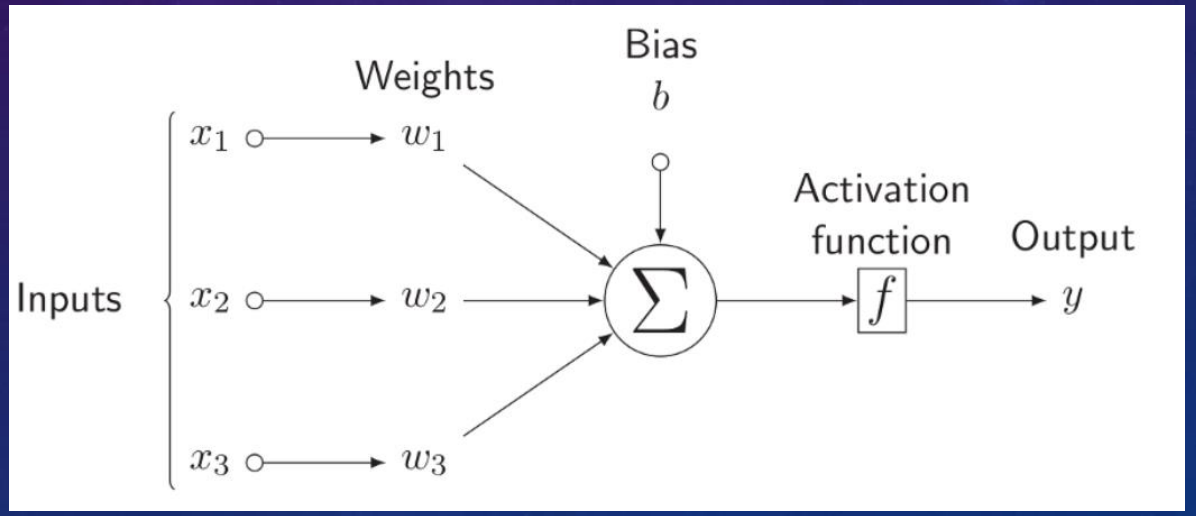
Pooling : Max Pooling



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

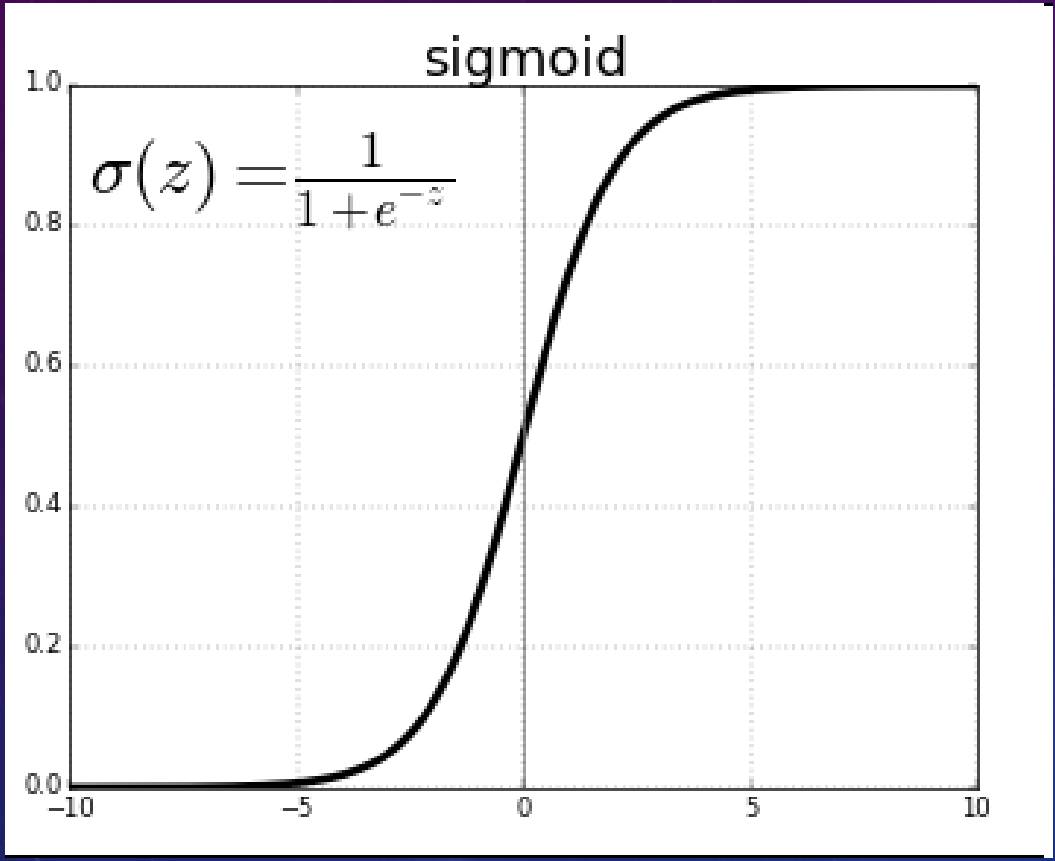
Activation Function

- 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.



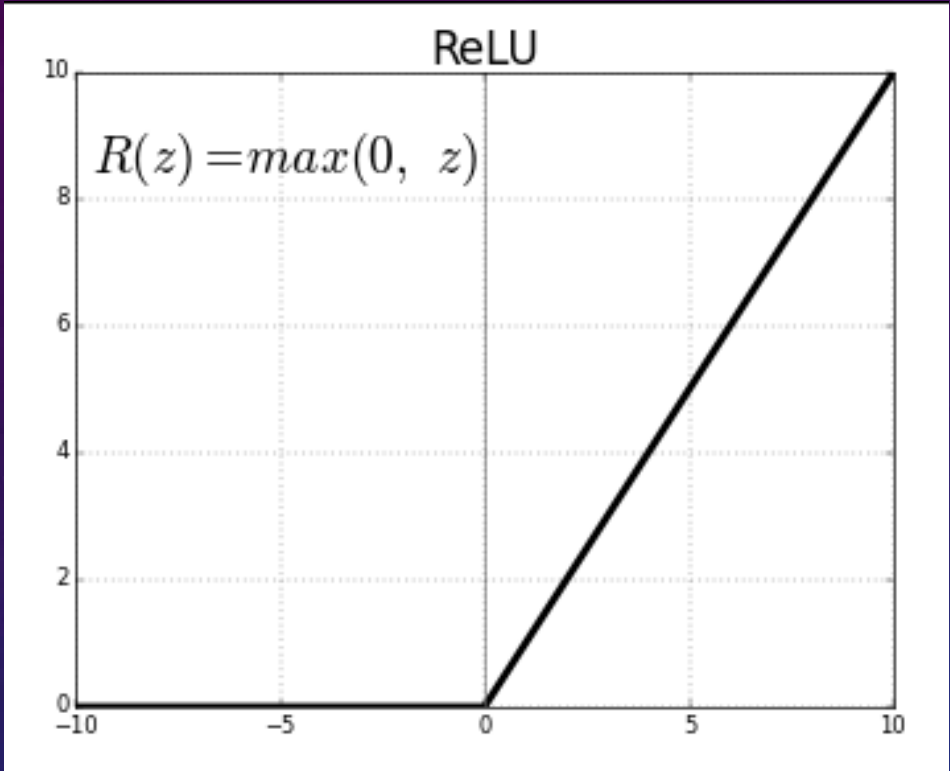
https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network

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

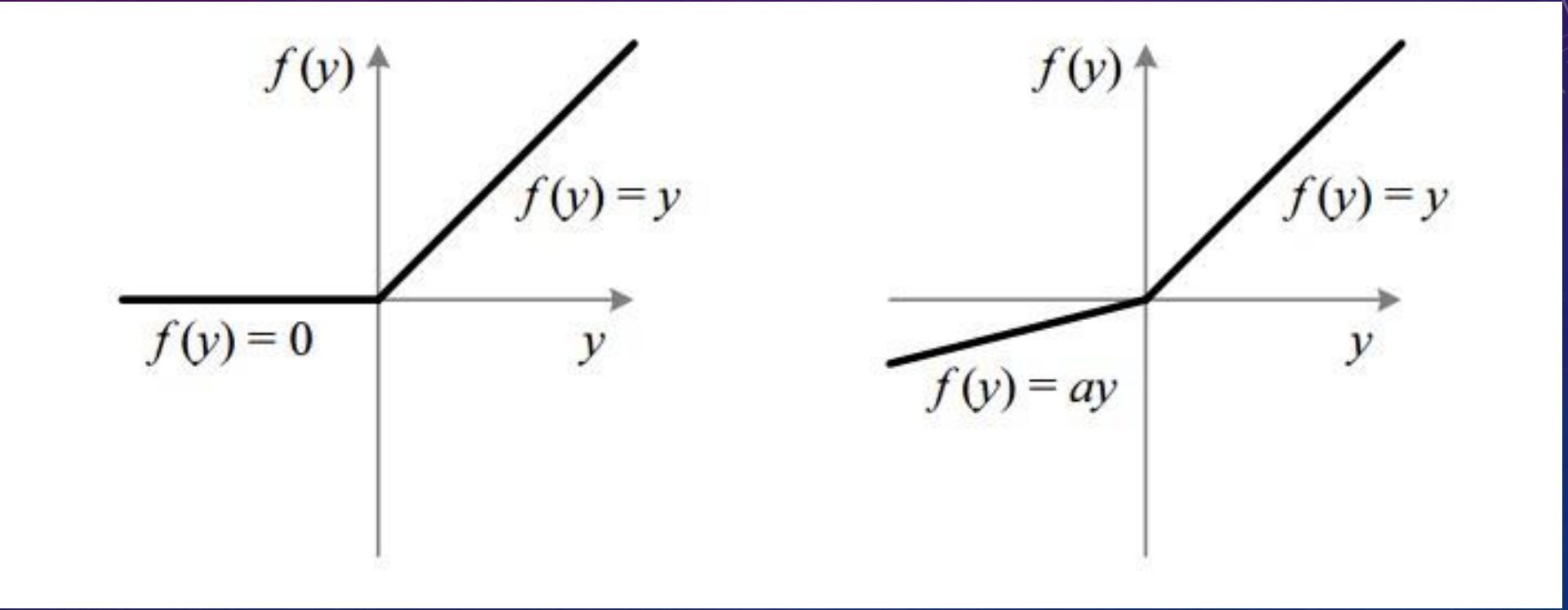


- The ReLU is half rectified (from bottom). *f(z)* is zero when *z* is less than zero and *f(z)* is equal to *z* when *z* is above or equal to zero.

- 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

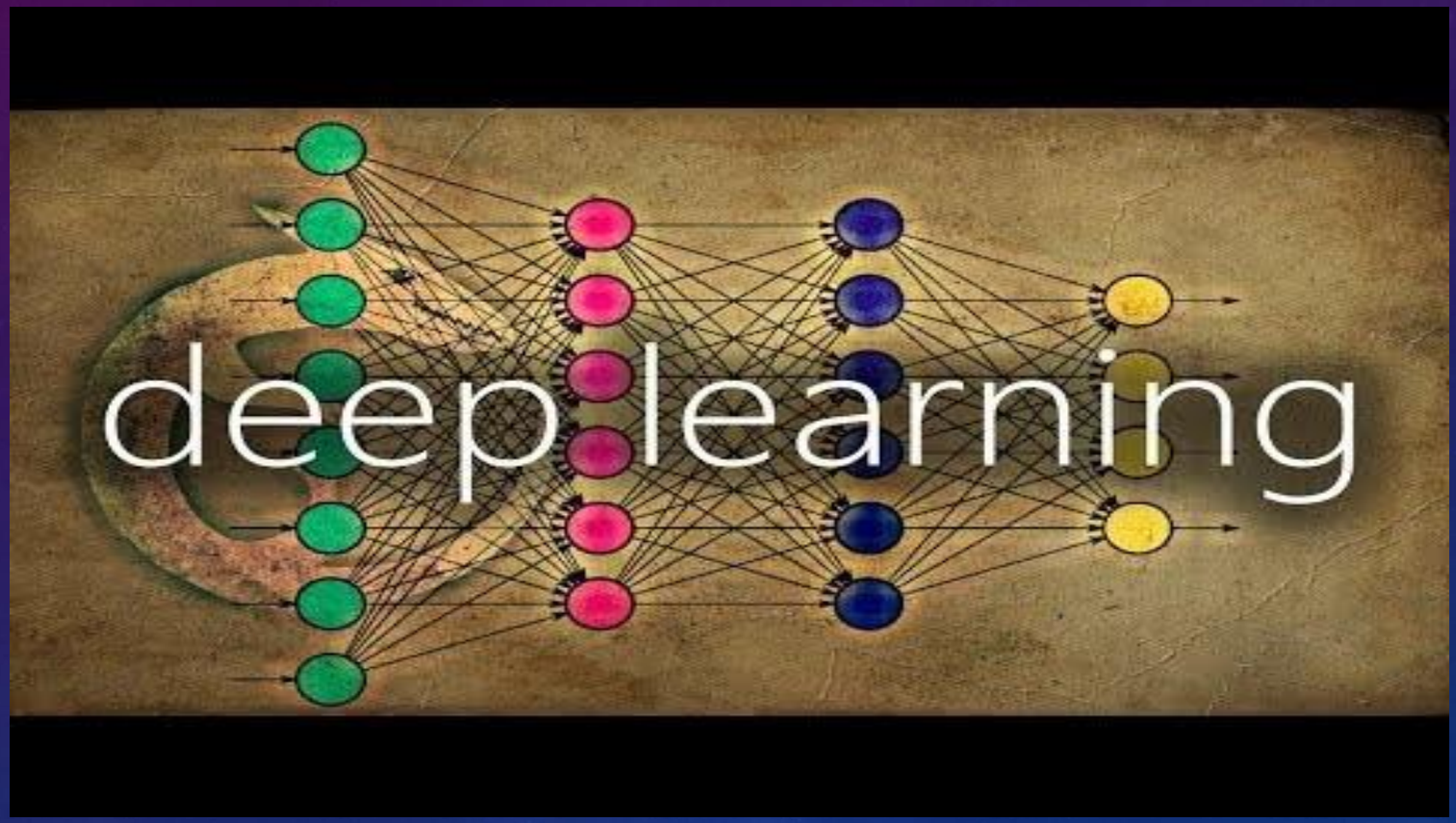


The leak helps to increase the range of the ReLU function.
When a is not 0.01 then it is called Randomized ReLU.

Which Activation Function Should I Use?

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

Weight Initialization

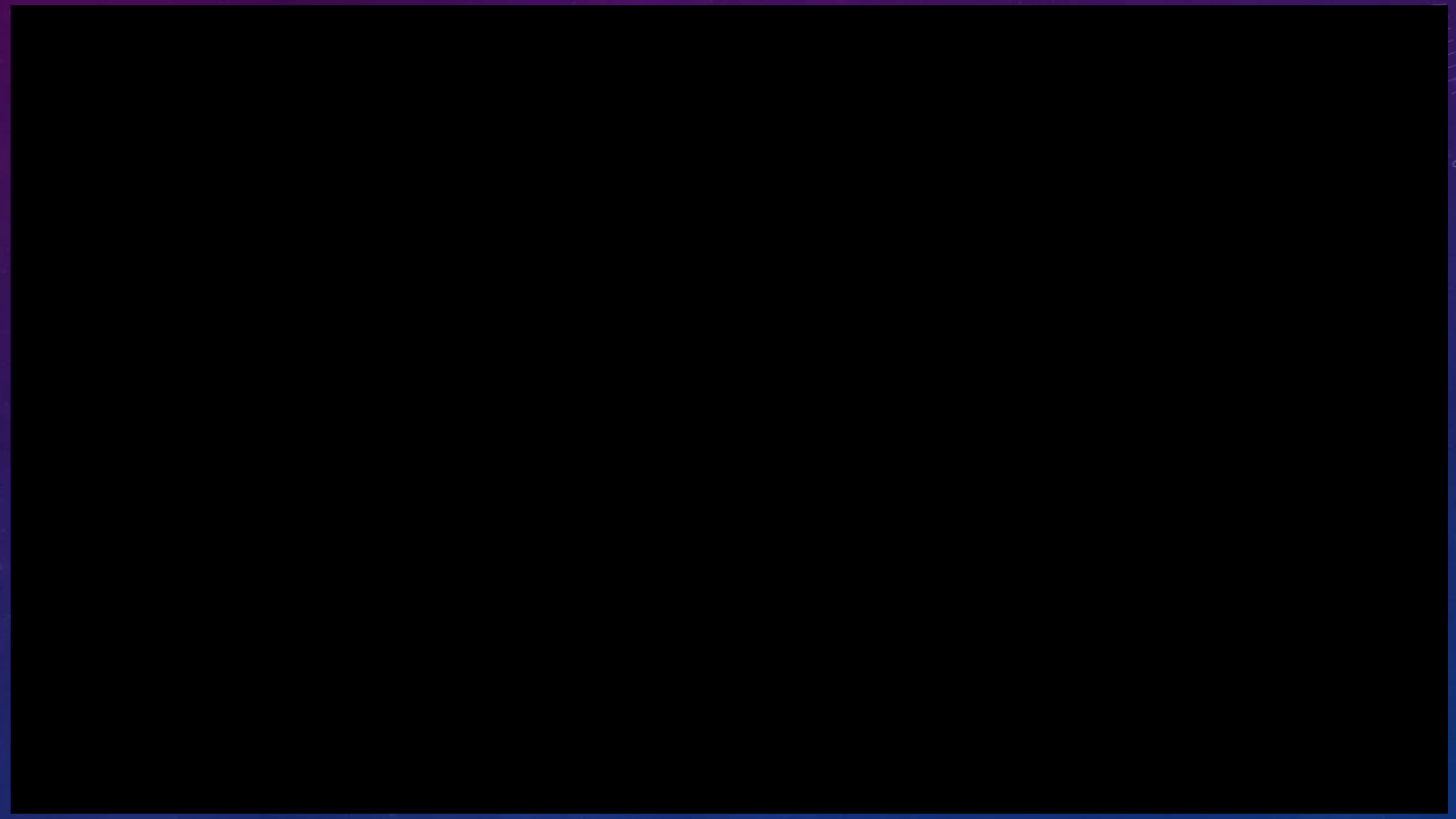


https://www.youtube.com/watch?v=8krd5qKVw-Q [10:00]

Weight Initialization – But Why?

- Prevent layer activation outputs from vanishing (0) or exploding (∞) during forward pass
- This is to avoid the gradients become 0 or too large in order for the backpropagation algorithm to work

Batch Normalization explained



https://www.youtube.com/watch?v=DtEq44FTPM4 [8:48]

Batch Normalization Mathematically

- Batch Normalization manipulates the layer inputs by calculating a batch's mean and variance. The data is then scaled and shifted
- Therefore, Batch Normalization is a special kind of preprocessing. The mathematical procedure can be seen on the right.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

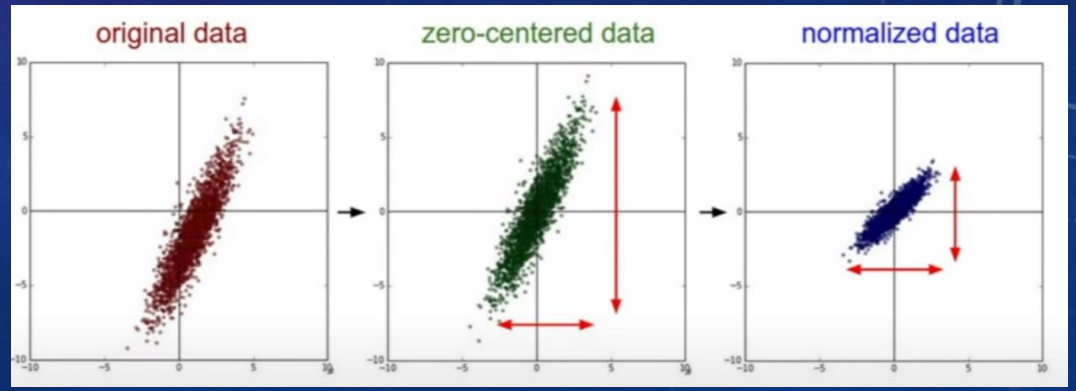
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

The simple equation behind batch normalization



Advantages of Batch Normalization

- Enables higher learning rates
- Accelerates the learning process
- Training Neural Nets with Sigmoid activations
- Bridged the other normalization methods such as Layer Normalization and weight normalization

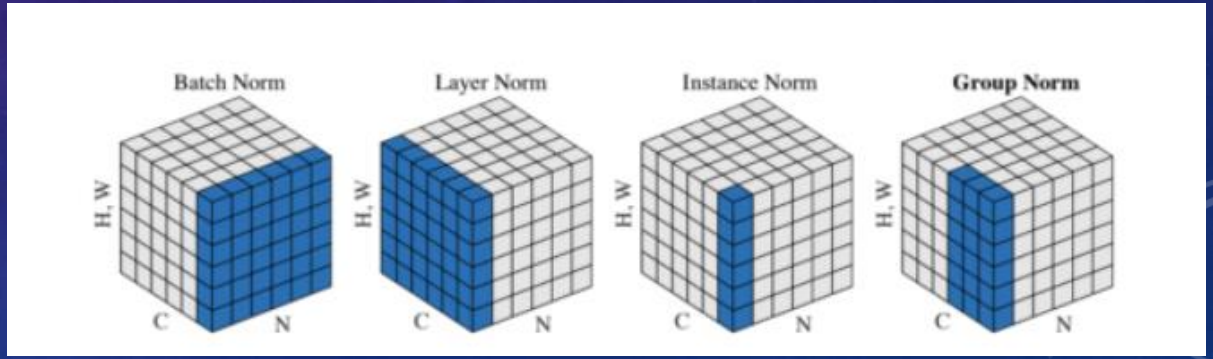
Layer Normalization

Mean of a layer

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

variance

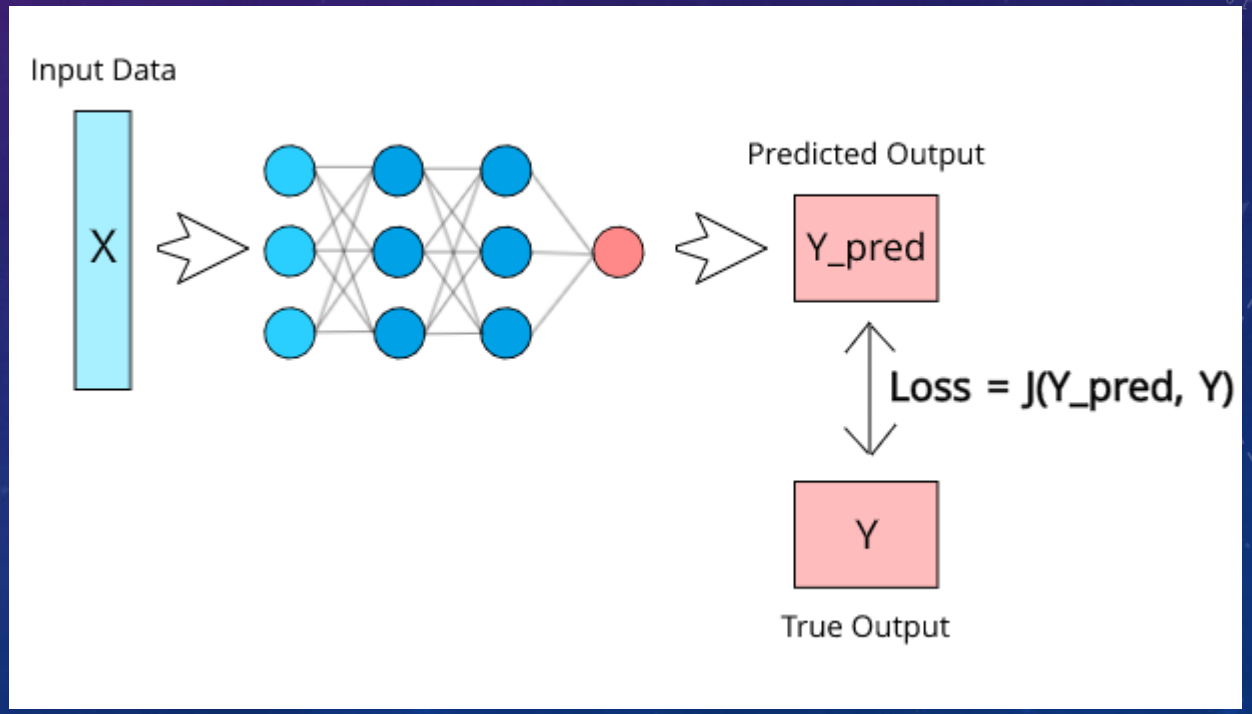
H=# of hidden units in one layer. Details in the paper https://arxiv.org/pdf/1607.06450.pdf



N is the batch axis, C is the Channel axis, (H,W) represent the spatial axis. The blue color indicates a normalization by same mean and variance, computed by the aggregation of the colored blocks

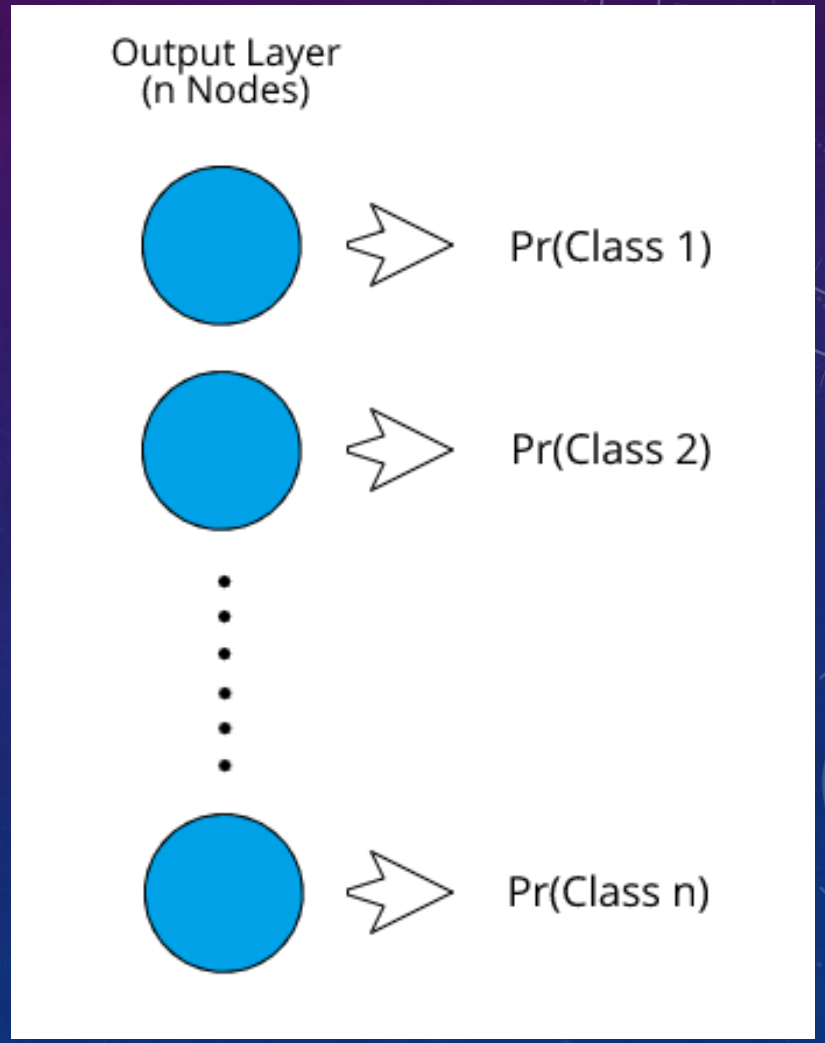
Loss Functions

- 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



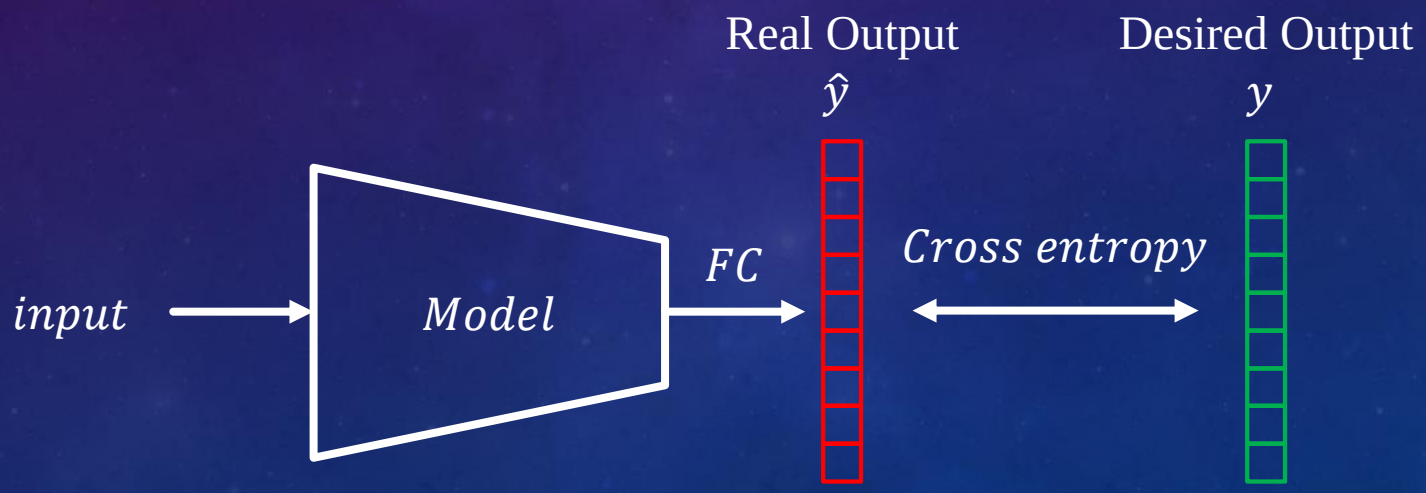
Classification Loss

- 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

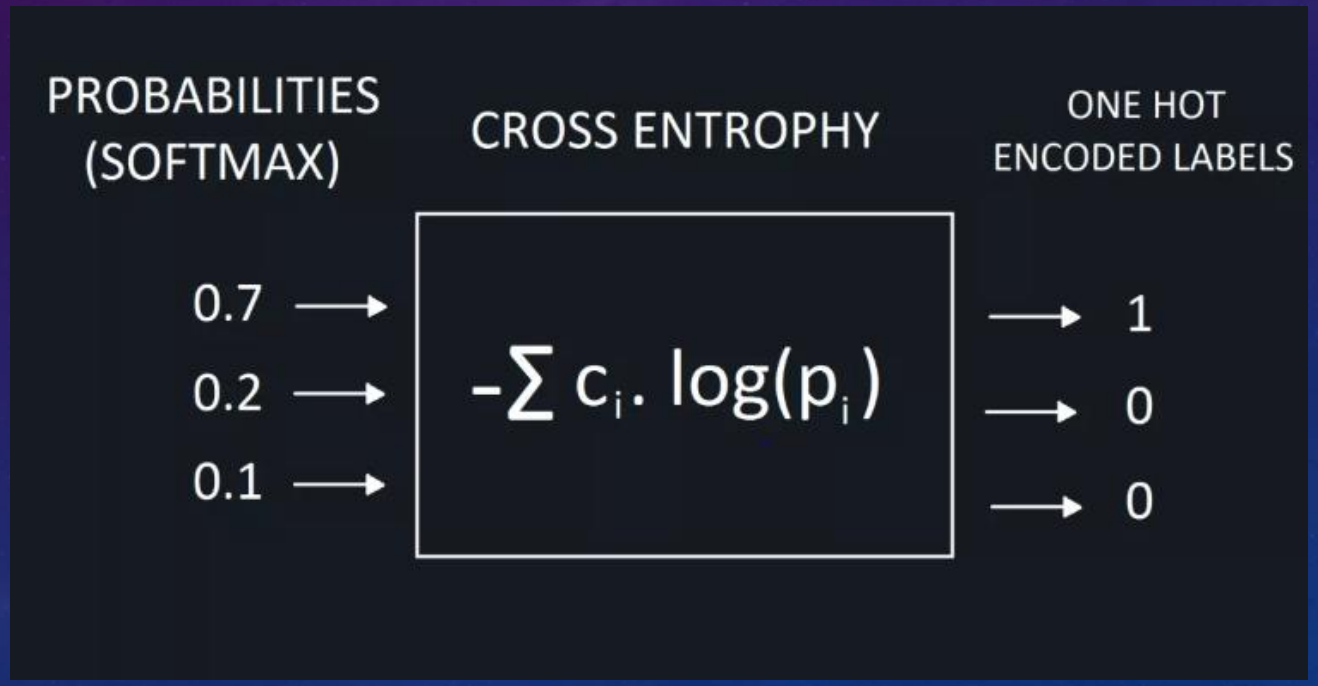
- We apply the standard exponential function to each element z_i of the input vector z and normalize these values by dividing by the sum of all these exponentials; this normalization ensures that the sum of the components of the output vector $\sigma(z)$ is 1.
- Instead of e, a different base $b > 0$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

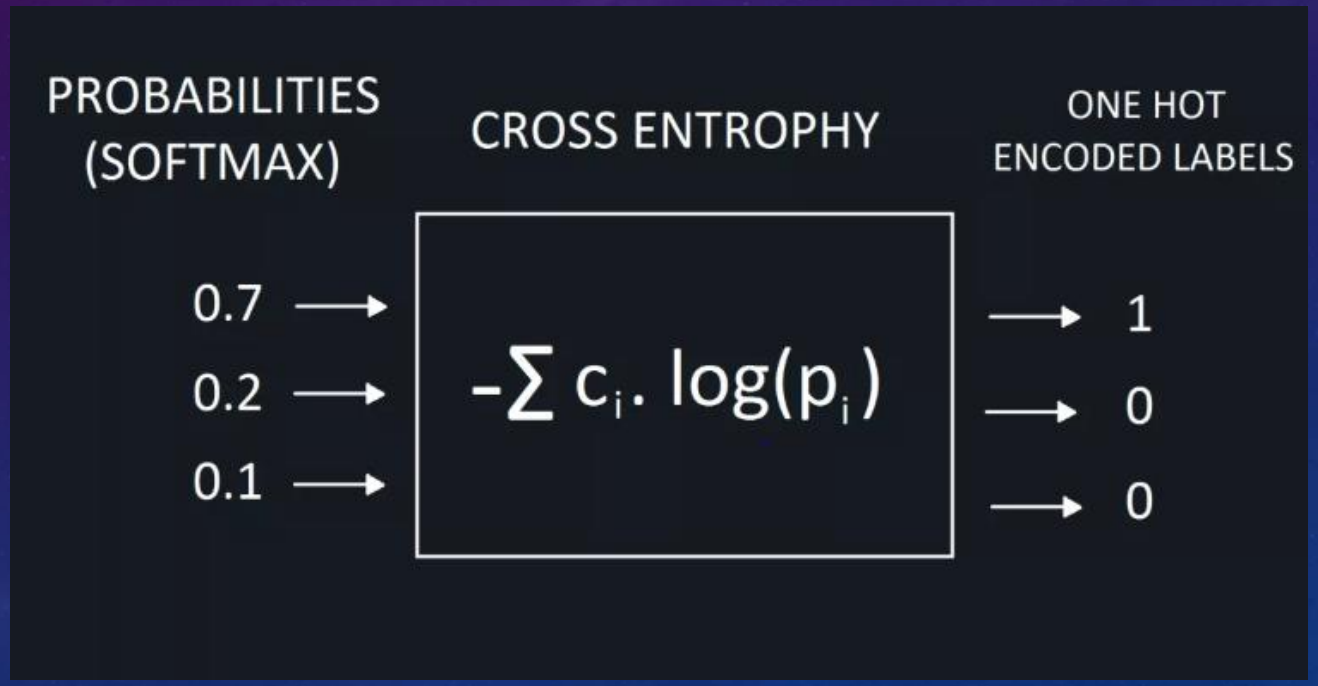


y
(Ground-Truth)

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.

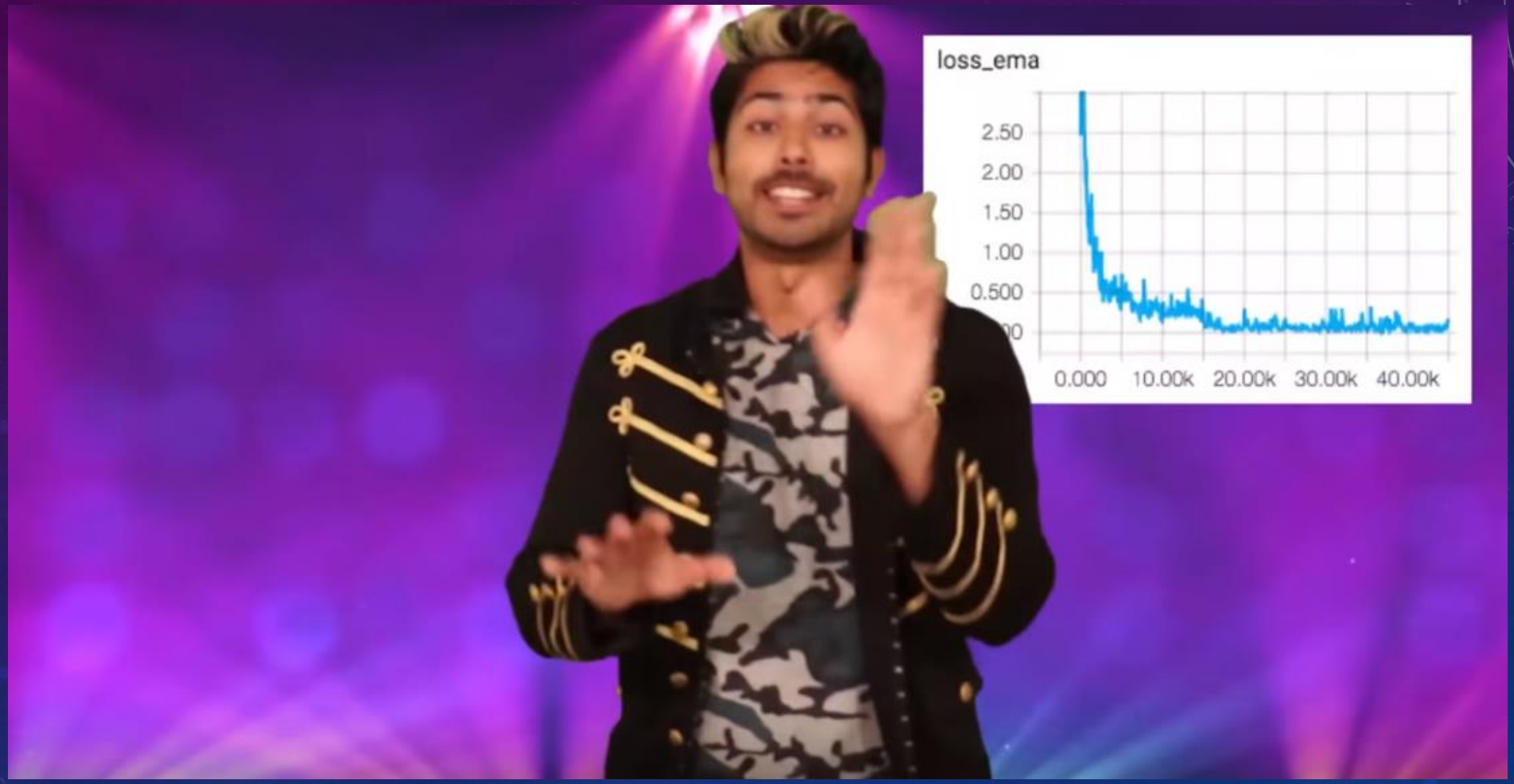
$$E(y, \hat{y}) = -\sum_i y_i log\hat{y}_i - (1 - y_i)\log(1 - \hat{y}_i)$$

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]

Backpropagation – Chain Rule – by BlueBrown

https://www.youtube.com/watch?v=tIeHLnjs5U8 [10:00]

Backpropagation

DEFINITION

1) **Calculate the forward phase** for each input-output pair $(\vec{x_d}, y_d)$ and store the results $\hat{y_d}$, a_j^k, and o_j^k for each node j in layer k by proceeding from layer 0, the input layer, to layer m, the output layer.

2) **Calculate the backward phase** for each input-output pair $(\vec{x_d}, y_d)$ and store the results $\frac{\partial E_d}{\partial w_{ij}^k}$ for each weight w_{ij}^k connecting node i in layer $k-1$ to node j in layer k by proceeding from layer m, the output layer, to layer 1, the input layer.

a) Evaluate the error term for the final layer δ_1^m by using the second equation.
b) Backpropagate the error terms for the hidden layers δ_j^k, working backwards from the final hidden layer $k = m - 1$, by repeatedly using the third equation.
c) Evaluate the partial derivatives of the individual error E_d with respect to w_{ij}^k by using the first equation.

3) **Combine the individual gradients** for each input-output pair $\frac{\partial E_d}{\partial w_{ij}^k}$ to get the total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ for the entire set of input-output pairs $X = \{(\vec{x_1}, y_1), \ldots, (\vec{x_N}, y_N)\}$ by using the fourth equation (a simple average of the individual gradients).

4) **Update the weights** according to the learning rate α and total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ by using the fifth equation (moving in the direction of the negative gradient).

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)

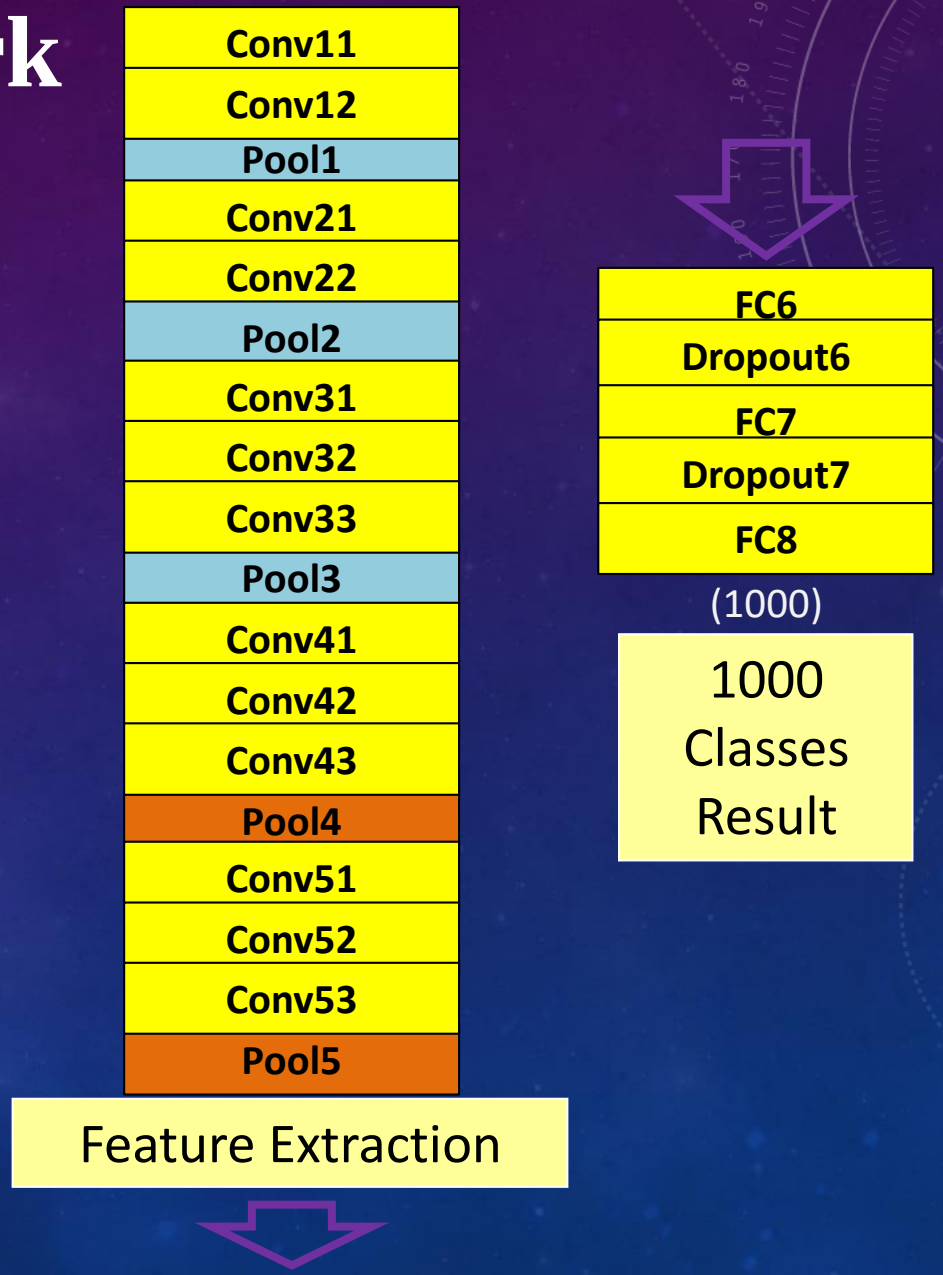
ResNet (47:24 – 53:08)

(53:56 – 55:01)

(58:16 – 1:02:33)

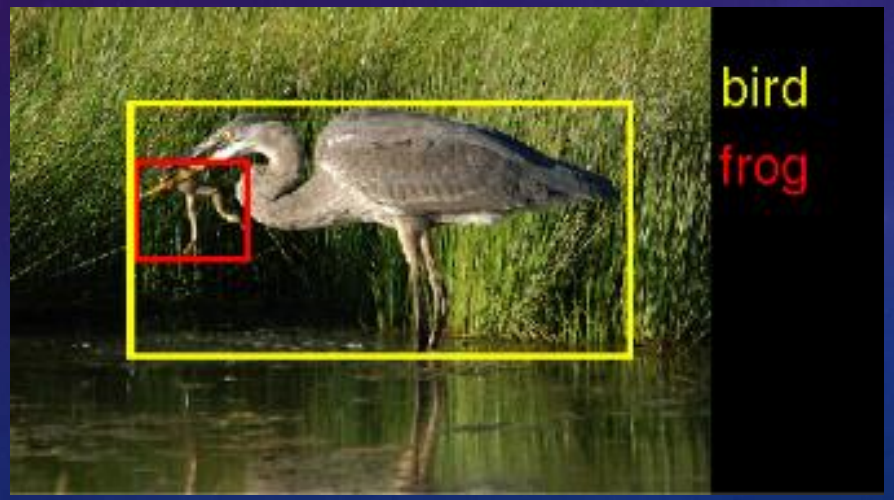
Convolution Neural Network

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



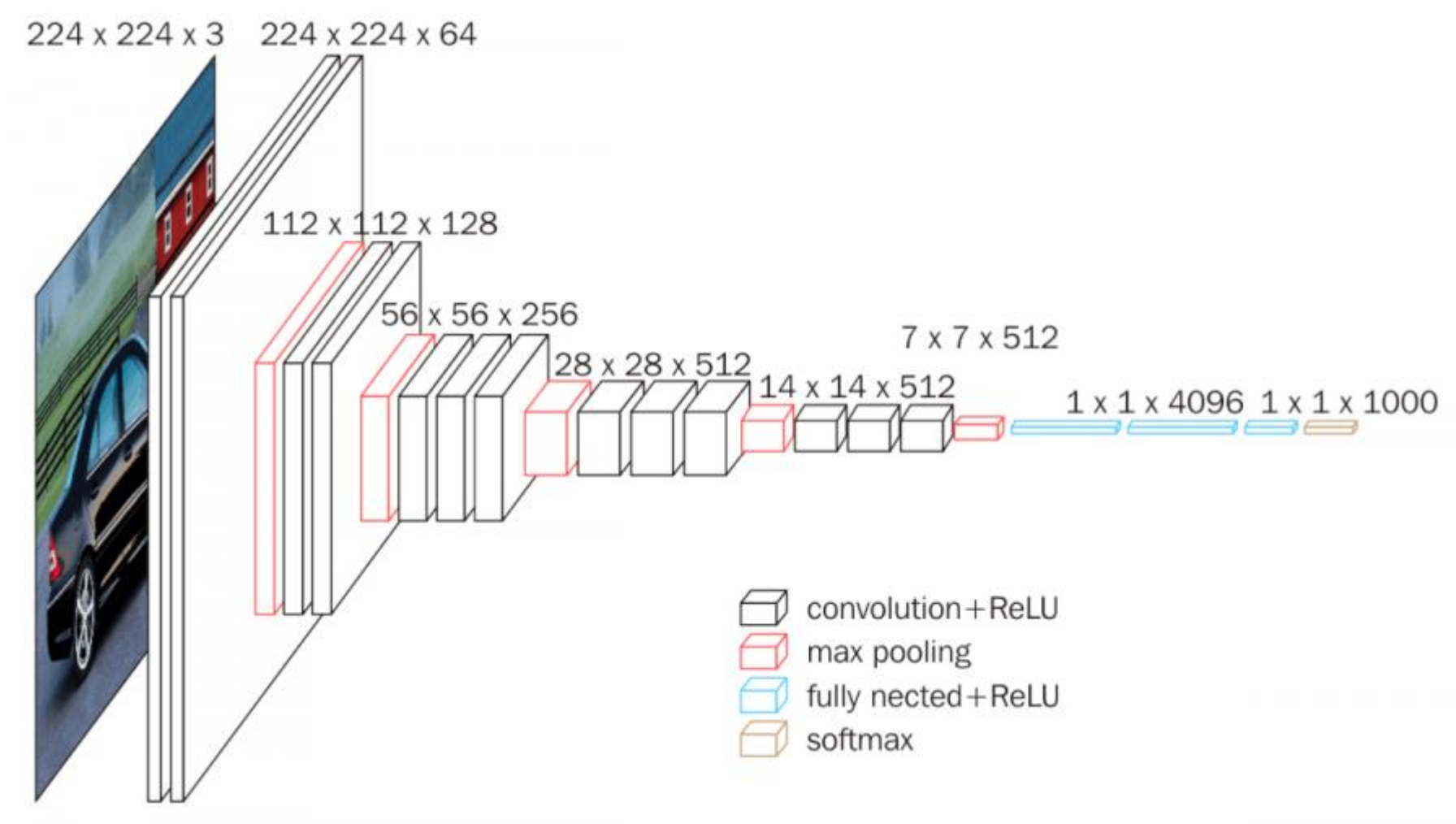
ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014 in the Classification Task

- 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/

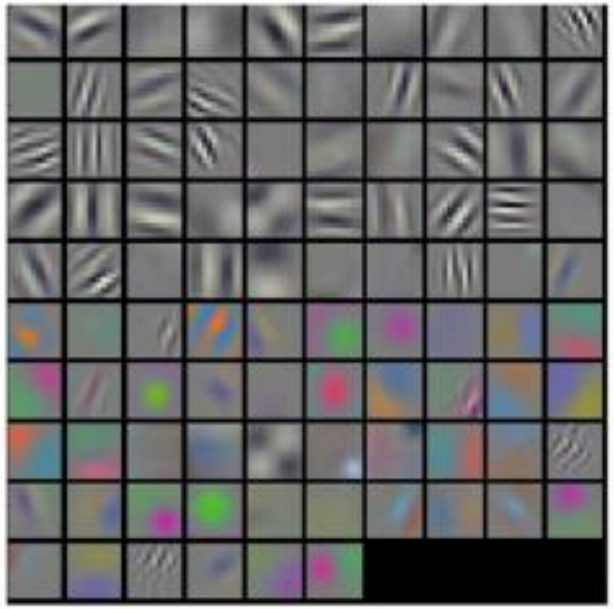
ILSVRC-2014 VGG Model



What does Filter Learn?

So what does it learn?

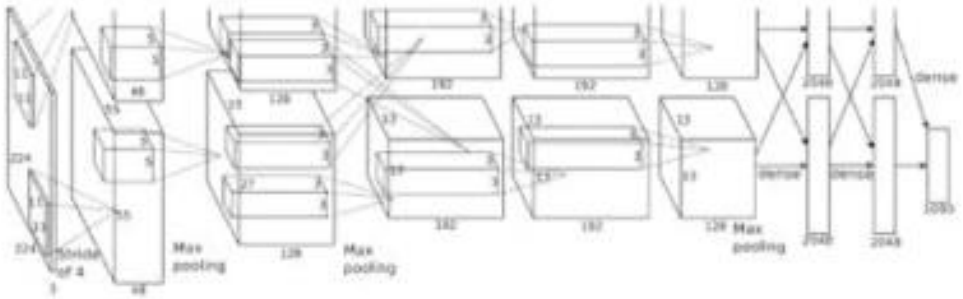
First conv layer weights

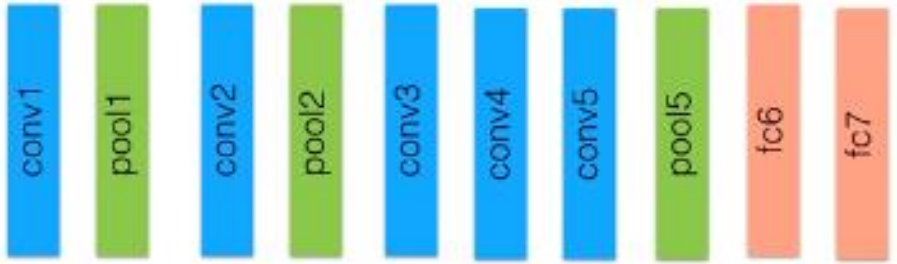


Well-Known Networks

Common Models

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



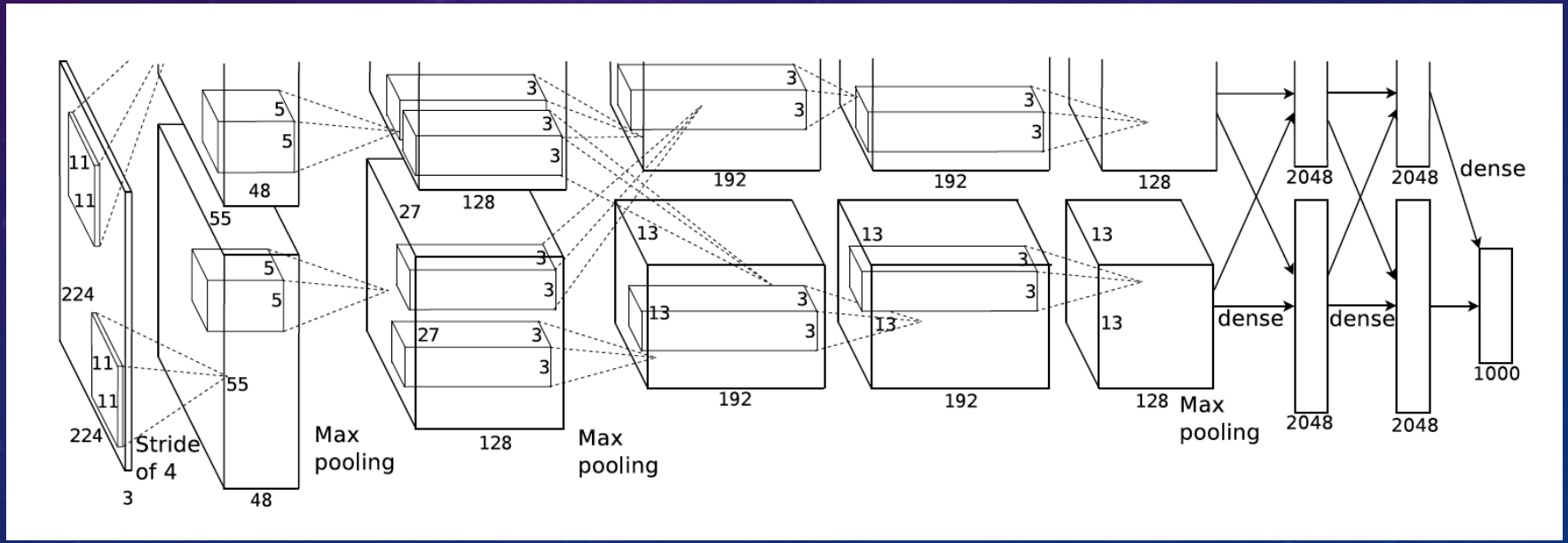


nervana

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

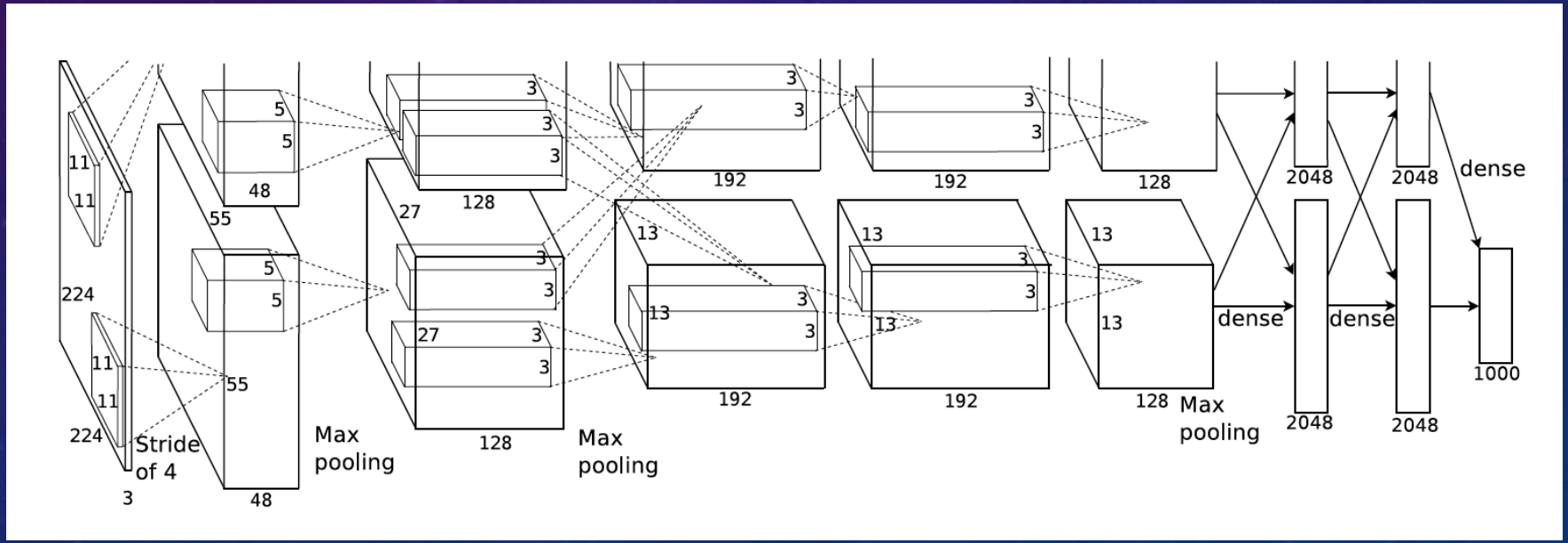
AlexNet

- Use Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
- Use dropout instead of regularisation 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%, repectively.



AlexNet

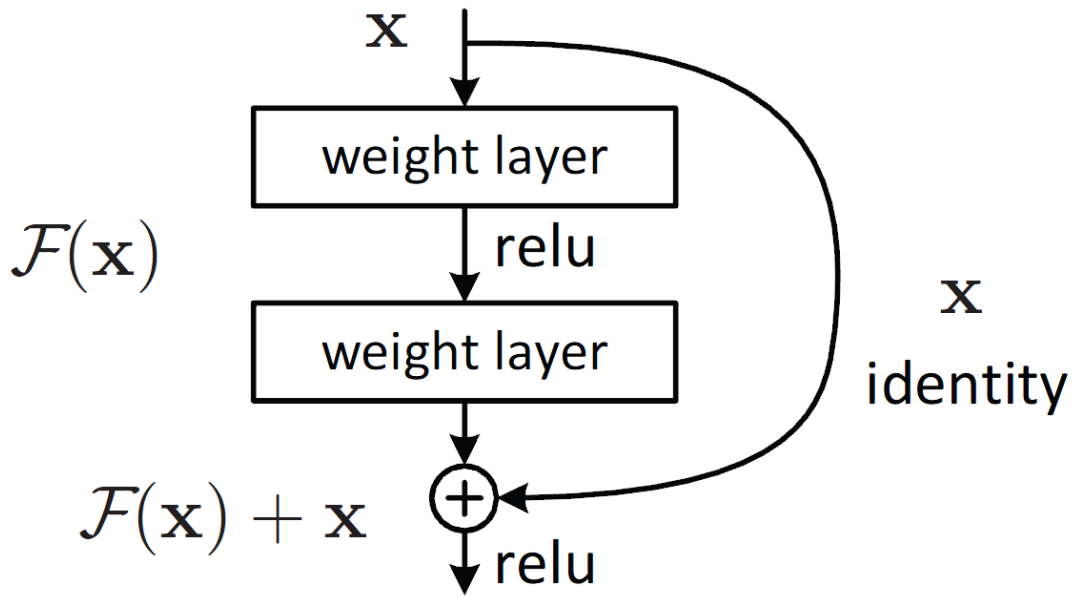
- Copy convolution layers into different GPUs; Distribute the fully connected layers into different GPUs.
- Feed one batch of training data into convolutional layers for every GPU (Data Parallel).
- Feed the results of convolutional layers into the distributed fully connected layers batch by batch (Model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



Well-Known Networks

Common Models

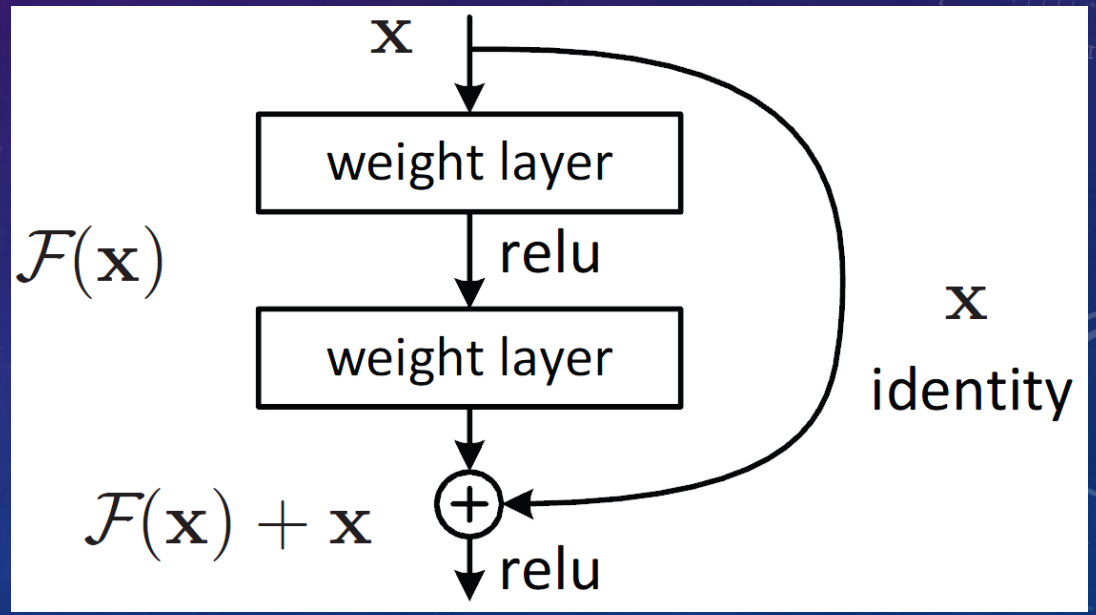
- AlexNet (ILSVRC 2012 winner)
- ZF Net (2013 winner)
- GoogLeNet (2014 winner)
- VGG (2014 runner-up)
- **ResNet (2015 winner)**



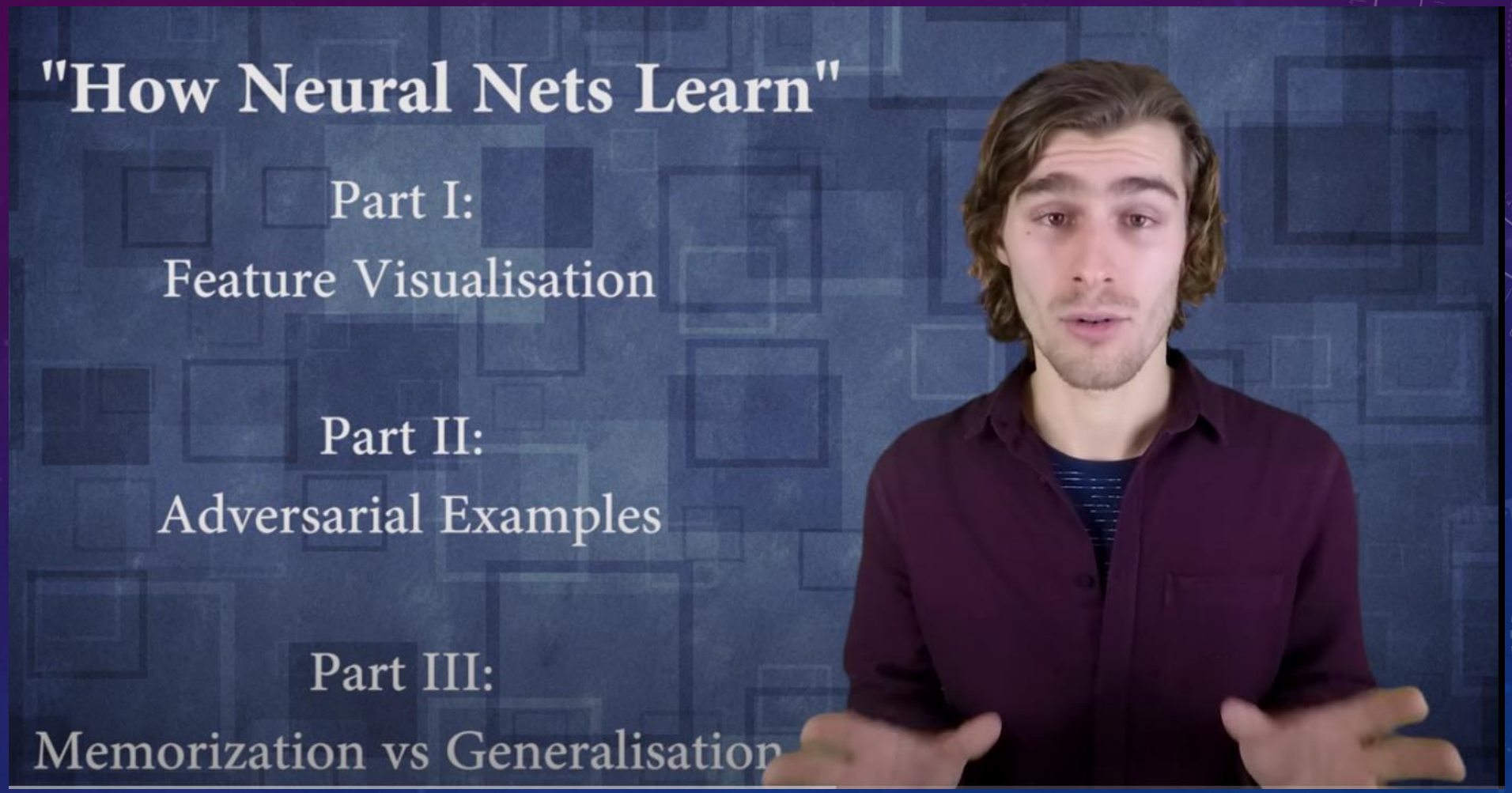
nervana

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:



‘How neural networks learn’ - Part I: Feature Visualization



https://www.youtube.com/watch?v=McgxRxi2Jqo&feature=youtu.be&ab_channel=ArxivInsights [14:59]

Exercise 2-1: Feature Map Visualization

- Pip install opencv-python and matplotlib in your pytorch environment.
- Please download the “2-1_Feature_map_visualization” on Clouds and choose your own images from Internet.
- Compare the feature maps that extract from “Conv21” and observe the size and dimension of the feature maps.

Please copy your results and code and paste to a MS Word, then upload to Moodle.

Exercise 2-2: Feature Map Visualization

- Please download the “2-1_Feature_map_visualization.zip” on the Moodle and choose your own images from Internet.
- Upload the 2-1_Feature_map_visualization.ipynb and imagenet1000_clsidx_to_labels.txt to the Google Colab.
- Compare the probability of the images that contain multi classes and different variations (pose, occlusion, age).
- Please write down results and your codes in MS Word to the Moodle



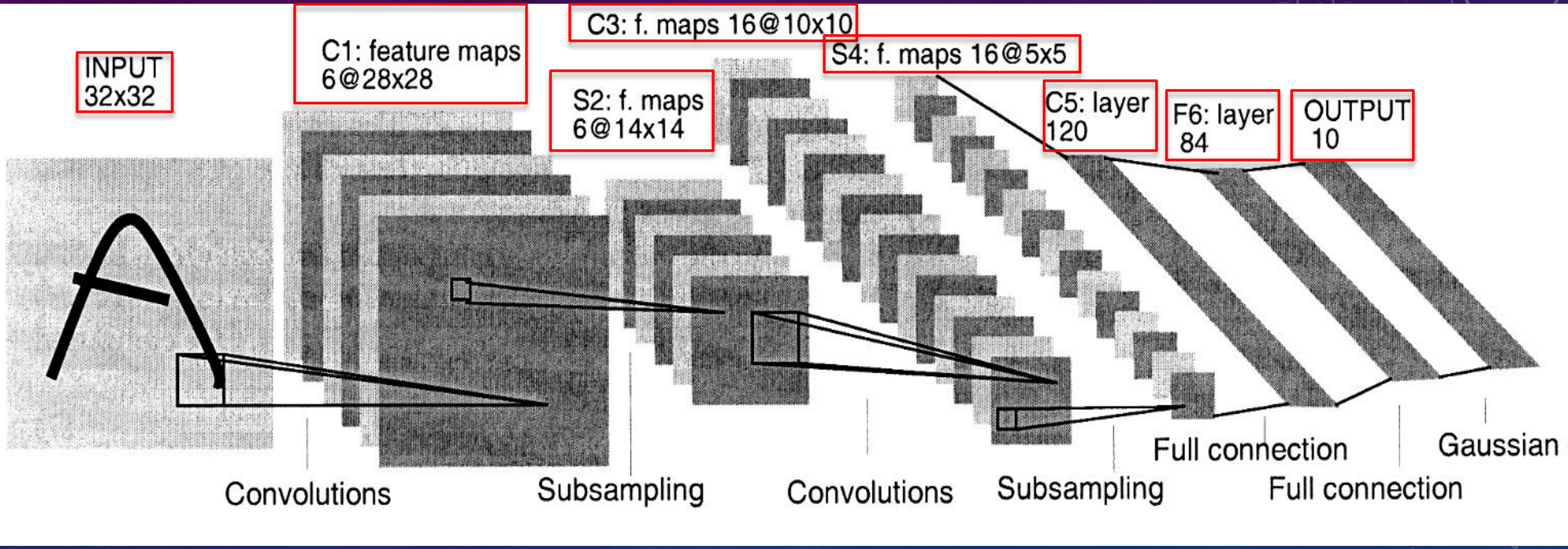

Exercise 2-3: Feature Comparison

- Please download the “2-3_Feature_map_visualization.zip” on the Moodle
- Upload the 2-3_Feature_map_visualization.ipynb and imagenet1000_clsidx_to_labels.txt to the Google Colab.
- Please use the ResNet-50 pretrained model provided by Pytorch Package.
- Compare the similarity of the images that contain two classes and different variations (pose, occlusion, age).
- Please write down result and your code in MS Words to the Moodle



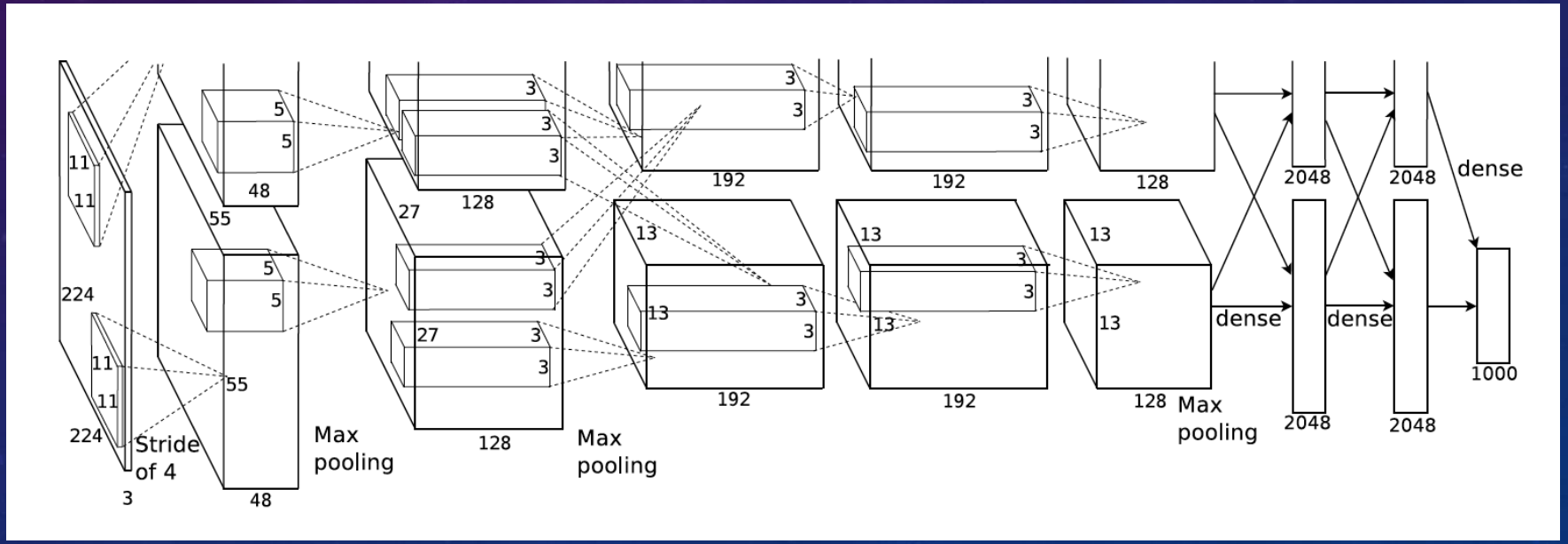

Architecture Introduction

The architecture of LeNet5

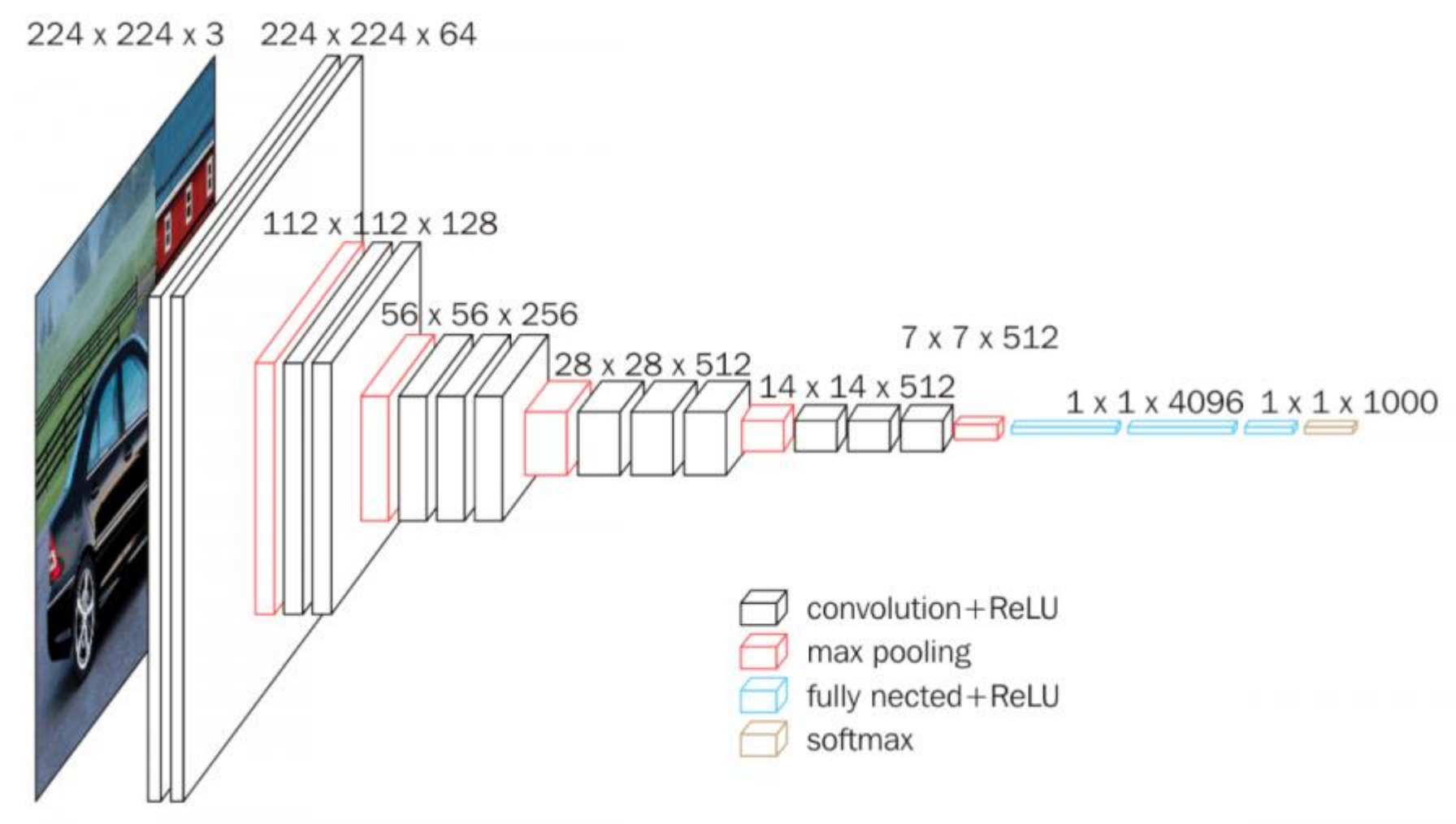


AlexNet

- Use Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
- Use dropout instead of regularisation 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%, repectively.

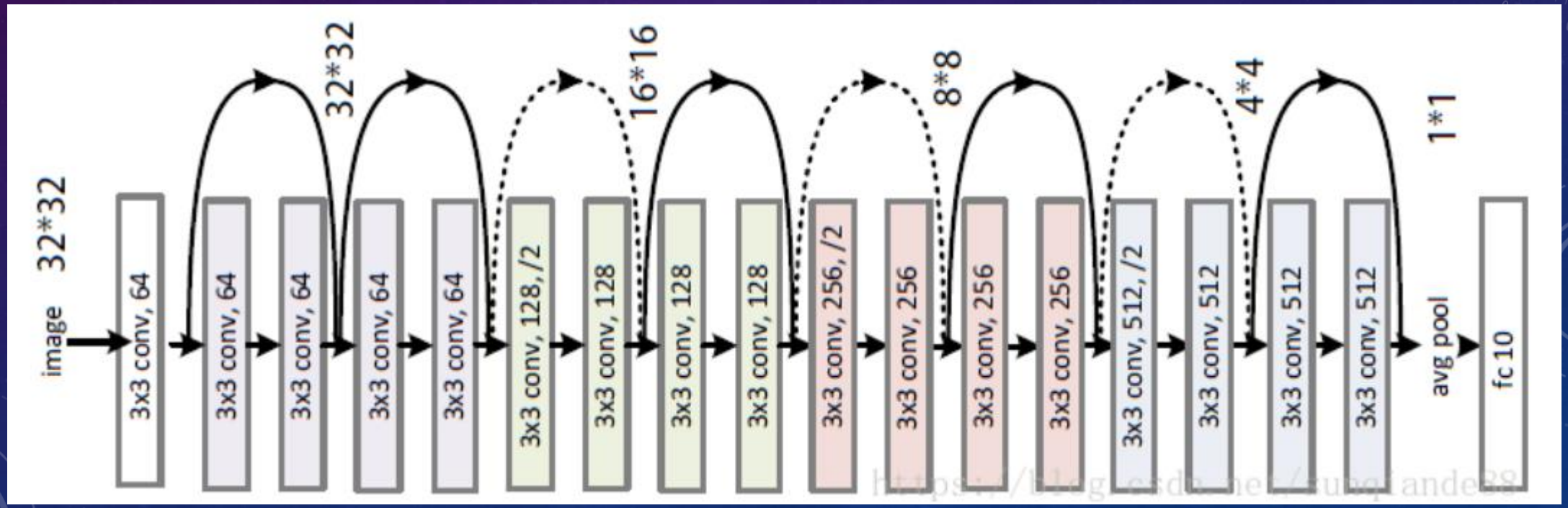


ILSVRC-2014 VGG Model



ResNet18

- 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:



LeNet Parameters

Layer Name	Input W×H×D	Kernel W×H×D/S	Output W×H×D	Params
C1: conv2d	32×32×1	5×5×6	28×28×6	1×5×5×6+6 =156 weights biases
S2: pool/2	28×28×6	2×2/2	14×14×6	0
C3: conv2d	14×14×6	5×5×16	10×10×16	6×5×5×16+16 =2,416
S4: pool/2	10×10×16	2×2/2	5×5×16	0
C5: conv2d	5×5×16	5×5×120	1×1×120	16×5×5×120+120 =48,120
F6: conv2d	1×1×120	1×1×84	1×1×84	120×1×1×84+84 =10,164
F7: conv2d	1×1×84	1×1×10	1×1×10	84×1×1×10+10 =850
			Total	61,706

AlexNet Parameters

AlexNet Network - Structural Details												
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
Total													**62,378,344**

VGG16 Parameters

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Convolution2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Convolution2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Convolution2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Convolution2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Convolution2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
flatten (Flatten)	(None, 25088)	0	block5_pool[0][0]
fc1 (Dense)	(None, 4096)	102764544	flatten[0][0]
fc2 (Dense)	(None, 4096)	16781312	fc1[0][0]
predictions (Dense)	(None, 1000)	4097000	fc2[0][0]

Total params: 138357544

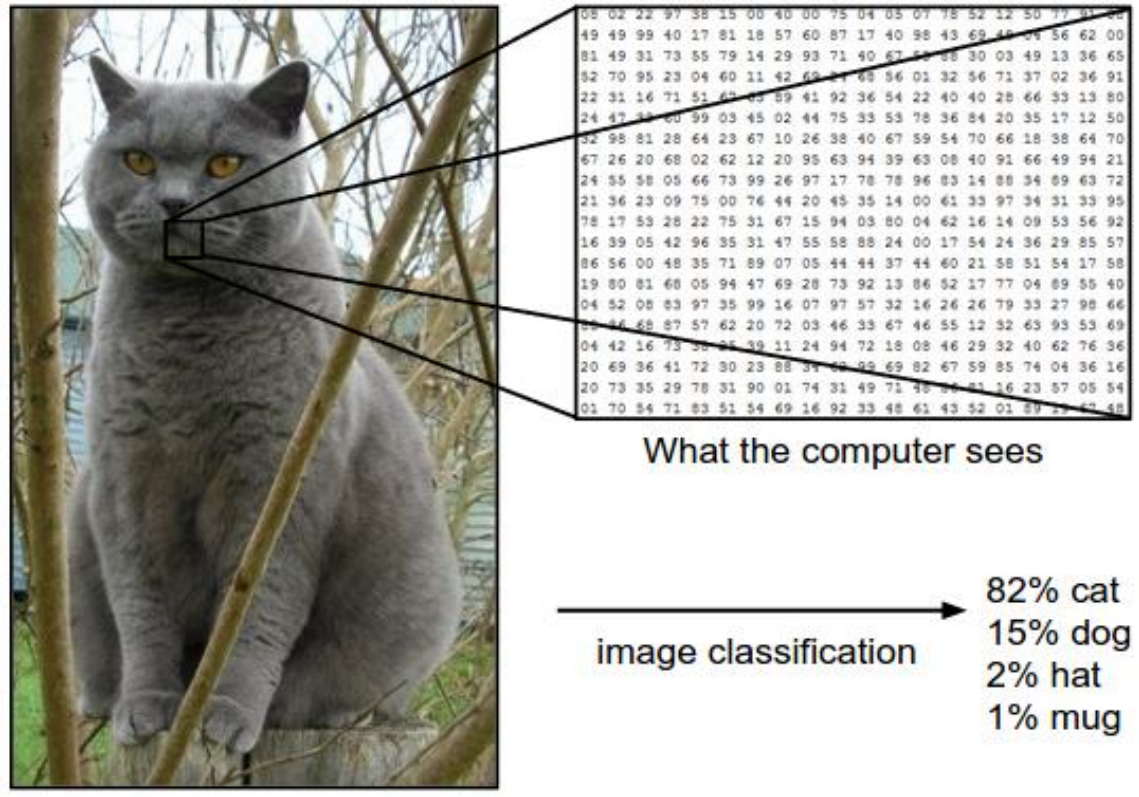
ResNet18 Parameters

ResNet18 - Structural Details														
#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
Total														11,511,784

Image Classification

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.

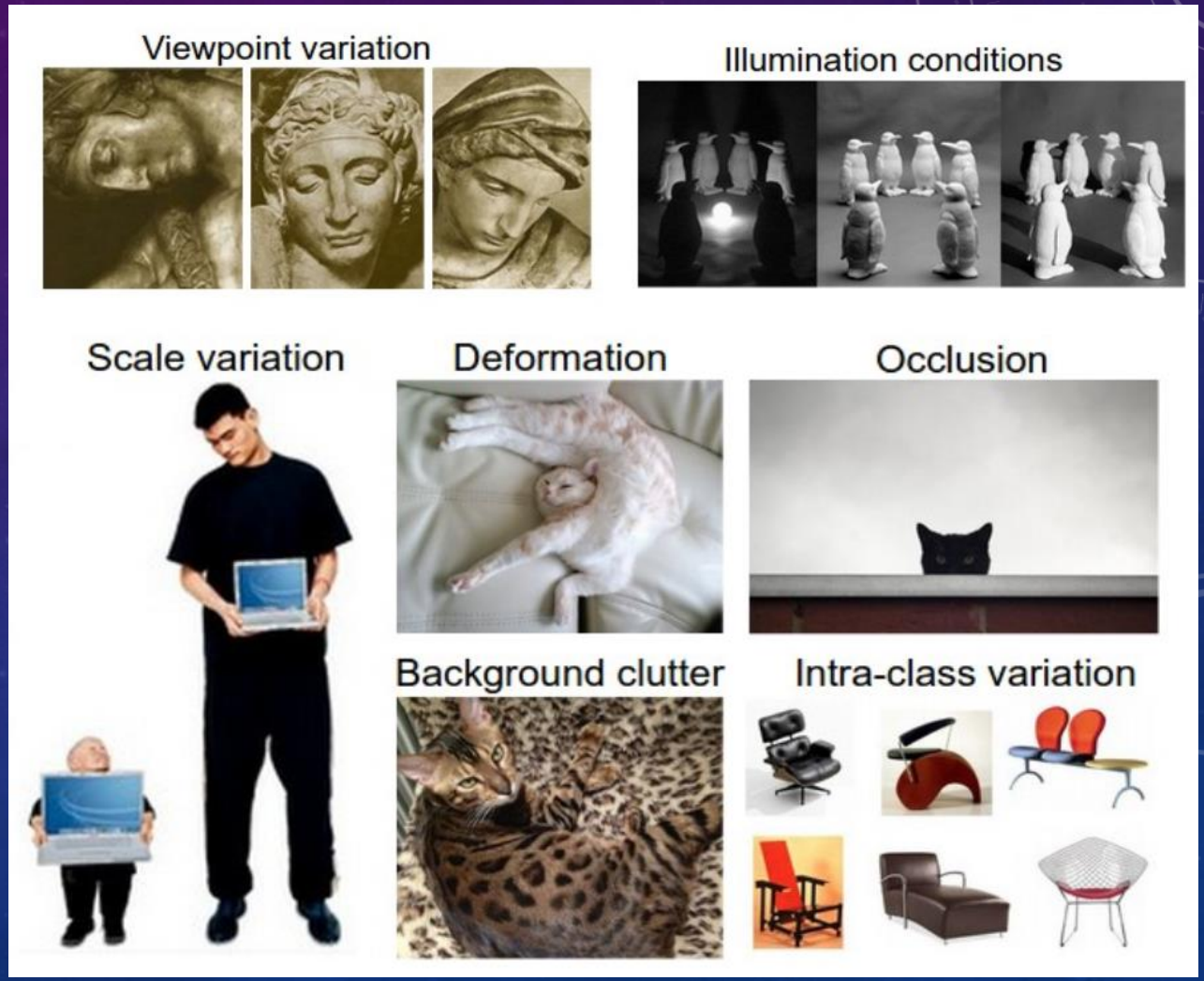


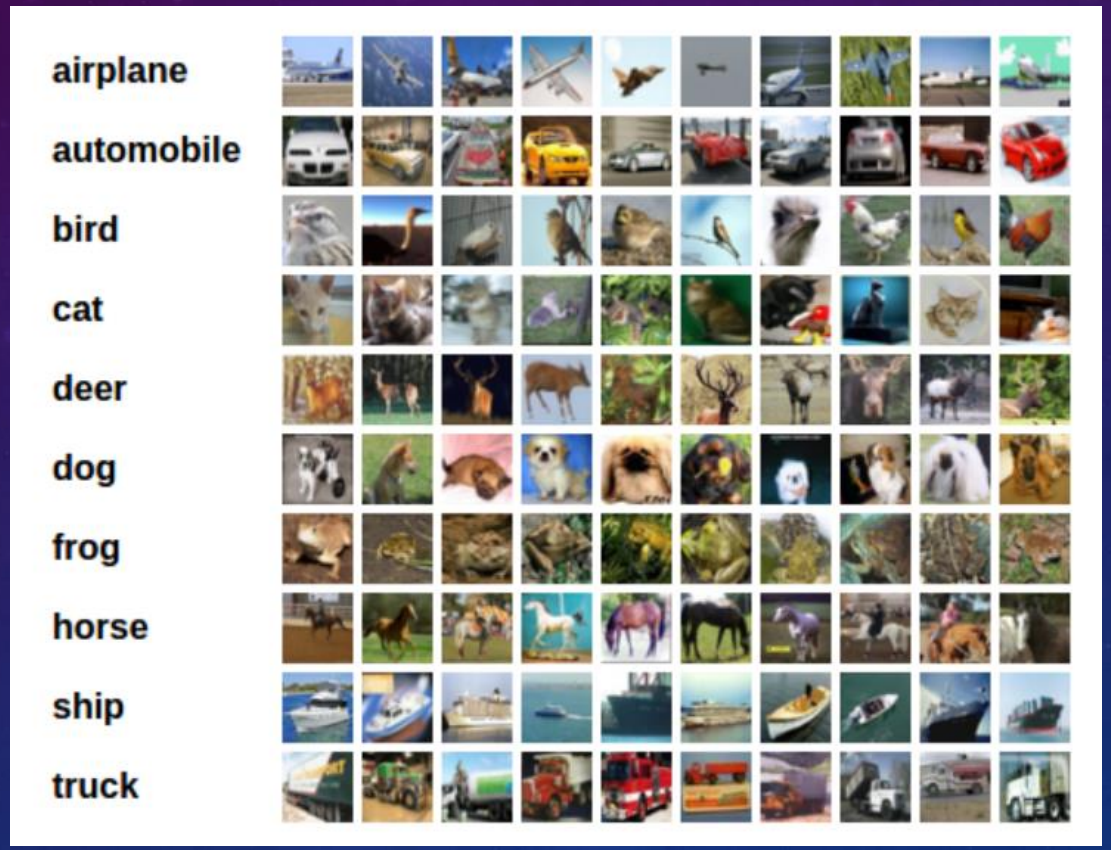
Image Classification

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

Exercise 2-4 – Build A Classifier

- Please install matplotlib with the command “pip install matplotlib”
- Download 2-4_CIFAR10.py and change the following parameters:
  - Epoch
  - Learning Rate: 0.1, 0.01, 0.001
- Please upload your result and observations in MS Words to the Moodle.

Hyperparameter optimization

- Hyperparameters we can adjust
  - Network architecture
  - Learning rate, its decay schedule, update type
  - Regularization methods
- Observe the loss curve and your image outputs to make improvements