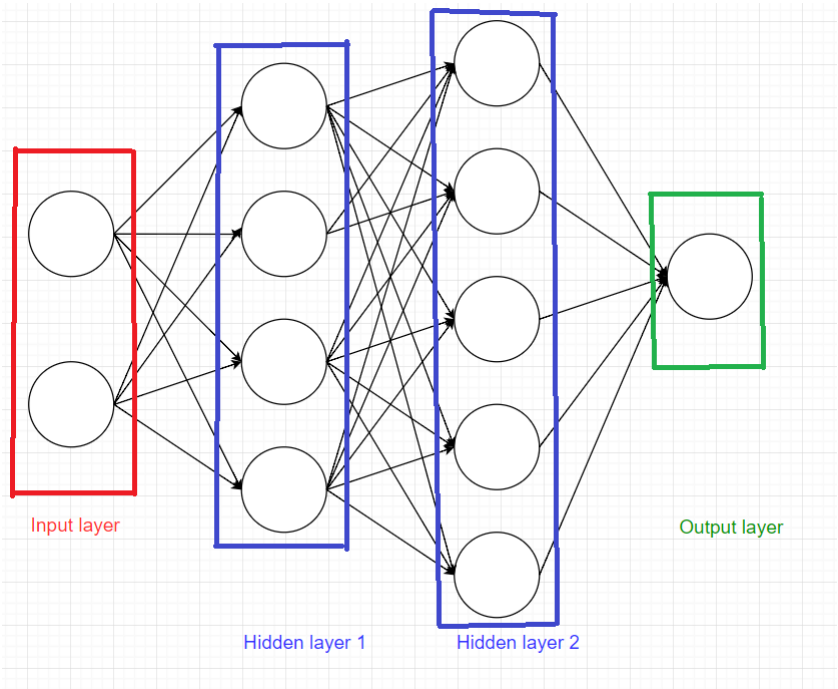
Introduction to Deep Learning

# Convolutional Neural Network (CNN)

# Problem

- Input: color image of size 64 * 64
- Output: image contains human face or not
- Method: ?

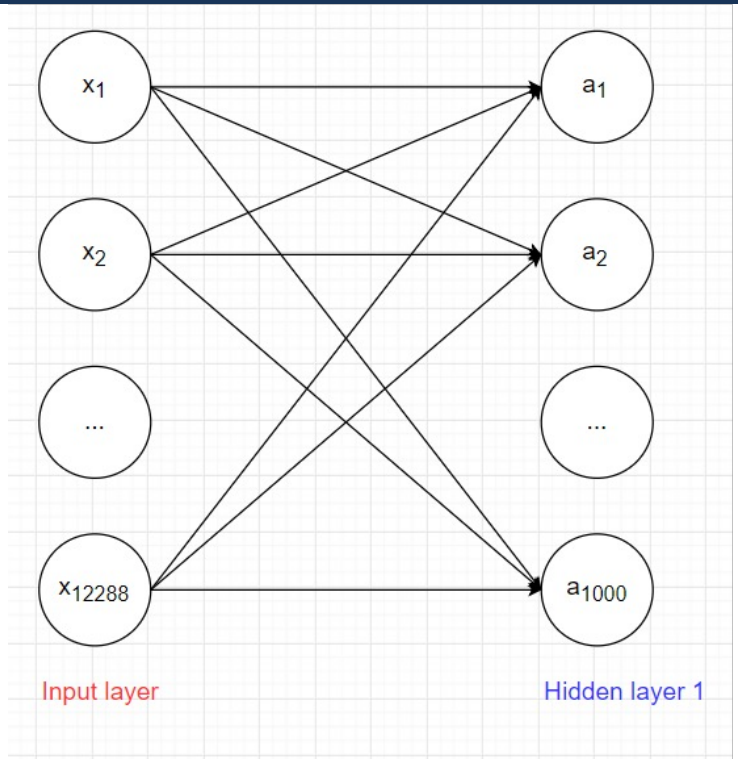
# Neural Network Model




- Each hidden layer is called a fully connected layer (or Dense layer)
- Each node in hidden layer is connected to all nodes in the previous layer

 Fully Connected Neural Network (FCN)

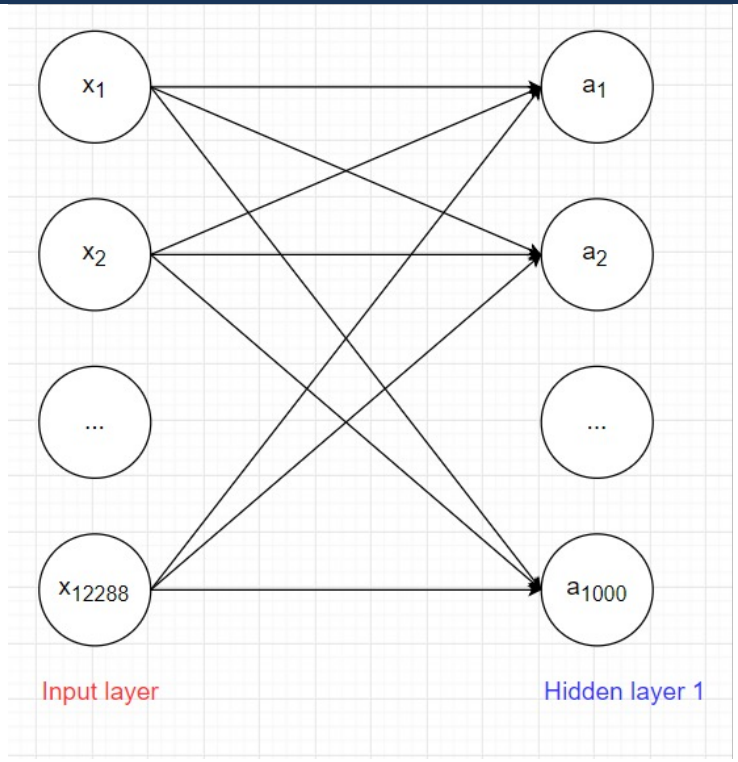
# Problem of Fully Connected Neural Network




- Color image size 64 * 64 needs 64 * 64 * 3 pixels

  ⍰ Input layer will have 12,288 values

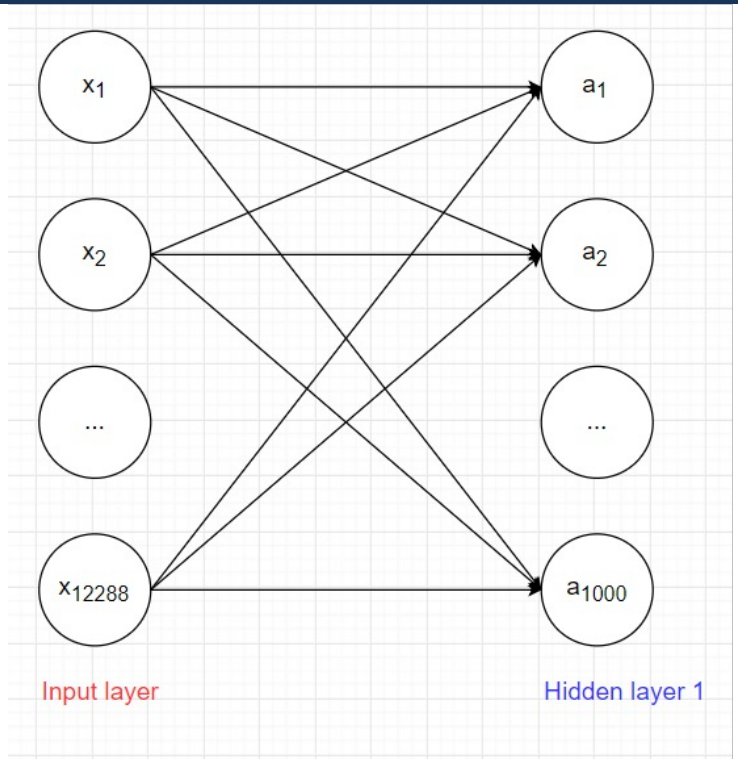
# Problem of Fully Connected Neural Network




- Color image size 64 * 64 needs 64 * 64 * 3 pixels
- Hidden layer 1 has 1000 nodes

→ # of weights is 12,288,000 + # of bias is 1000
→ # of paramemers between input layer and hidden layer 1 is 12,289,000
→ What happen if we have 10 hidden layers and the size of image is 512 x 512?

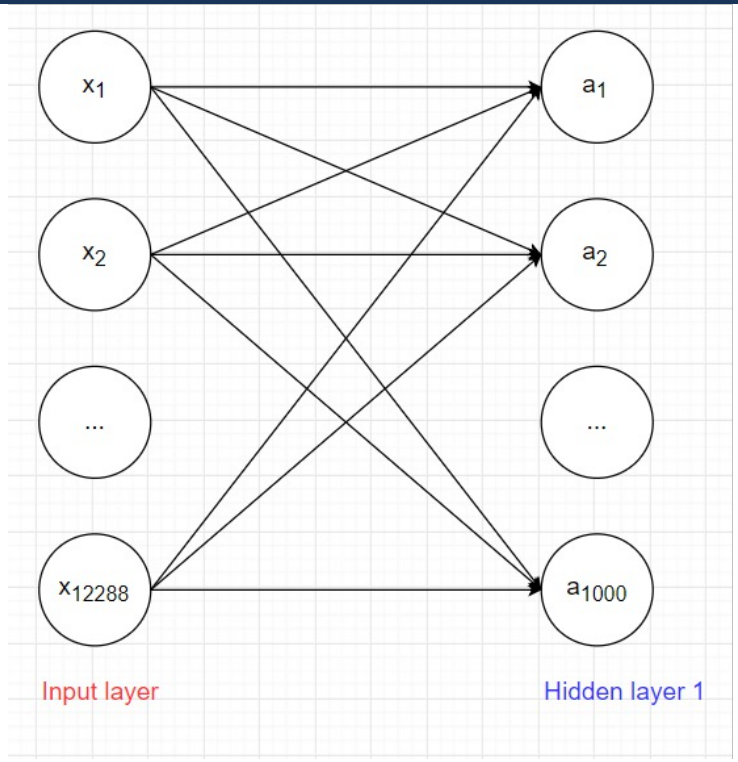
# Problem of Fully Connected Neural Network




- Color image size 64 * 64 needs 64 * 64 * 3 pixels
- Hidden layer 1 has 1000 nodes

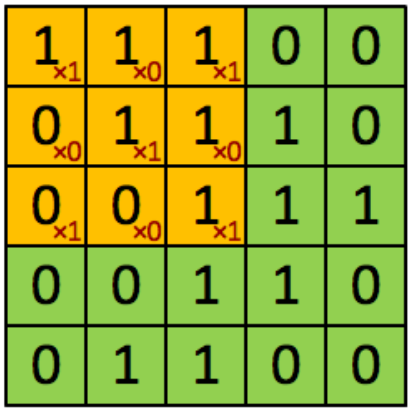
→ # of weights is 12,288,000 + # of bias is 1000
→ # of parameters between input layer and hidden layer 1 is 12,289,000
→ What happen if we have 10 hidden layers and the size of image is 512 x 512? extremely large number of parameters to learn !!!

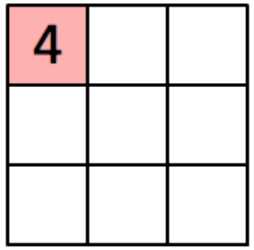
# Problem of Fully Connected Neural Network




- Color image size 64 * 64 needs 64 * 64 * 3 pixels
- Spatial organization of the input is preserved until flatten ⍰ do it efficient for large images?

# Convolution in a neural network

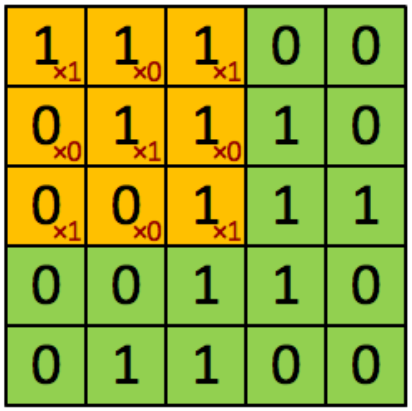


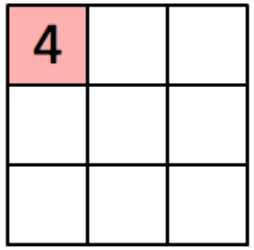




- Neuron depends only on a few local input neurons

 Similarly to the local connectivity of visual features in images

# Convolution in a neural network

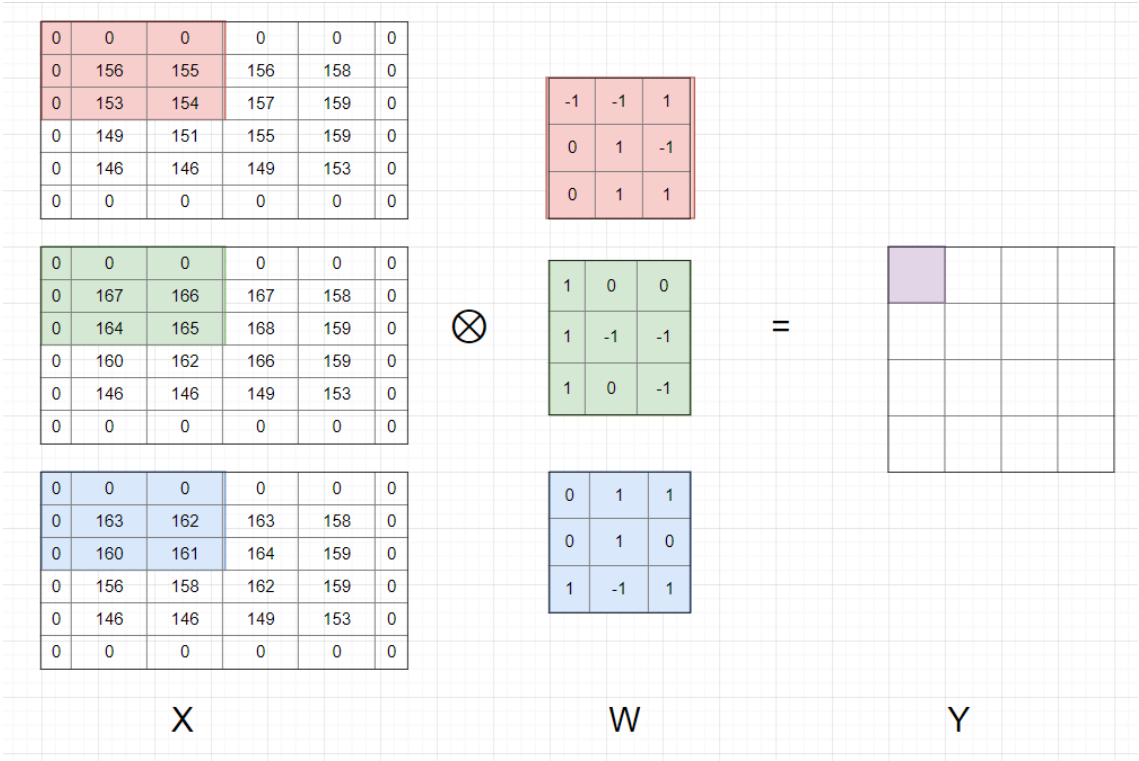



- *x* is a *3x3* chunk (yellow area) of the image (green area)
- Each output neuron is parametrized with the *3x3* weight matrix w (small red numbers in yellow area)
- Output image contains convolved features (in pink)

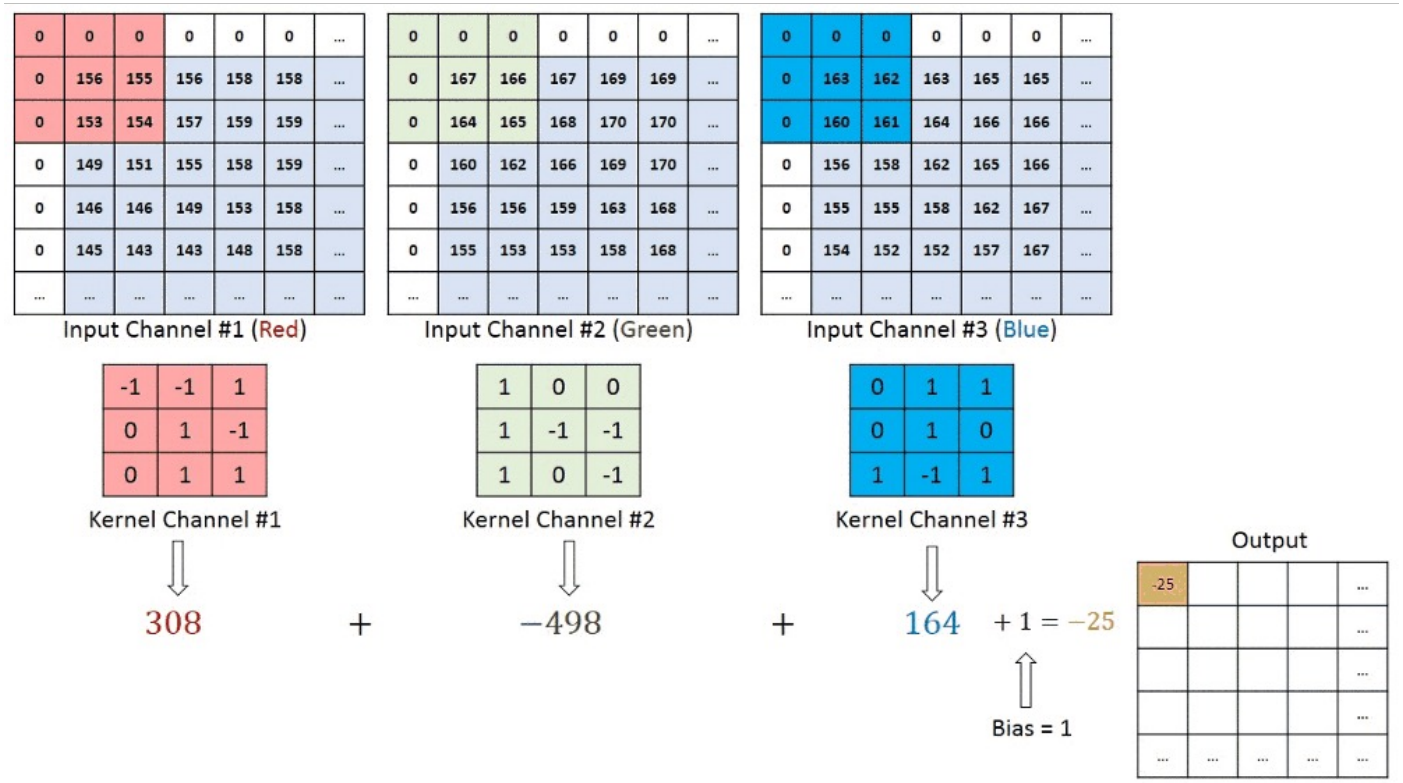
 The process is performed by sliding the 3x3 window through the image

# Convolution in color image



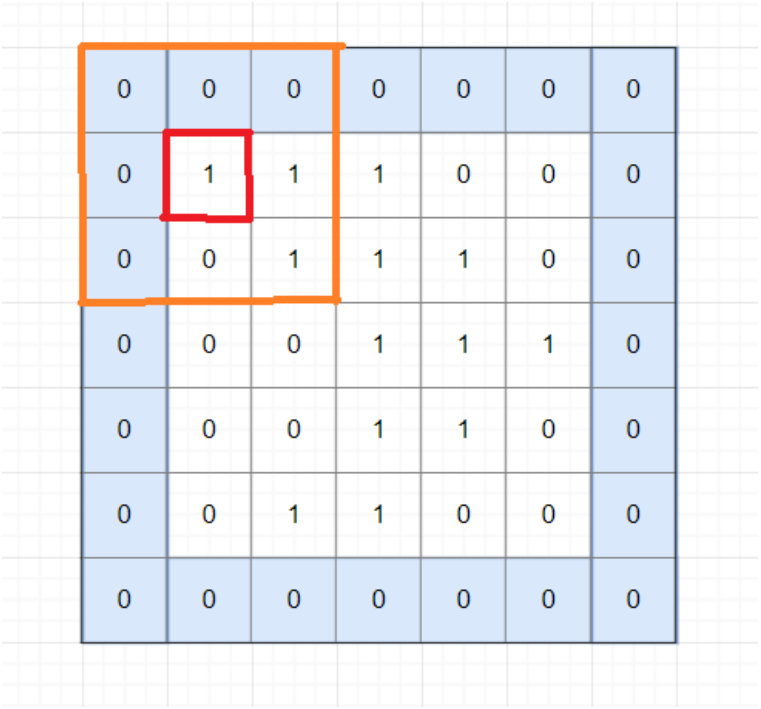

$$y_{11} = b + (x_{111} * w_{111} + x_{121} * w_{121} + x_{131} * w_{131} + x_{211} * w_{211} + x_{221} * w_{221} + x_{231} * w_{231} + x_{311} * w_{311} + x_{321} * w_{321} + x_{331} * w_{331}) + (x_{112} * w_{112} + x_{122} * w_{122} + x_{132} * w_{132} + x_{212} * w_{212} + x_{222} * w_{222} + x_{232} * w_{232} + x_{312} * w_{312} + x_{322} * w_{322} + x_{332} * w_{332}) + (x_{113} * w_{113} + x_{123} * w_{123} + x_{133} * w_{133} + x_{213} * w_{213} + x_{223} * w_{223} + x_{233} * w_{233} + x_{313} * w_{313} + x_{323} * w_{323} + x_{333} * w_{333}) = -25$$

# Convolution in color image



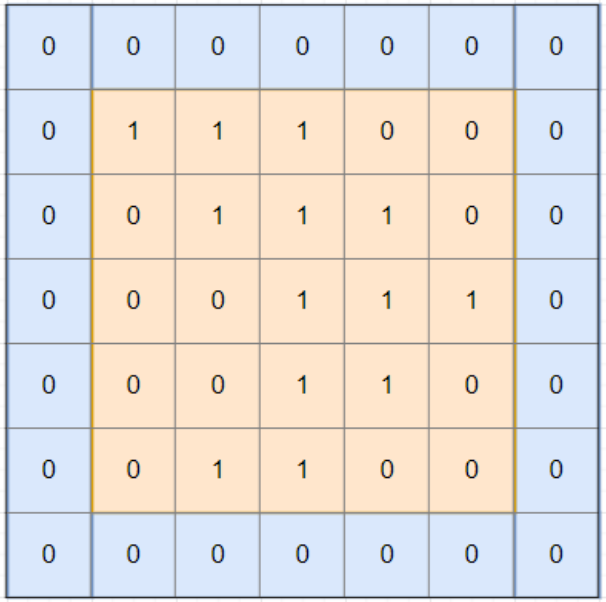

- Output Y of convolution operation with color image is a matrix
- Bias = 1 is added to the operation

# Padding

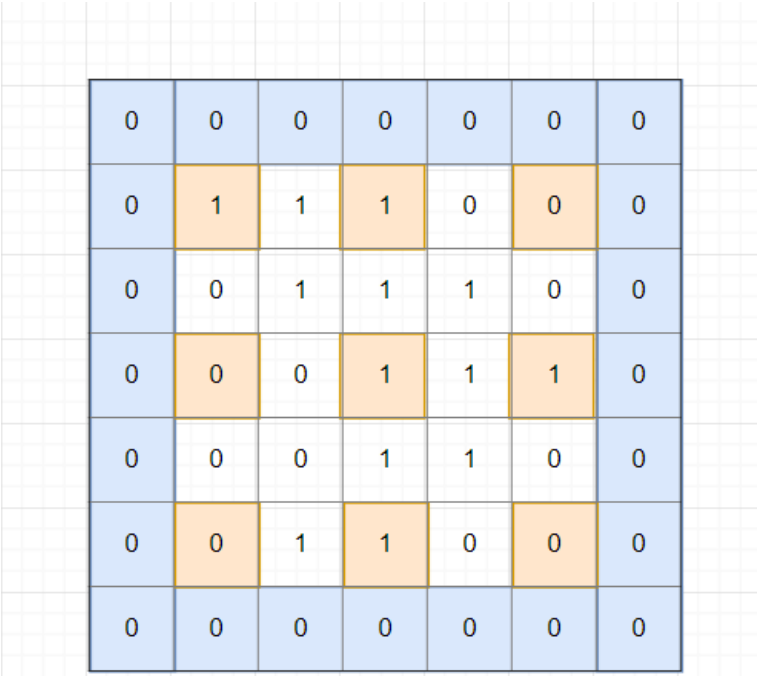



Border of the image is added with zero values

# Stride

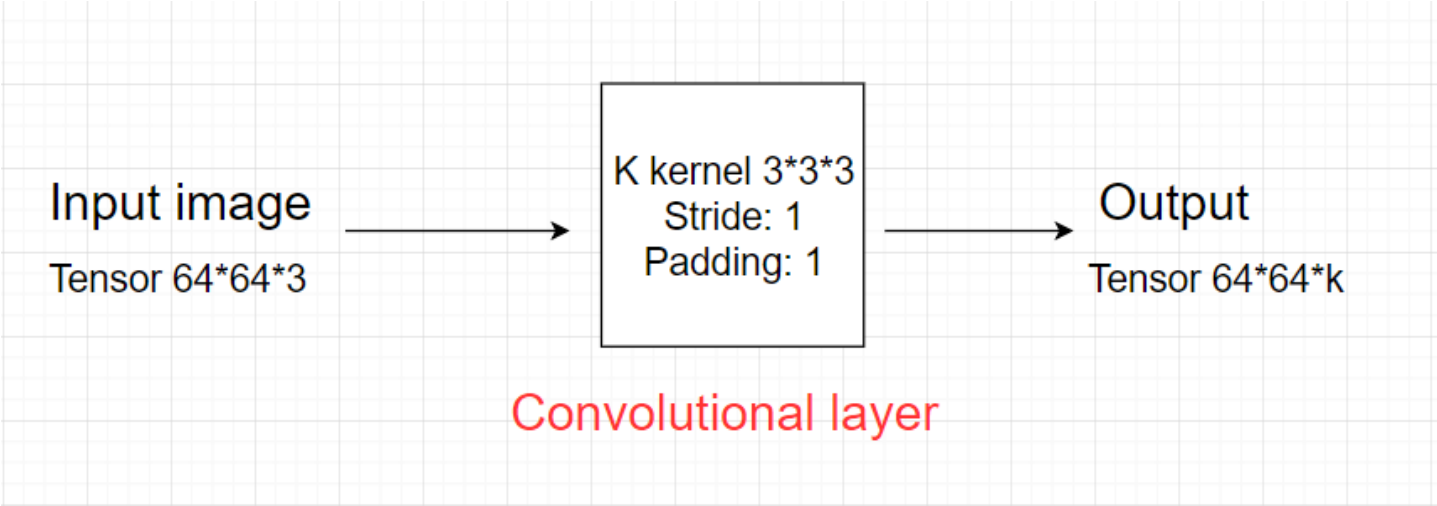



Padding = 1, stride = 1



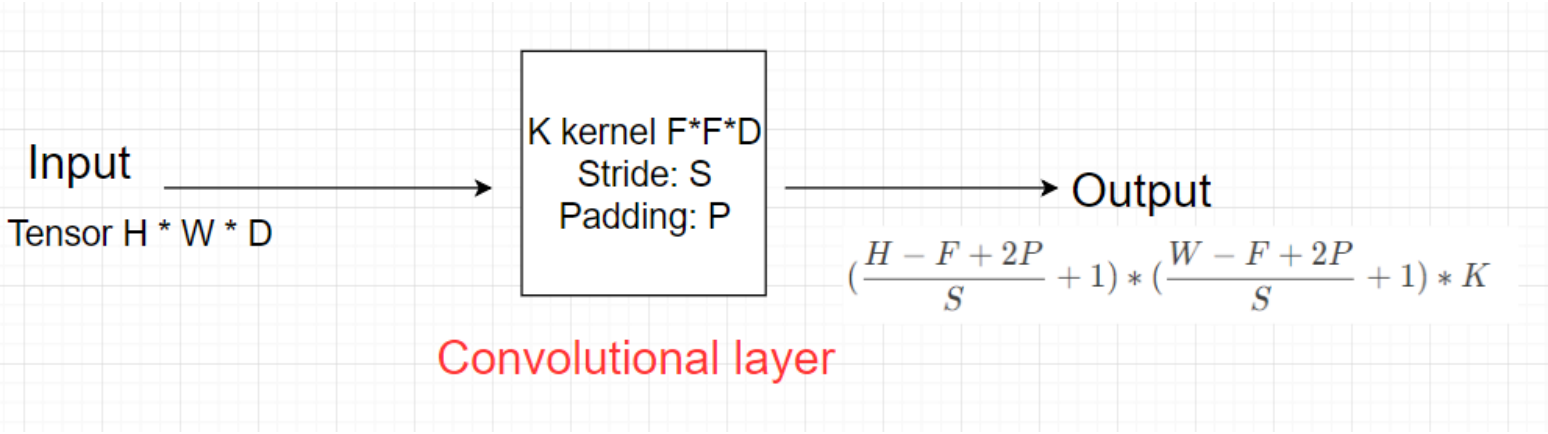

Padding = 1, stride = 2

# First Convolutional Layer



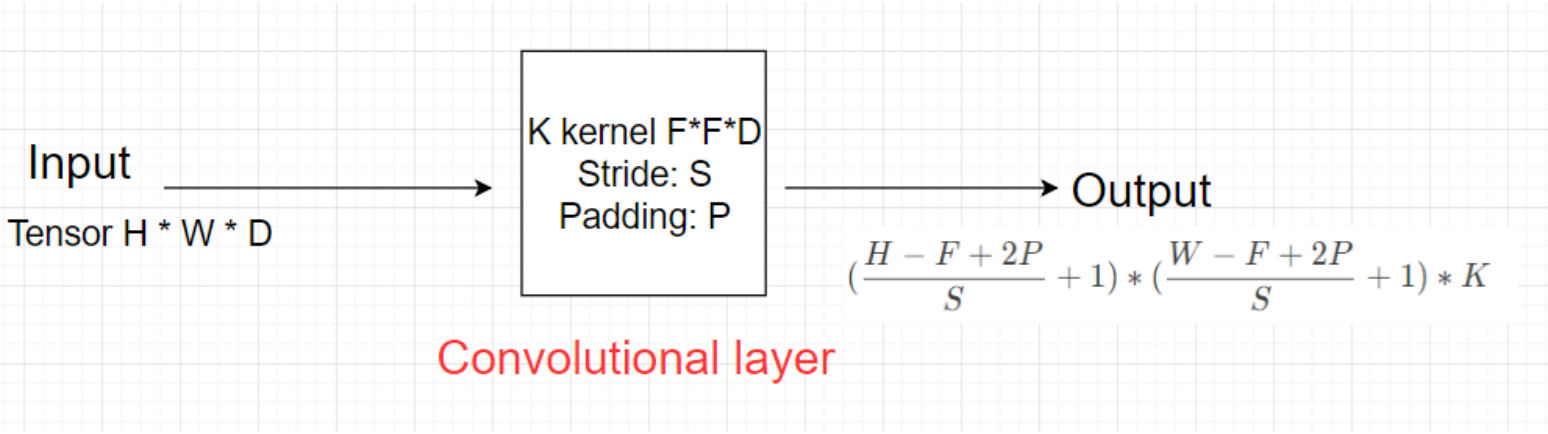

- Output of the first convolutional layer will be input of the next convolutional layer

# General Convolutional Layer



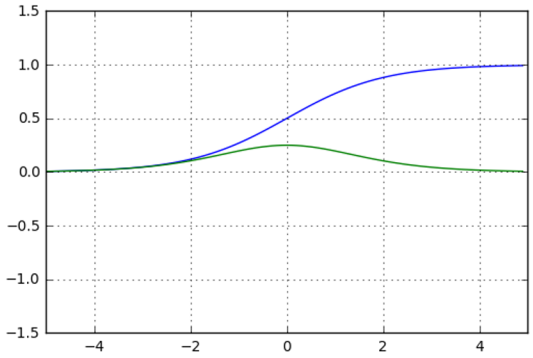

 # of parameters of each kernel is F*F*D + 1 (for bias)  # of parameters of layer is K * (F*F*D + 1)

# General Convolutional Layer



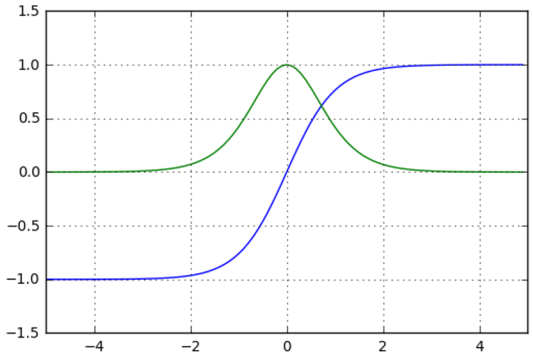

- Output of the convolutional layer will be applied with a non-linear activation function before being the input of the next convolutional layer

# Element-wise activation functions



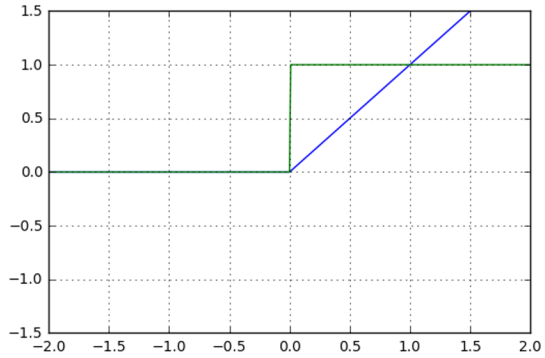

$$\text{sigm}(x) = \frac{1}{1+e^{-x}}$$

$$\text{sigm}'(x) = \text{sigm}(x)(1-\text{sigm}(x))$$




$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

$$\tanh'(x) = 1-\tanh(x)^2$$



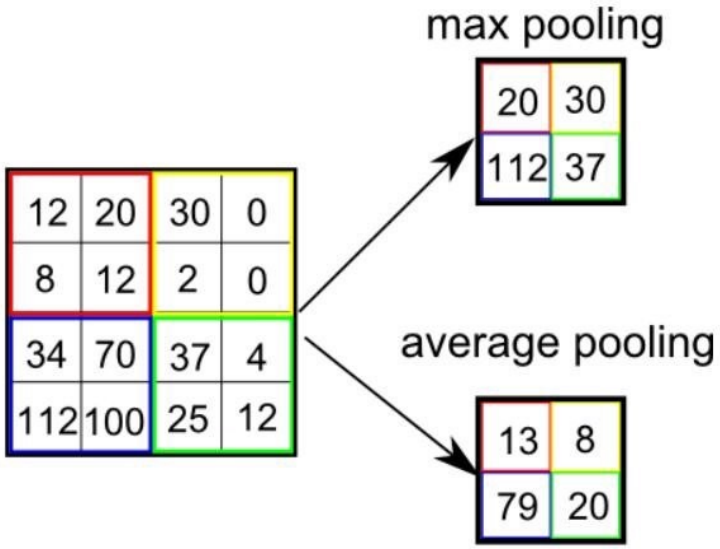

$$\text{relu}(x) = \max(0,x)$$

$$\text{relu}'(x) = 1_{x>0}$$

- Blue line: activation function
- Green line: derivative

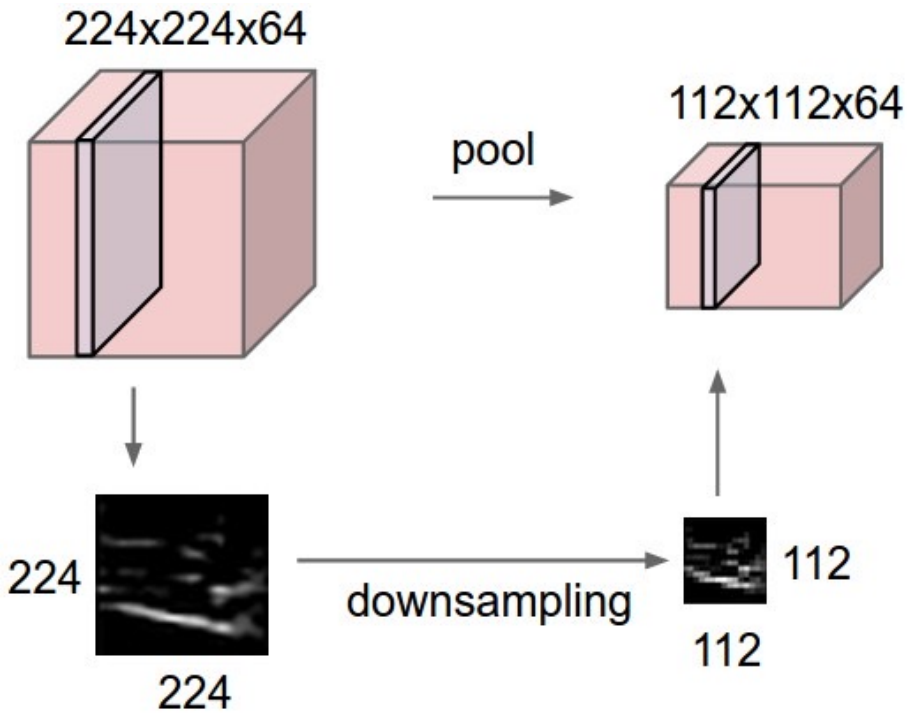
 Relu activation function is often used after each convolutional layer since it is an efficient activation function without heavy computation

# Pooling Layer



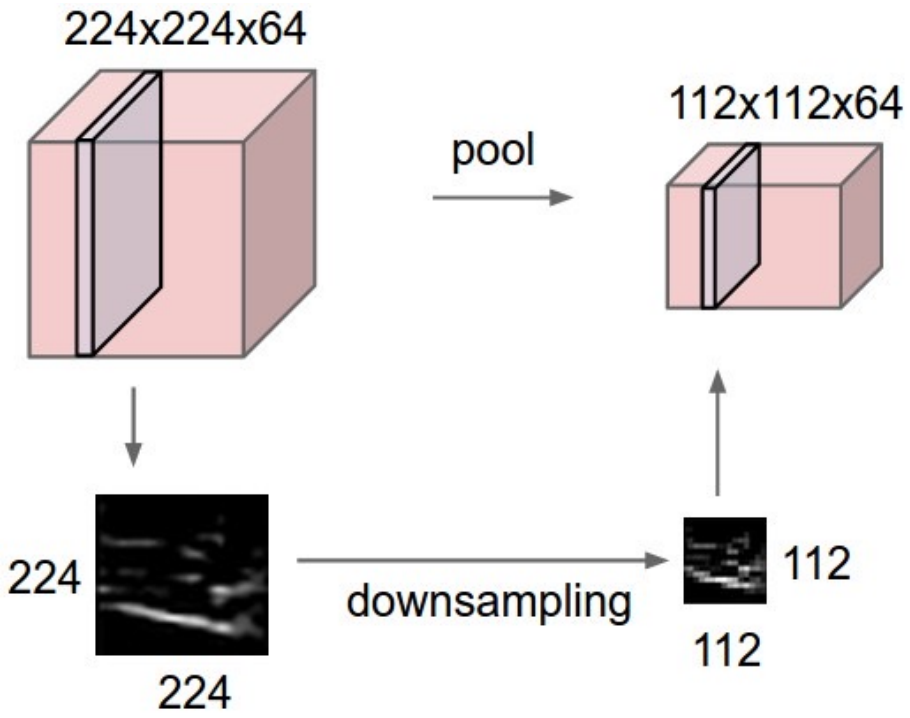

- Pooling layer is placed between two convolutional layers to reduce sizes of output data and still preserve the important features of images

# Pooling Layer



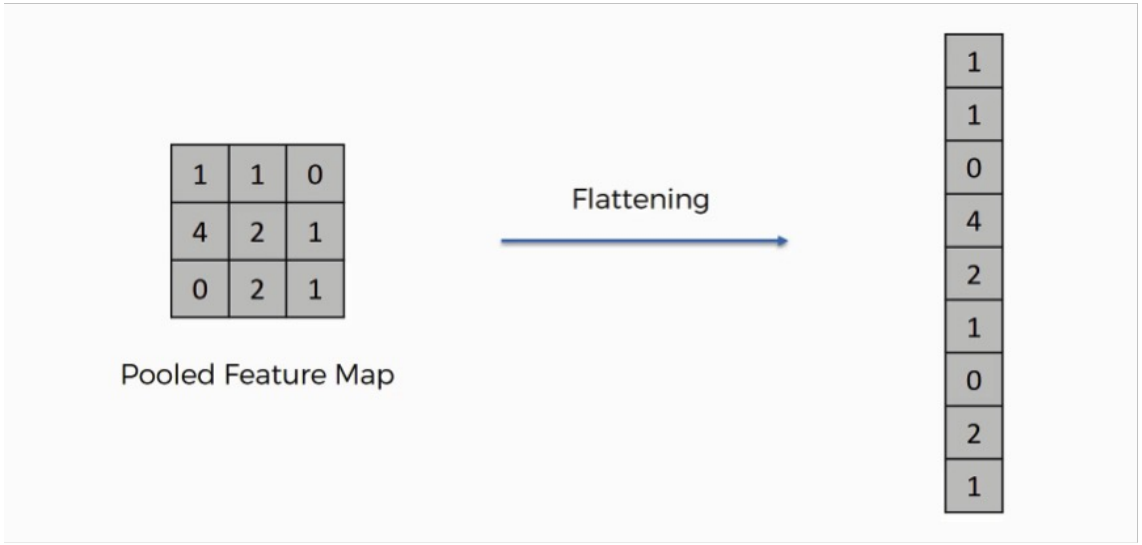

- In practice, pooling layer with size = (2,2), stride = 2 and padding = 0 is often used so that output width and height of data are reduced half while depth is unchanged

# Pooling Layer



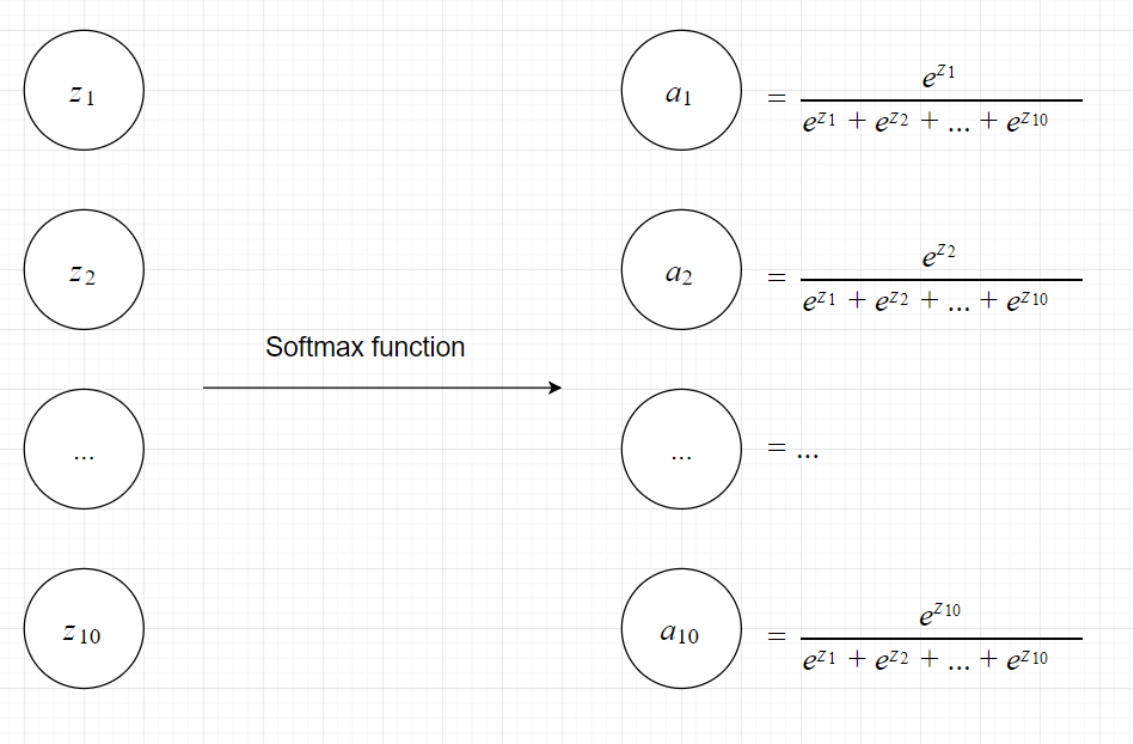

- In practice, pooling layer with size = (2,2), stride = 2 and padding = 0 is often used so that output width and height of data are reduced half while depth is unchanged
- Note: in some models, convolutional layer with stride > 1 is used to reduce data sizes instead of pooling layer

# Fully Connected Layer




- Tensor of output of last layer with size (H*W*D) is flatten to the vector with size (H*W*D,1)
- The fully connected layers are then applied to this vector to combine different image features learned by convolutional layers to produce output of the model

# Softmax activation function



# Softmax Function

- Softmax function formula:

$$a_k = \frac{e^{z_k}}{\sum_{i=1}^{10} e^{z_i}}$$

In which:

- $\sum_{i=1}^{10} a_i = 1$
- $0 < a_i < 1$

--> Each value ($a_i$) in the output of the softmax function is interpreted as the probability of membership for each class

# Softmax Function

- Softmax activation is used to normalize ***the outputs of the last dense layer***, converting them from weighted sum values into probabilities that sum to 1

- Specifically, softmax activation outputs one value for each node in the output layer. The output values are interpreted as probabilities of the membership for each class

# Classic CNN Architecture

**Input**

**Conv blocks:**

- Convolution + activation (relu)
- Convolution + activaton (relu)
- …
- Maxpooling 2x2

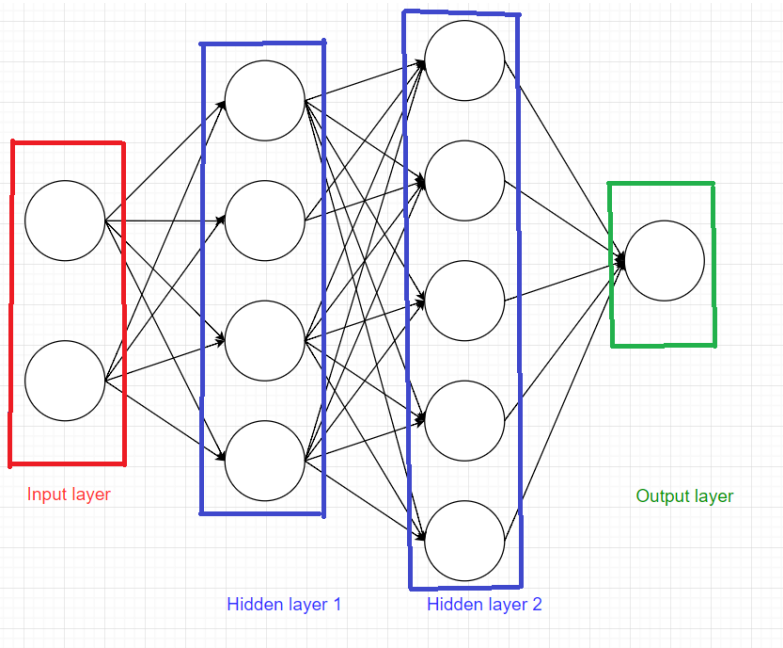
**Output**

- Fully connected layers
- Softmax / Sigmoid activation function

# Classic CNN Architecture

Output:

- Fully connected layers



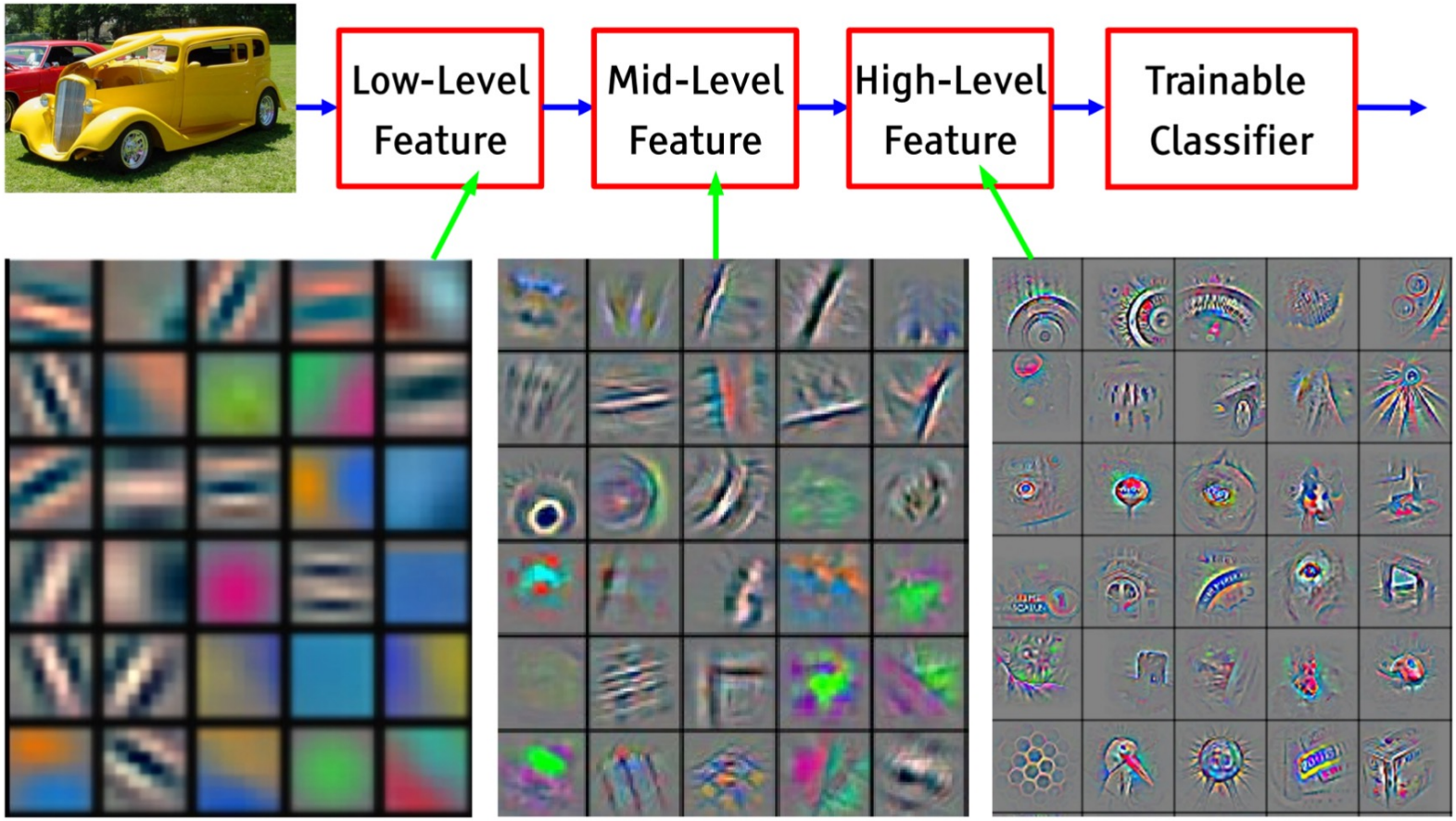

If the last dense layer has ***only one node***:

- Sigmoid activation function is used

If the last dense layer has ***more than one node***:

- Softmax activation function is used

# Feature extraction with CNN




Visualization of image features learned automatically by convolutional layers

# Popular CNN Architectures

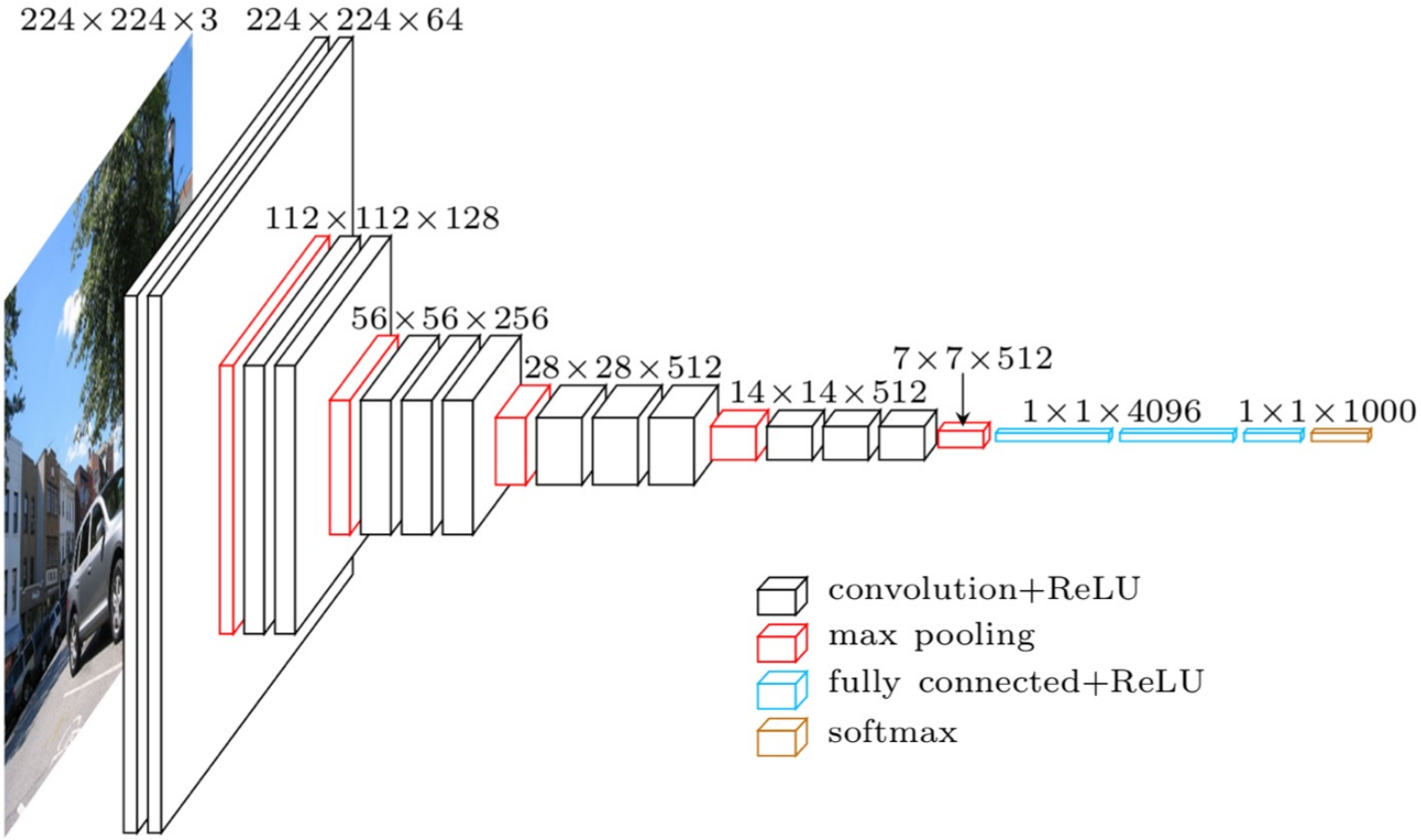
- VGG (Visual Geometry Group)
- ResNet (Residual Network)

# VGG Architecture

- VGG is a deep CNN architecture containing classical blocks of CNN such as convolutional layers (conv), pooling layers (pool) and fully connected layers (fc)
- Network architecture of VGG such as VGG16, VGG19
- VGG is proposed by Simonyan, Karen, and Zisserman in "Very deep convolutional networks for large-scale image recognition." (2014)

# VGG Architecture

- VGG16 architecture:



# VGG Architecture

- VGG16 architecture:

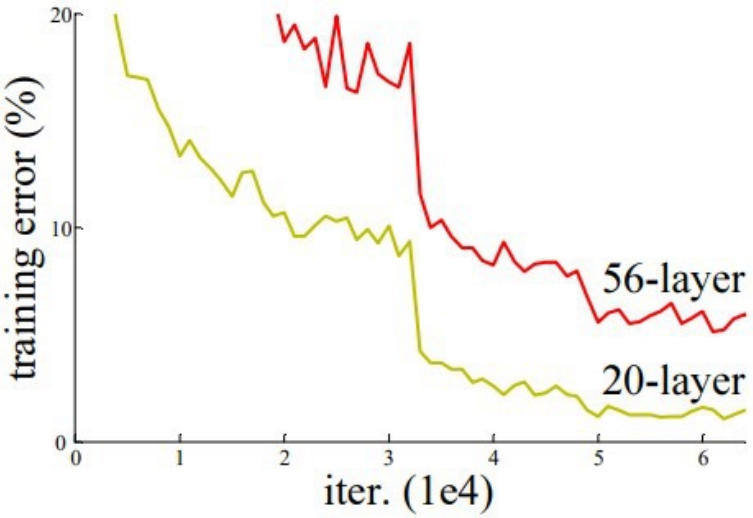

- Conv: size 3x3, padding = 1, stride = 1, # of kernels = 64 or dept of output layer
- Pool/2: max pooling layer with size = 2x2, stride = 2
- fc 4096: fully connected layer with 4096 nodes
- From left to right: size of output features decreases, but depth increases
- After passing through all conv layers and pooling layers, data are flattened and fed into the fc layers

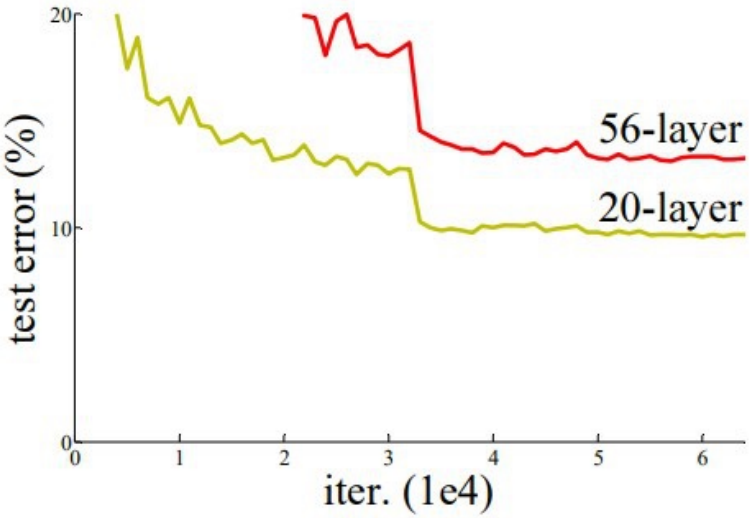
# VGG Architecture

- VGG16 architecture:


- Conv: size 3x3, padding = 1, stride = 1, # of kernels = 64 or dept of output layer
- Pool/2: max pooling layer with size = 2x2, stride = 2
- fc 4096: fully connected layer with 4096 nodes
- From left to right: size of output features decreases, but depth increases
- After passing through all conv layers and pooling layers, data are flattened and fed into the fc layers

# Problem of classical CNN Architecture

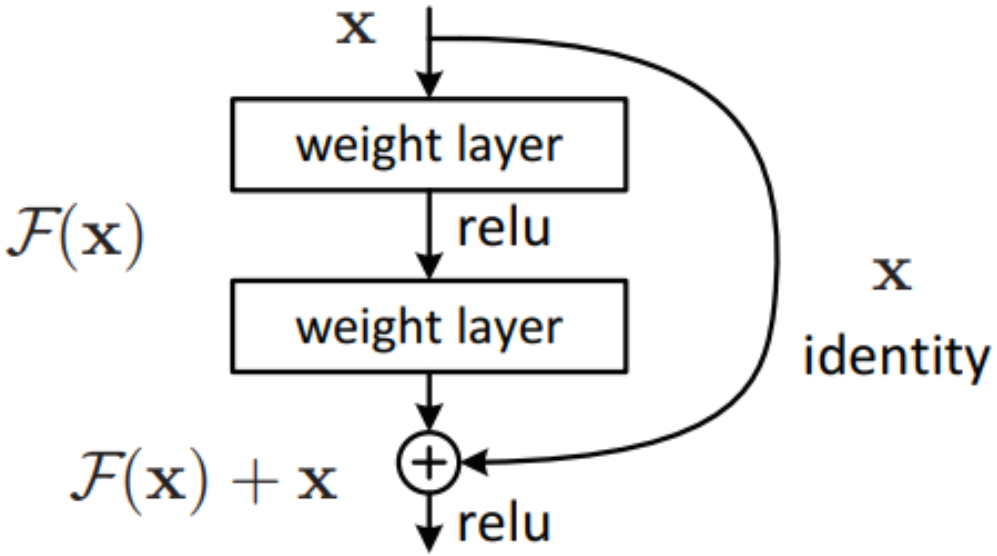



- A 56-layer CNN gives more error rate than a 20-layer CNN in both training and testing dataset
- Problem may cause by ***vanishing or exploding gradient*** (gradient becomes 0 or too large) during the backpropagation process

# ResNet Architecture

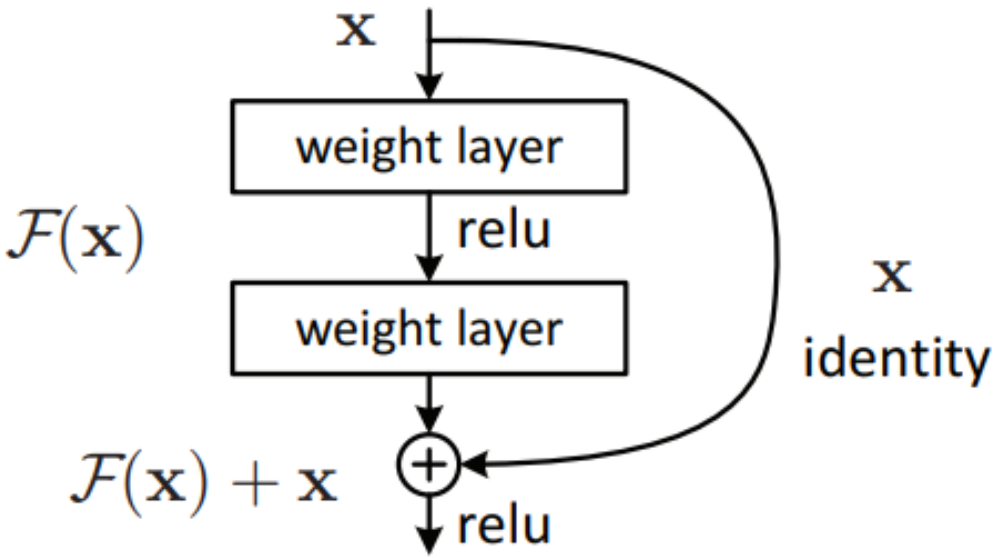
- ResNet introduces the concepts called ***residual block*** *using* ***skip (shortcut) connection***




Residual learning: a building block

# ResNet Architecture

- A ResNet architecture is created by stacking a set of residual blocks together

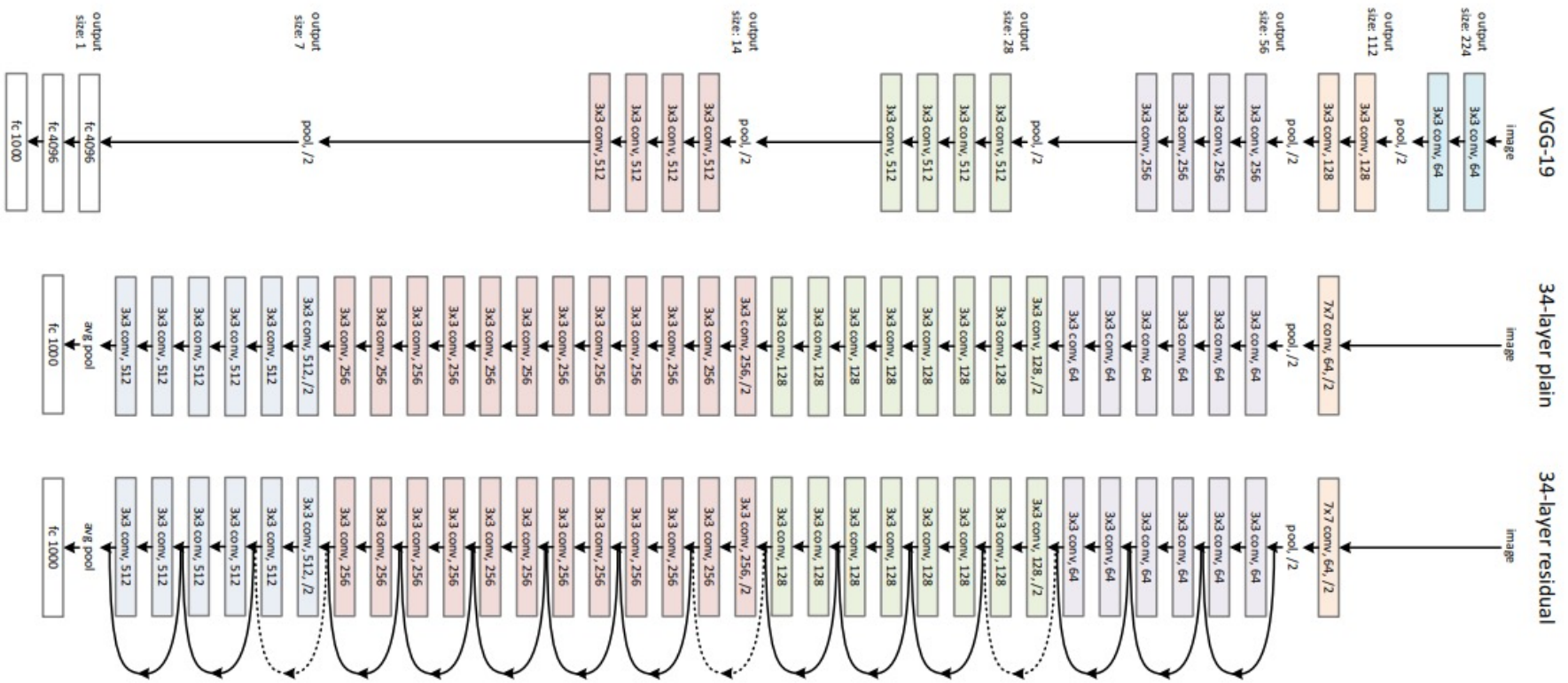



A residual block

# ResNet Architecture

- ResNet solves the problem of vanishing or exploding gradient
- ResNet is able to support hundreds or thousands of convolutional layers
- Proposed by He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.

# Network Architecture of ResNet

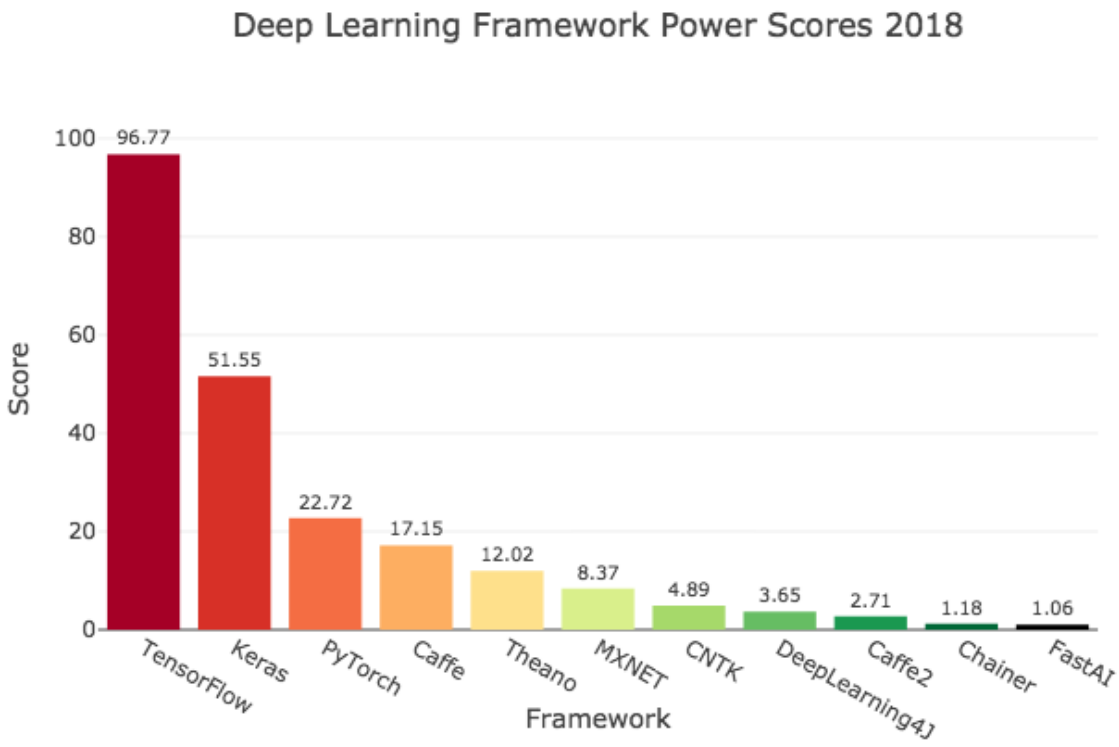



ResNet-34 architecture

# VGG19 vs. ResNet50

| | VGG19 | ResNet50 |
|---|---|---|
| Accuracy | 5.25 top-5 error | 7.1% |
| Parameters | 25M | 138M |
| Computational complexity | 3.8B Flops | 15.3B Flops |
| Convolution | Fully convolution until the last layer | Contains several fully connected layers |

# Tools and Framework




- TensorFlow is most popular, but difficult to use
- Keras is easier to use with high level APIs. Keras can be run on top of TensorFlow, Theano, CNTK

# Solving AI Problem with Deep Learning

1. Problem definition
2. Dataset preparation
3. Model construction
4. Loss function definition
5. Apply backpropagation and gradient descent to find parameters (weight and bias) to optimize loss function (noted that another optimizer can also be used)
6. Predict new output with new data using learnt parameters and weights

# Deep Learning Datasets

- You can visit this website to get information of the dataset: https://paperswithcode.com/datasets
- Kaggle is also useful source for you to get datasets for deep learning models

# Exercises 3

- **TODO1:** Download and study the properties of "**Dogs and Cats**" dataset for binary classification problem

- **TODO2:** Download and study the properties of "**MNIST**" dataset for multi-class classification problem

- **TODO3:** Write a short report to express your understanding of each dataset