



# LECTURE 10

## OBJECT RECOGNITION

Punnarai Siricharoen, Ph.D.

## OBJECTIVES

- Understand and be able to describe fundamental workflows of object recognition using traditional approach and modern approach

## TODAY'S CONTENT

- Introduction
- Features – Histogram of oriented gradients
- Neural networks
- Convolutional Neural networks
- Demo: handwritten digits recognition
  - Traditional approach
  - Modern approach



boltplusbumblebee

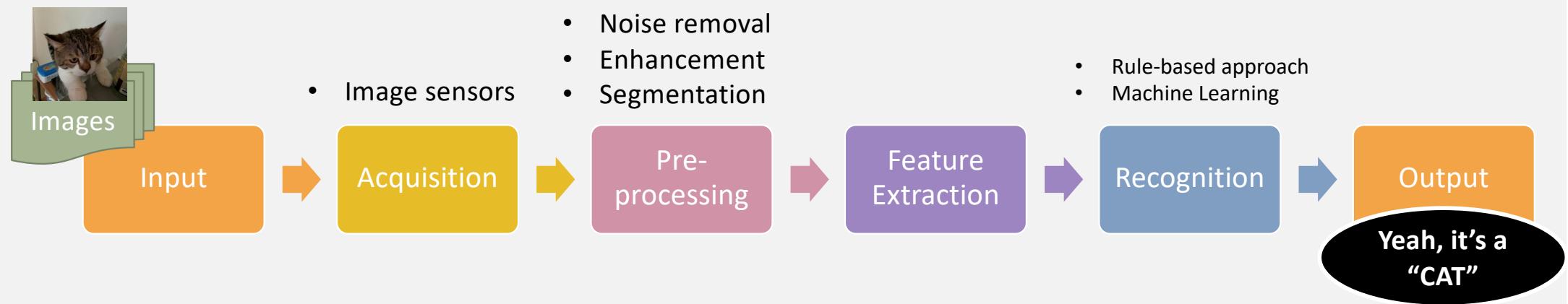


Britannica : Lion

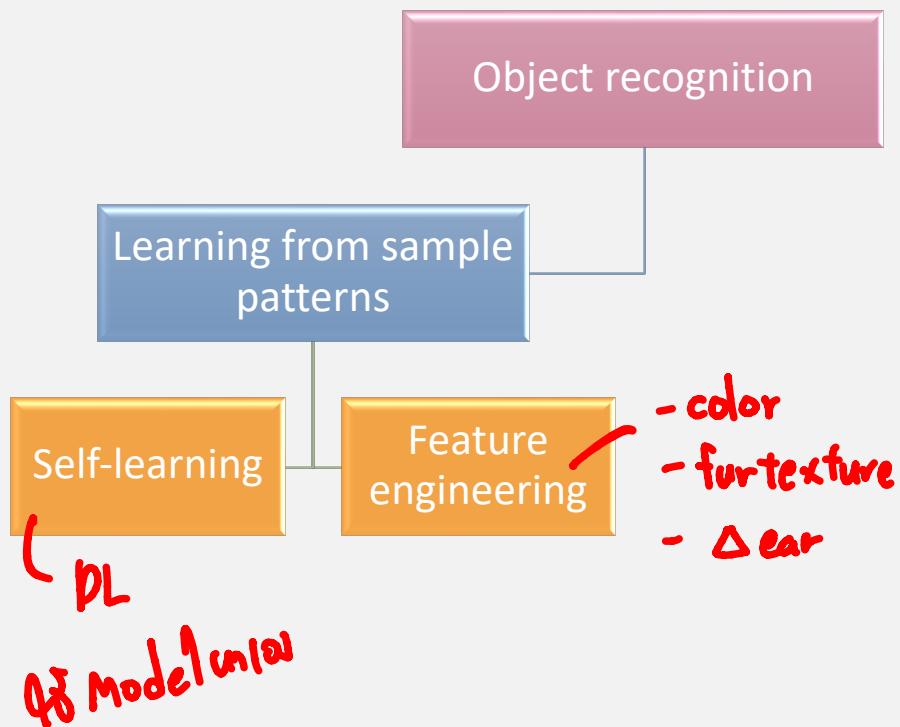


Amazon : cat hat

# OBJECT RECOGNITION FLOW



# INTRODUCTION



- Patterns – arrangements of descriptors
- Features – used in pattern recognition literature
- Pattern arrangements used in practices are vectors

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \end{bmatrix}$$

## FEATURES

Là điều gì ở pixel : Giai đáp ?

Hand Crafted

- Feature Engineering
  - process of selecting and transforming raw data into a format that is suitable for machine learning algorithms
  - making a problem easier by expressing it in a simpler way. It usually requires understanding the problem in depth.
- Colour is powerful
  - RGB colour model, HSV, CIELAB, YCbCr
- Texture
  - Co-occurrence matrix
  - Tamura's textural features
- Shape
  - Region properties
- Edge detection (human-perception)
  - First / second derivatives – sobel, prewitt, canny

Raw data:  
pixel grid



How to read the time?

# FEATURES

- Feature Engineering

- making a problem easier by expressing it in a simpler way. It usually requires understanding the problem in depth.
- Colour is powerful
  - RGB colour model, HSV, CIELAB, YCbCr
- Texture
  - Co-occurrence matrix *Homogeneity contrast*
  - Tamura's textural features
- Shape
  - Region properties
  - Edge detection (human-perception)
    - First / second derivatives – sobel, prewitt, canny

Raw data:  
pixel grid



Better  
features:  
clock hands'  
coordinates

{x1: 0.7,  
y1: 0.7}  
{x2: 0.5,  
y2: 0.0}

{x1: 0.0,  
y1: 1.0}  
{x2: -0.38,  
y2: 0.32}

Even better  
features:  
angles of  
clock hands

theta1: 45  
theta2: 0

theta1: 90  
theta2: 140

*At Hough*

*With 2 m 100x100  
(1 image = 100x100)*

## FEATURES

- Iris flowers has taxonomic problems, contains three plant species (setosa, virginica, versicolor) and four features measured for each sample

4 features

- ↔ Petal length
- ↔ Petal width
- ↔ Sepal length
- ↔ Sepal width

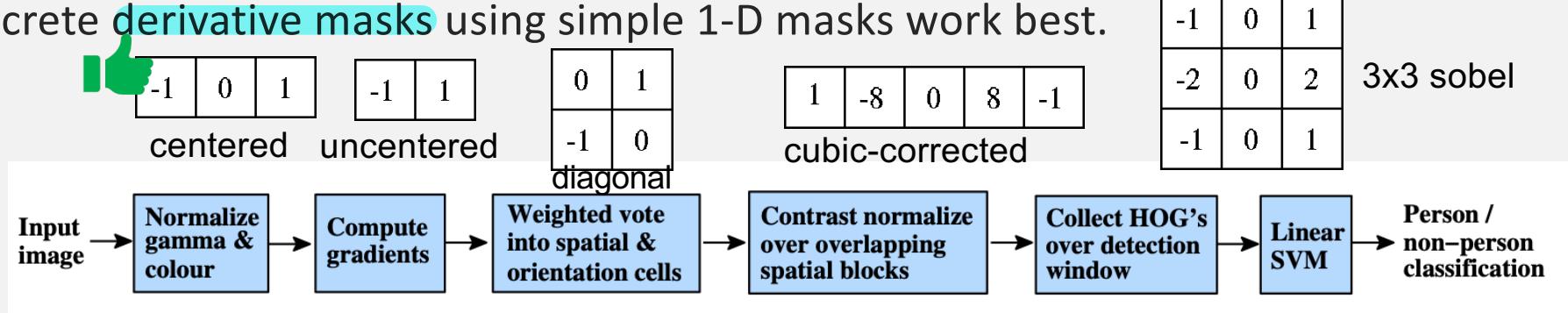


<https://rpubs.com/yao33/611068>

## FEATURES - HOG

edge

- Powerful features include histogram of oriented gradients (Dalal and Triggs, 2005) and scale-invariant feature transform (Lowe, 2004) both of which are based on gradients (edges) of the image
- HOG - (1) RGB and LAB colour spaces optionally with power law (gamma) equalization
- (2) Discrete derivative masks using simple 1-D masks work best.

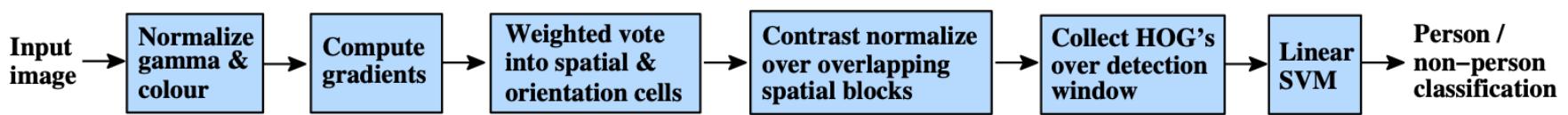
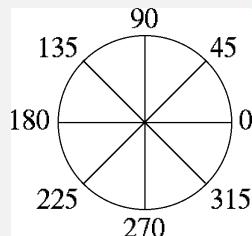


Overview of HOG feature extraction for person / non-person classification (Dalal and Triggs, 2005)

Dalal and Triggs, 2005, Histograms of Oriented Gradients for Human Detection (CVPR'05)

D.G. Lowe, 2004, Distinctive Image Features from Scale-Invariant Keypoints (IJCV'04)

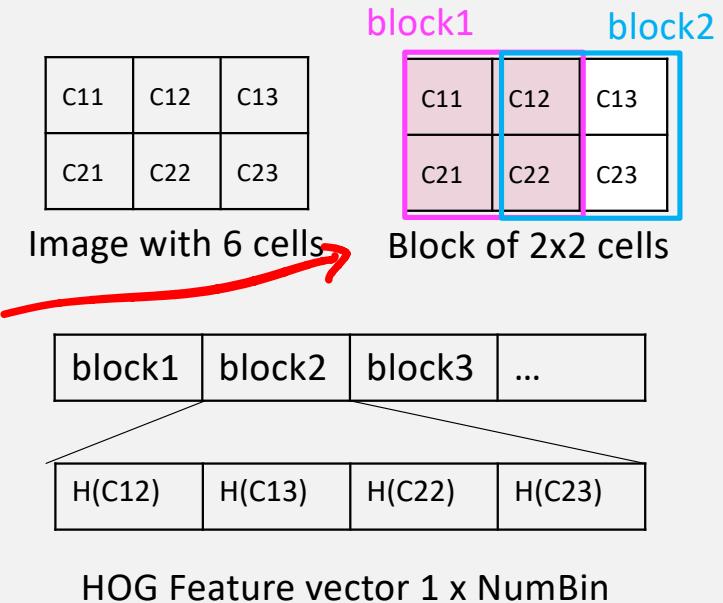
# FEATURES - HOG



Overview of HOG feature extraction for person / non-person classification (Dalal and Triggs, 2005)

- (3) Spatial and orientation binning

- An image is partitioned into non-overlapping cells.
- Each cell contains pixels which will be calculated using a weighted vote for an edge orientation histogram channel based on the orientation
- Then construct blocks of cells, e.g., block size of 2x2 cells to create each HOG block.
- Arrangement of Histograms in HOG Feature Vectors.
- The HOG feature vector is arranged by HOG blocks. The cell histogram,  $H(C_{xy})$ , is 1-by-NumBins.



$$\tan^{-1}\left(\frac{g_y}{g_x}\right)$$

## FEATURES - HOG

Weighted vote for an edge orientation histogram

Gradient Direction

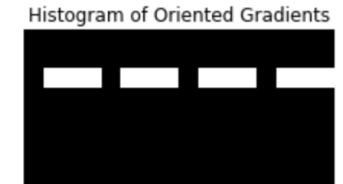
80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	59	102	163	152	176
76	13	1	68	159	22	125	143
120	70	14	150	115	144	145	140
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Magnitude

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	51	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	36	108	27	48	110

$$\sqrt{g_x^2 + g_y^2}$$

8 x 16 pixels



5 bins, 4 x 4 pixels per cell, 2 x 2 cells per block

What is the size of the feature descriptor?

```
[0. 0. 0. 0.70710678 0. 0. 0. 0. 0.70710678 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0.70710678 0. 0. 0. 0.
0.70710678 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.70710678 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

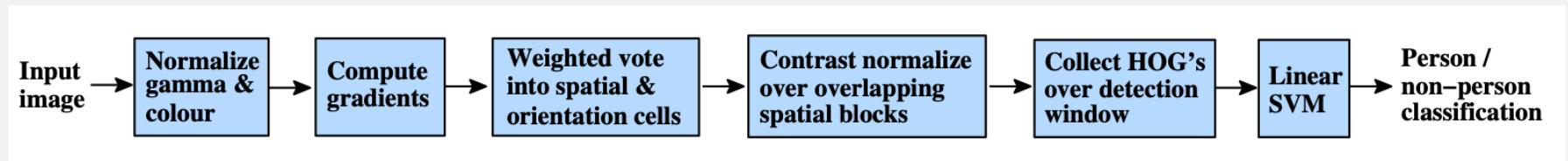
Feature descriptor = \_\_\_\_\_

$\text{num feature} = \# \text{block} \times \# \text{blocksize} \times \# \text{bin}$

= 3 blocks x 4 cells x 5

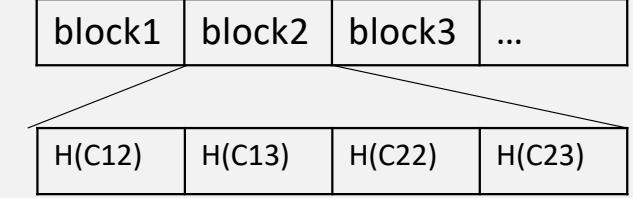
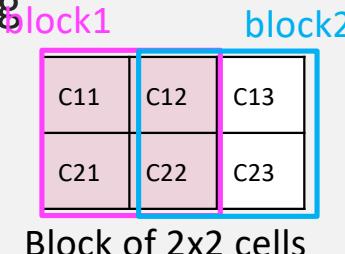
= 60 feature

## FEATURES - HOG



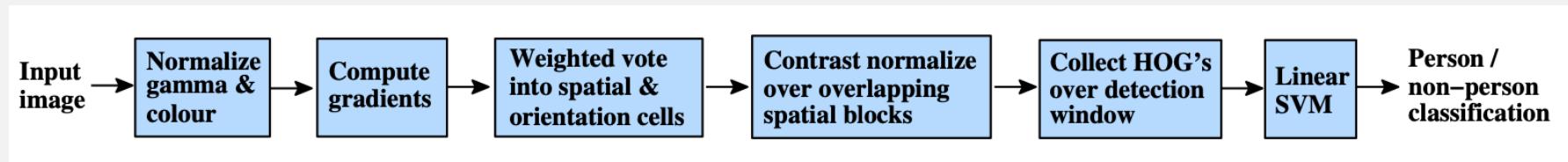
Overview of HOG feature extraction for person / non-person classification (Dalal and Triggs, 2005)

- (4) Contrast normalize over the overlapping blocks
- (5) Collect HOG's over detection window
- (6) Classifier using Linear SVM

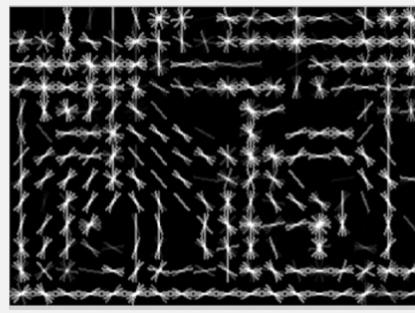


HOG's feature vector over a window

## FEATURES - HOG



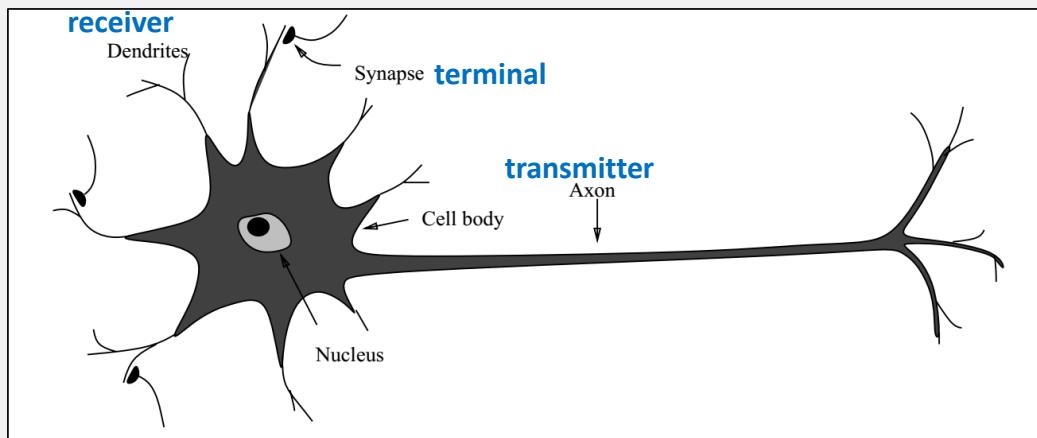
Overview of HOG feature extraction for person / non-person classification (Dalal and Triggs, 2005)



Dalal and Triggs, 2005, Histograms of Oriented Gradients for Human Detection (CVPR'05)

# NEURAL NETWORKS

- A basic component in neural networks is a neuron, which imitates the biological neuron in the brain.
- A neuron (or nerve cell) is a special biological cell with information processing ability.



Brain neuron

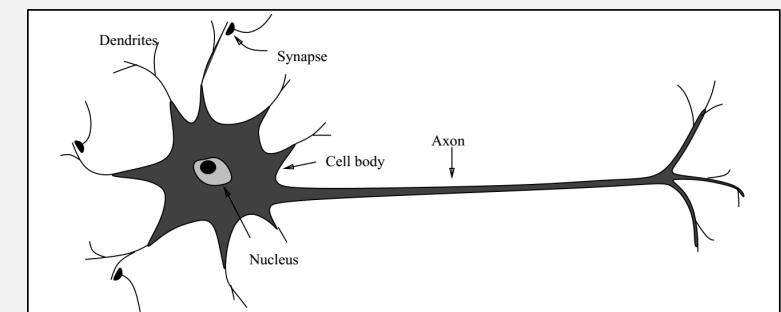
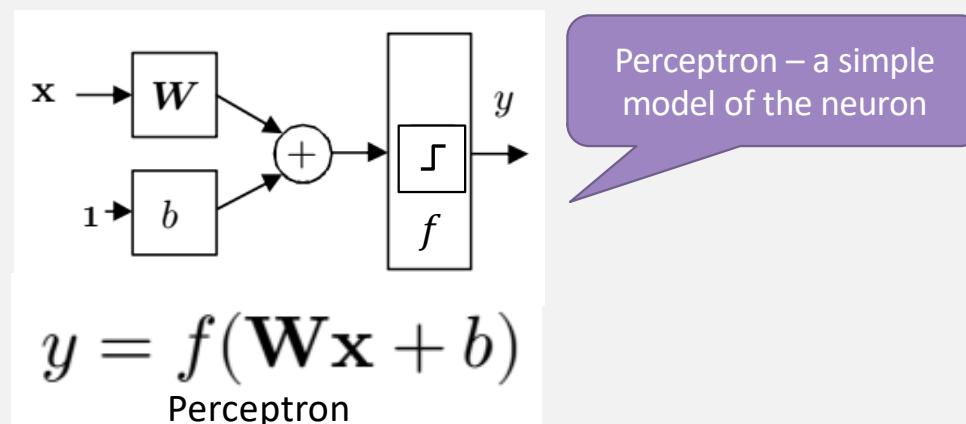
The effectiveness of a synapse can be adjusted by the signals passing through it so that the synapses can learn from the activities in which they participate.

“human ability to remember”

# NEURAL NETWORKS

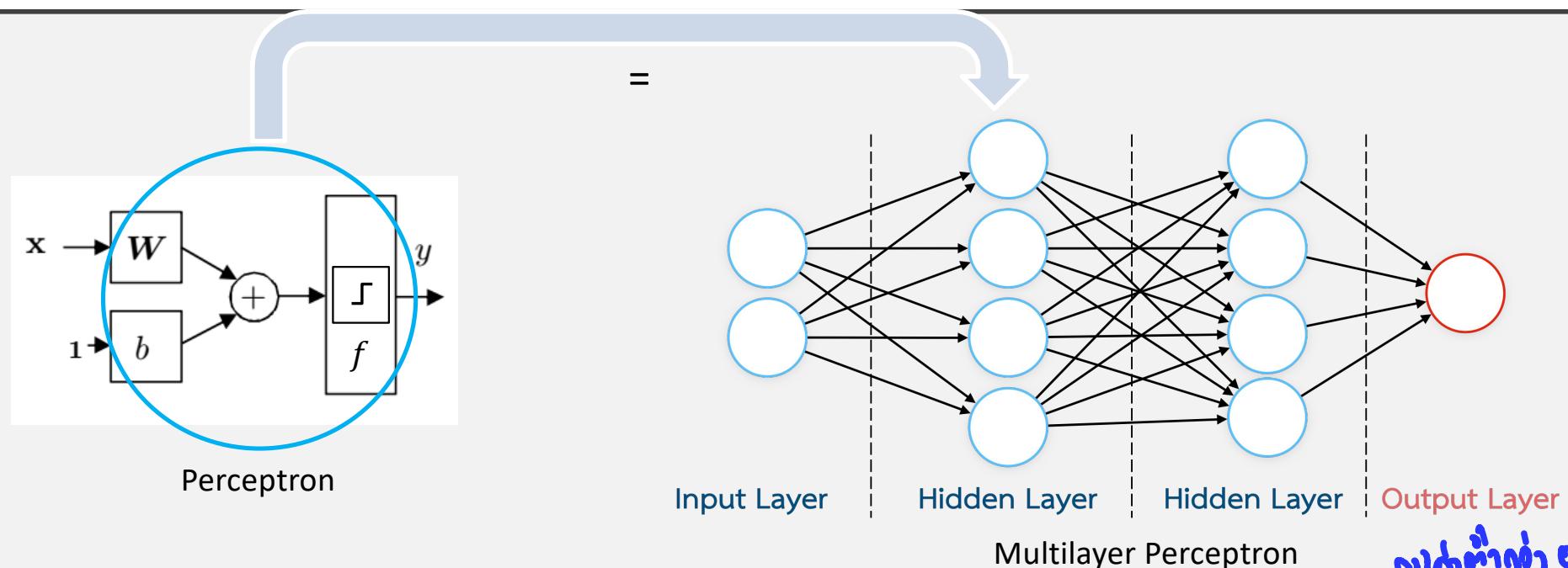
- Perceptron – learns a linear decision function with respect to the components of the pattern vectors.
- An input vector ( $x$ ) weighted by a weight matrix ( $W$ ) and the output ( $y$ ) is the function ( $f$ ) of the sum weighted input and the bias ( $b$ )
- The function ( $f$ ) initially used a binary threshold unit as a computational model for a neuron. Function  $f$  is called activation function.

$$\frac{wx+b}{\text{weight bias}} = y$$

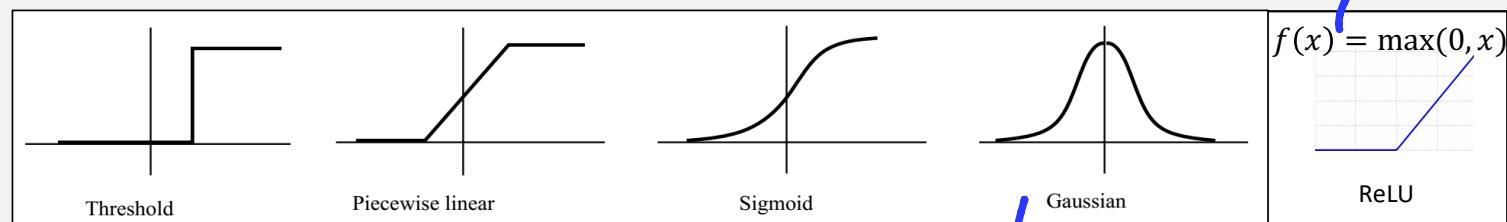


Brain neuron

# NEURAL NETWORKS



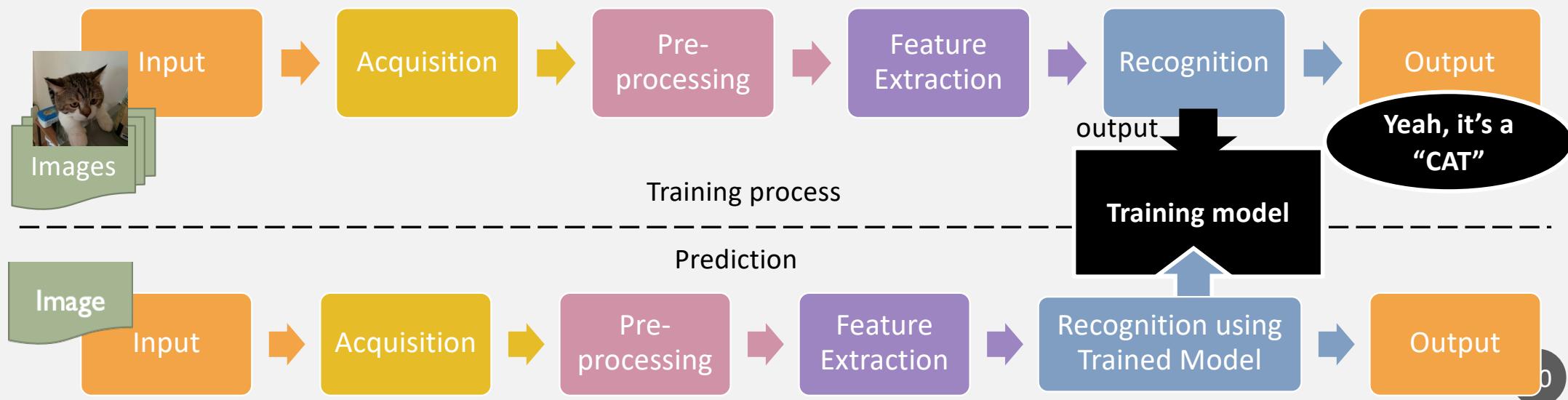
To increase generalization is to use **activation functions** other than the threshold function, e.g., a piecewise linear, sigmoid, or Gaussian



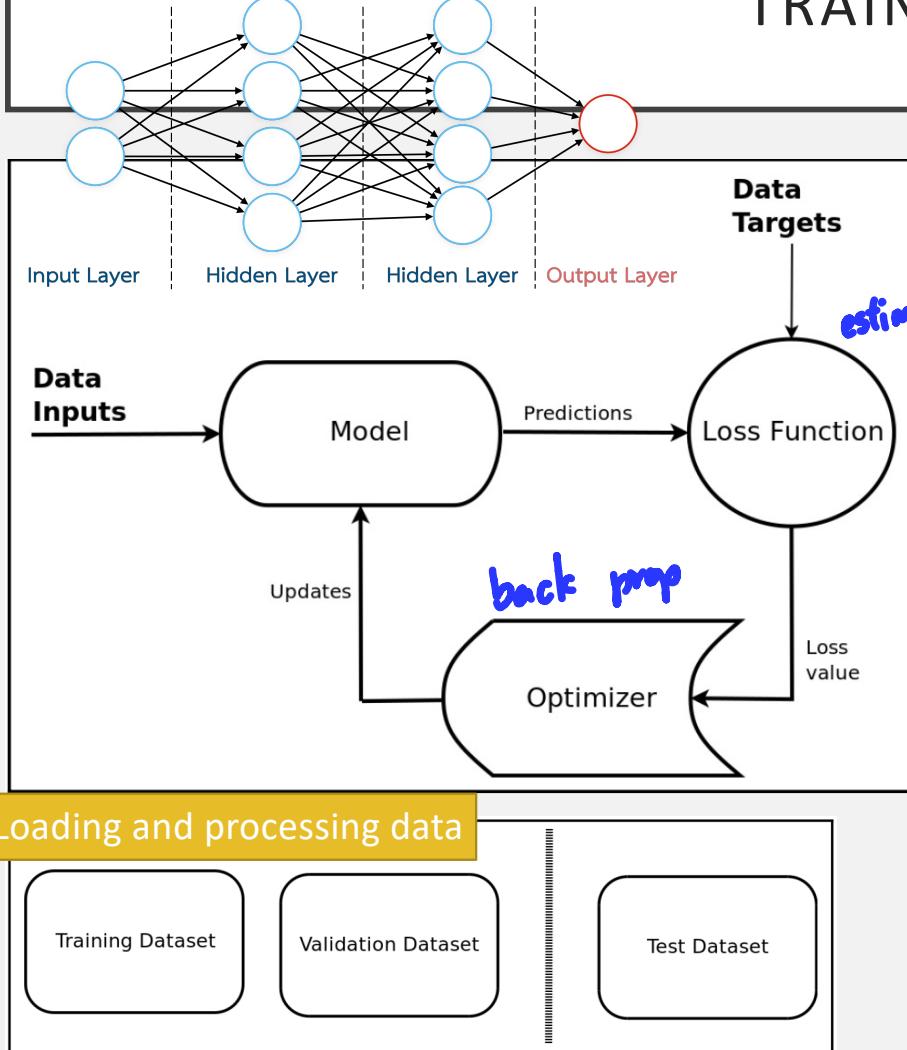
detect long distance, far

# TRAINING PROCESS

- The patterns used to estimate these parameters usually are called “training patterns”, and a set of such patterns from each class is called a “traning set”. The process is used to obtain decision function is called “learning” or “training”.



# TRAINING PROCESS



Training process is an iterative process. Processing data – **Inputs and targets** are divided into three main subsets:

## 1. Training dataset

60-80% of the whole data

- the model learns patterns, relationships, and features from this data to make predictions or classifications.
- Used to adjust its internal parameters based on these examples to minimize the error or loss function.

## 2. Validation dataset

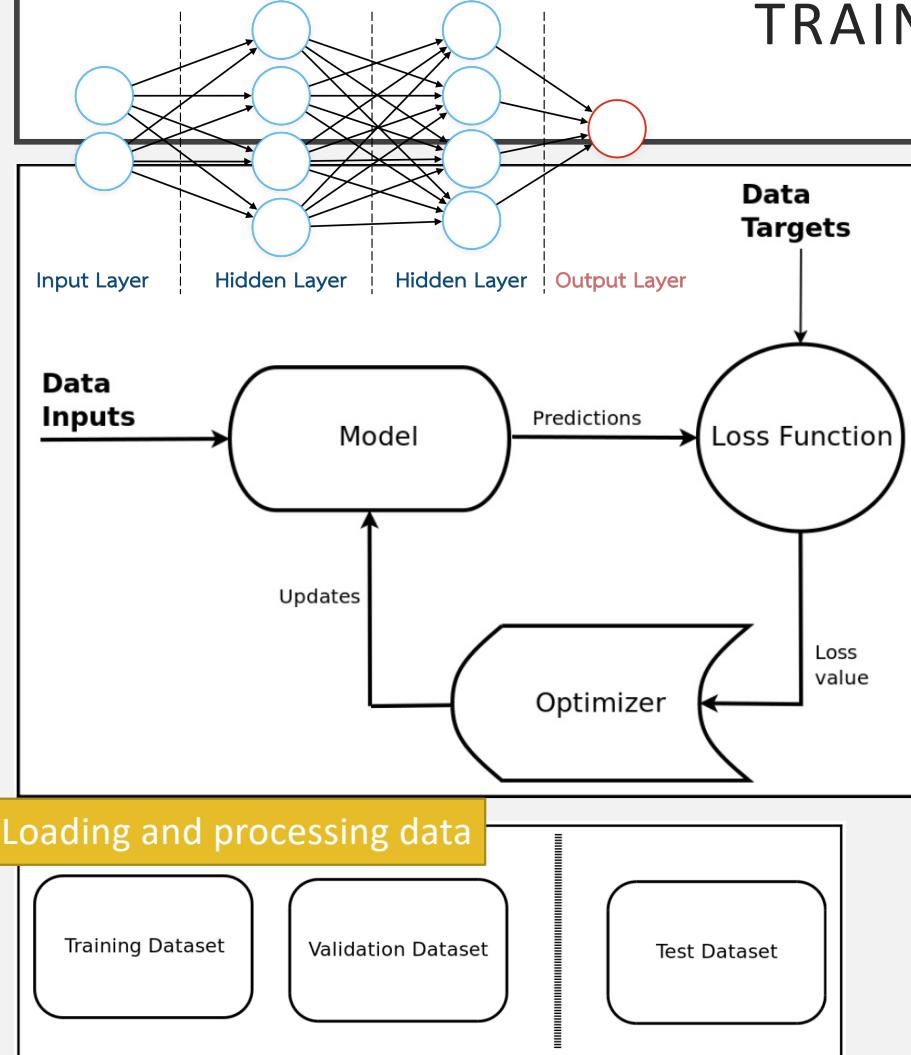
*external params ; epoch, lr . .*

- 10-20% of the whole data
- used to fine-tune model hyperparameters and monitor its performance during training
- helps in selecting the best hyperparameters and deciding when to stop training

## 3. Testing dataset : *model* *on* *Fully Unseen*

- simulating how the model will perform in the real world
- The model's performance metrics, such as accuracy, precision, recall, or F1-score, are computed to assess its generalization ability.
- should be large enough to provide statistically significant results but should not overlap with the training or validation datasets

# TRAINING PROCESS



**Loss function** - known as a cost function or objective function

- to measure how well or poorly a model is performing, with the goal of minimizing this function during the training process.
- to serve as a guide for optimization algorithms
- These optimized parameters make the model's predictions as close as possible to the true target values

## Examples of Loss Functions

- Mean Squared Error (MSE)
- Binary Cross-Entropy
- Categorical Cross-Entropy

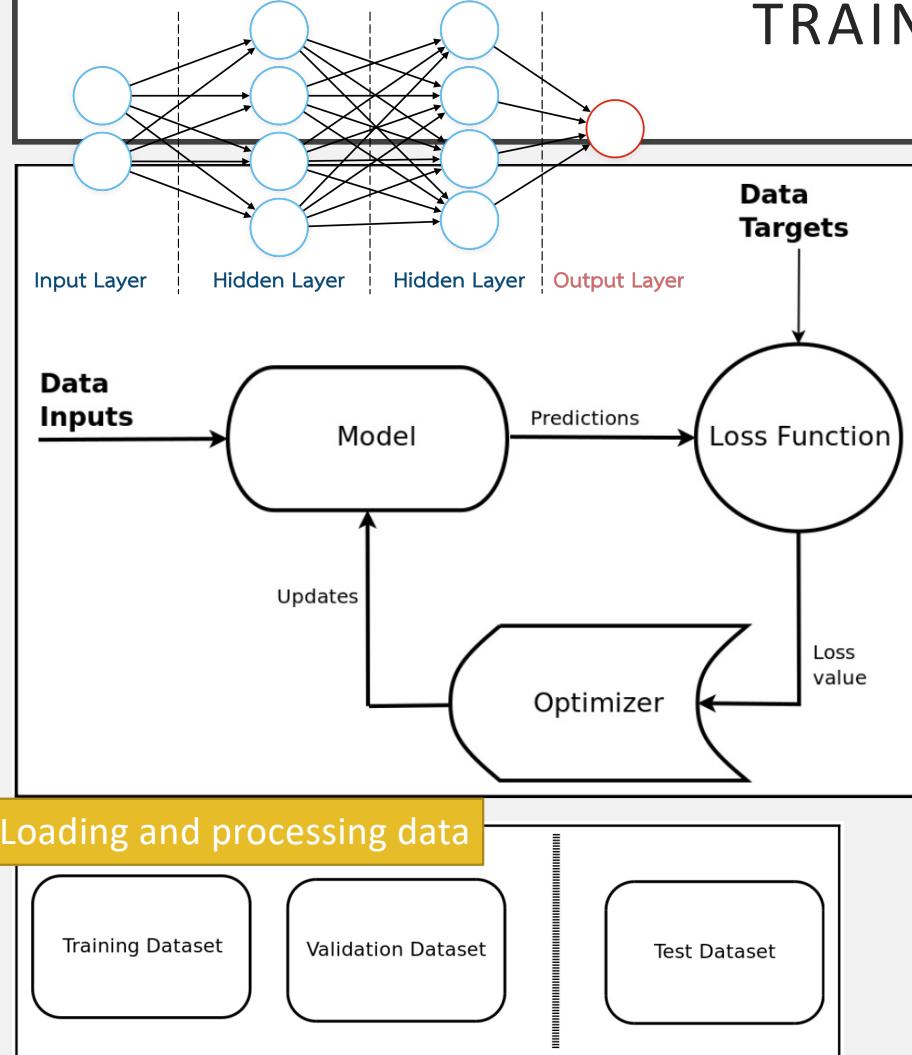
$$\text{loss} = \frac{1}{n} \sum_{i=1}^n (y_{true} - y_{pred})^2$$

$y_i = \text{real}$     $\text{loss} = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i)$  : **Regression**, **Classification**  
 $y_i$  is class indicator and  $p_i$  is probability of the sample  $i$ .

$$\text{loss} = \sum_{c=1}^C -y_{i,c} \log(p_{i,c})$$

$y_{i,c}$  is class indicator and  $p_{i,c}$  is the probability of the sample  $i$  in class  $c$

# TRAINING PROCESS



**Loss function** - known as a cost function or objective function

- to measure how well or poorly a model is performing, with the goal of minimizing this function during the training process.
- to serve as a guide for optimization algorithms
- These optimized parameters make the model's predictions as close as possible to the true target values

**Examples of Loss Functions**

- Mean Squared Error (MSE)
- Binary Cross-Entropy

$$\text{loss} = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i)$$

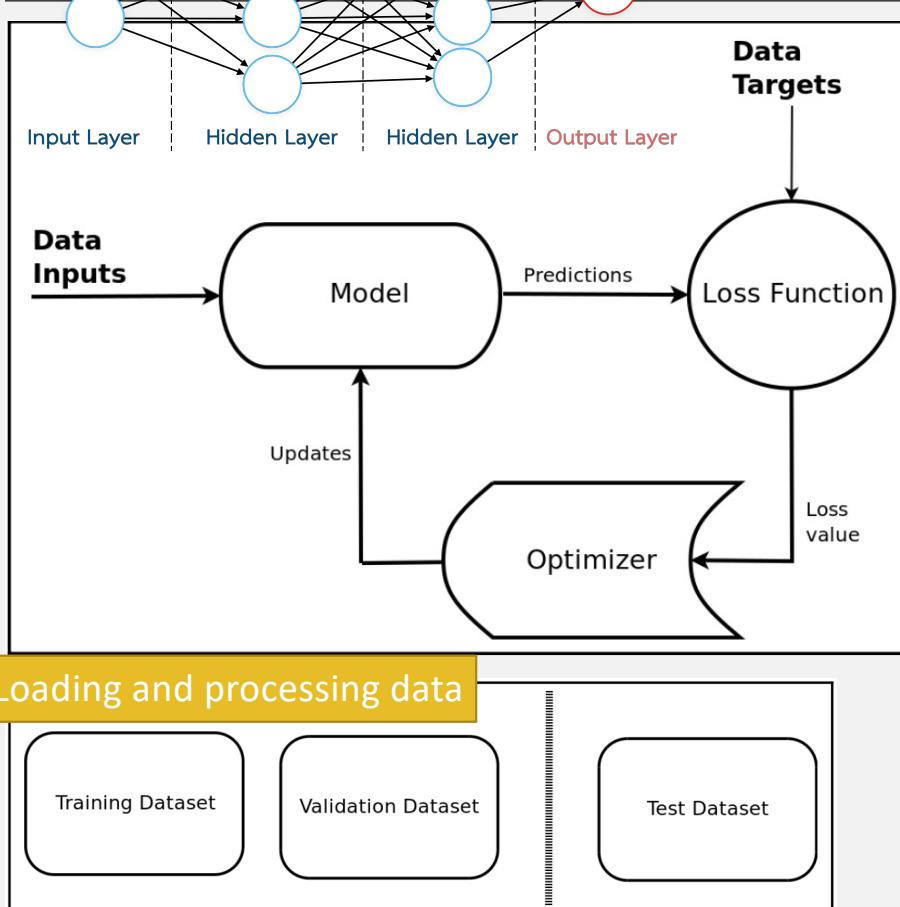
$y_i$  is class indicator and  $p_i$  is probability of the sample  $i$ .

- Categorical Cross-Entropy

$$\text{loss} = \sum_{c=1}^M -y_{i,c} \log(p_{i,c})$$

$y_{i,c}$  is class indicator and  $p_{i,c}$  is the probability of the sample  $i$  in class  $c$

# TRAINING PROCESS



**Optimizer** - an algorithm that adjusts the weights and biases of the network to minimize the loss function

- aims to find the **minimum** of the loss function. By doing so, it improves the model's predictions using **gradient** of the loss function

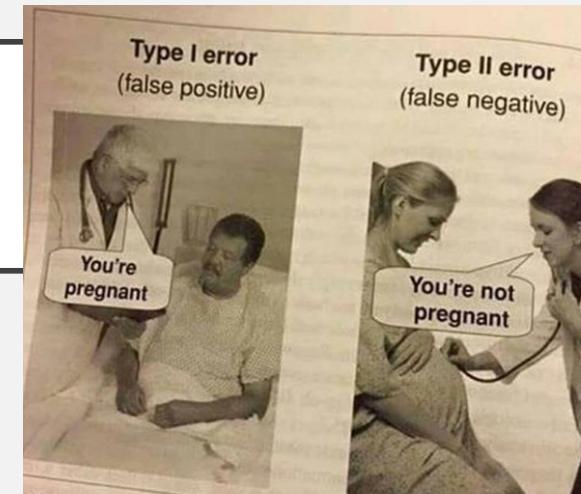
**Examples of Optimizers:**

- Gradient Descent - GD uses the entire dataset to calculate the gradient of the cost function for each iteration
- Stochastic Gradient Descent (SGD) - updates weights by considering one data point *per 1 epoch* /batch
- Momentum - considers the previous gradients to keep the momentum
- RMSprop - adapts the learning rates during training
- Adam - adjusts the learning rates which is the size of the steps that the optimizer takes while updating the weights of each parameter, benefiting from adaptive learning rates.

# PERFORMANCE EVALUATION (IMAGE CLASSIFICATION)

- Performance evaluation:

- counts of test records **correctly** and **incorrectly** predicted by the model using “**confusion matrix**”



Confusion matrix for a 2-class problem.

		Predicted Class	
		<i>Class = 1</i>	<i>Class = 0</i>
Actual Class	<i>Class = 1</i>	$f_{11}$ <b>TP</b>	$f_{10}$ <b>FN</b>
	<i>Class = 0</i>	$f_{01}$ <b>FP</b>	$f_{00}$ <b>TN</b>

True positive (TP)  
True negative (TN)  
False positive (FP) – false alarm – Type I  
False negative (FN) – Type II

T : ຖັນ, P : ໄມວ, F : ໄປວາ  
T : ຖັນ, P : ໄມວ, F : ໄປວາ  
PIM  
ບັນຫຼືນ

- the total number of correct predictions made by the model is  $(f_{11} + f_{00}) = \text{TP+TN}$
- the total number of incorrect predictions is  $(f_{10} + f_{01}) = \text{FN+FP}$

# PERFORMANCE EVALUATION (IMAGE CLASSIFICATION)

- Accuracy

$$\text{Accuracy} = \frac{\# \text{ of correct predictions}}{\text{total } \# \text{ of predictions}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}} = \frac{\textcolor{violet}{TP+TN}}{\textcolor{violet}{ALL}}$$

- Precision

$$\text{Precision} = \frac{\textcolor{violet}{TP}}{\textcolor{violet}{TP+FP}}$$

- Recall

$$\text{Recall} = \frac{\textcolor{teal}{TP}}{\textcolor{teal}{TP+FN}}$$

- F-measure

$$\text{F-score} = 2 \cdot \frac{\textcolor{magenta}{precision \cdot recall}}{\textcolor{magenta}{precision + recall}}$$

- Others: ROC curve, PR curve

Confusion matrix for a 2-class problem.

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	$f_{11}$ <b>TP</b>	$f_{10}$ <b>FN</b>
	Class = 0	$f_{01}$ <b>FP</b>	$f_{00}$ <b>TN</b>

## CASE STUDY: MNIST DATASET

- **MNIST** - a database of handwritten digits made up of a training set of 60,000 examples and a test set of 10,000 examples.
  - Apply HOG features and use neural networks for classification and visualization
- Load data

```
import numpy as np
from skimage.feature import hog
from sklearn import preprocessing

(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)

## CASE STUDY: MNIST DATASET

- Data Normalization

ມະນະຂໍ້ຕົວລັບ

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255 # 0 - 1
X_test /= 255
```

- Feature Extraction

```
list_hog = []
for feature in X_train:
    fd = hog(feature.reshape((28,28)), orientations=9,
              pixels_per_cell=(14,14), cells_per_block=(1,1))
    list_hog.append(fd)

hog_features_train = np.array(list_hog, 'float64')
```

# CASE STUDY: MNIST DATASET

- Feature normalization

Scale each feature by its maximum absolute value. The maximal absolute value of each feature in the training set will be 1.0

```
preProcess = preprocessing.MaxAbsScaler().fit(hog_features_train)
hog_features_transformed_train = preProcess.transform(hog_features_train)
```

- Training model

Currently, MLPClassifier supports only the Cross-Entropy loss function for **classification** task.

```
model_hog_mlp = MLPClassifier(activation='relu', hidden_layer_sizes=(100, 100),
verbose=1,max_iter=20)
model_hog_mlp.fit(hog_features_transformed_train, y_train)
print("Training accuracy ::\n{}".format(model_hog_mlp.score(hog_features_transformed_train, y_train)))
```

[https://scikit-learn.org/stable/modules/neural\\_networks\\_supervised.html](https://scikit-learn.org/stable/modules/neural_networks_supervised.html)

[https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

# CASE STUDY: MNIST DATASET

- Testing

```
list_hog_test = []
for feature in X_test:
    fd = hog(feature.reshape((28,28)), orientations=9, pixels_per_cell=(14,14),cells_per_block=(1,1) )
    list_hog_test.append(fd)
hog_features_test = np.array(list_hog_test, 'float64')

hog_features_transformed_test = preprocess.transform(hog_features_test)

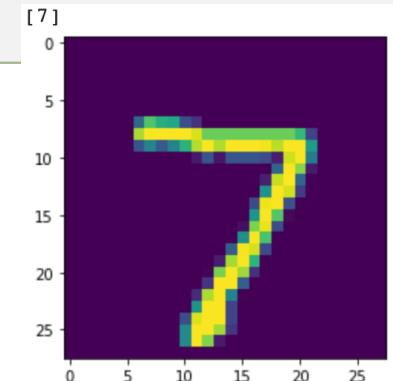
print("Testing Score :: {}".format(model_hog_mlp.score(hog_features_transformed_test, y_test)))
```

Testing Score :: 0.8922

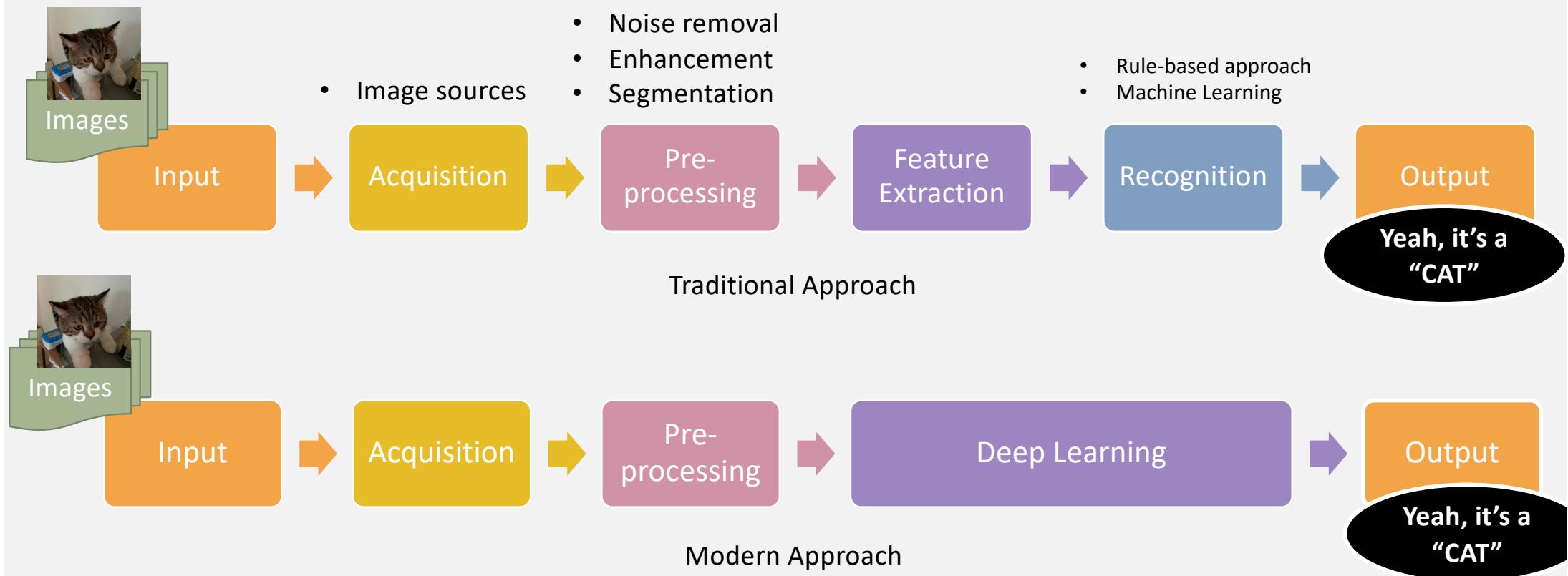
- Deployment

```
fd_test = hog(X_test[0].reshape((28,28)), orientations=9,
pixels_per_cell=(14,14),cells_per_block=(1,1))
hog_features_transformed_test =
    preprocess.transform(fd_test.reshape((1,hog_features_test.shape[1])))
print(model_hog_mlp.predict(hog_features_transformed_test))

plt.imshow(X_test[0])
plt.show()
```

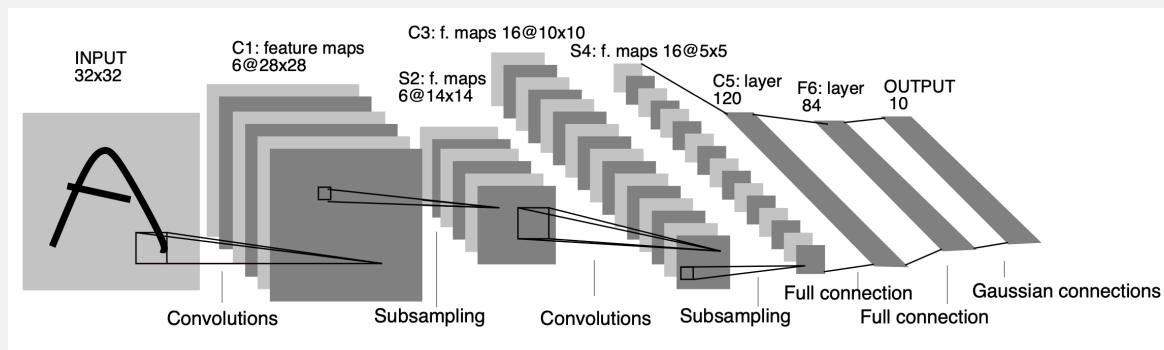


# OBJECT RECOGNITION FLOW



# CONVOLUTIONAL NEURAL NETWORKS

- Deep learning uses neural networks to learn useful representations of features directly from data.
- The state-of-the-art **convolutional neural networks (CNNs or ConvNets)** is used for classification and regression. A Convolutional Neural Network (CNN, or ConvNet) designed to recognize visual patterns directly from pixel images with minimal preprocessing.



Architecture of LENET-5

Y. Lecun, et al, 1998, Gradient-based learning applied to document recognition

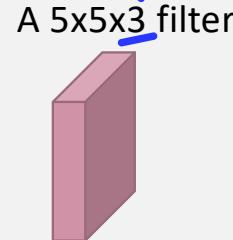
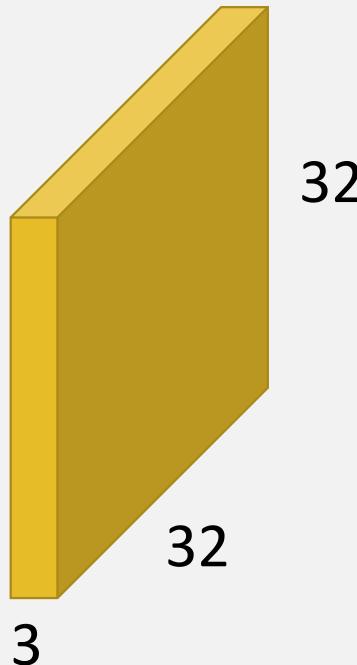
Three main layers

- Convolution Layer
- Pooling **down sampling**
- Fully Connected **NN**

# CONVOLUTIONAL NEURAL NETWORKS

- Convolution layer

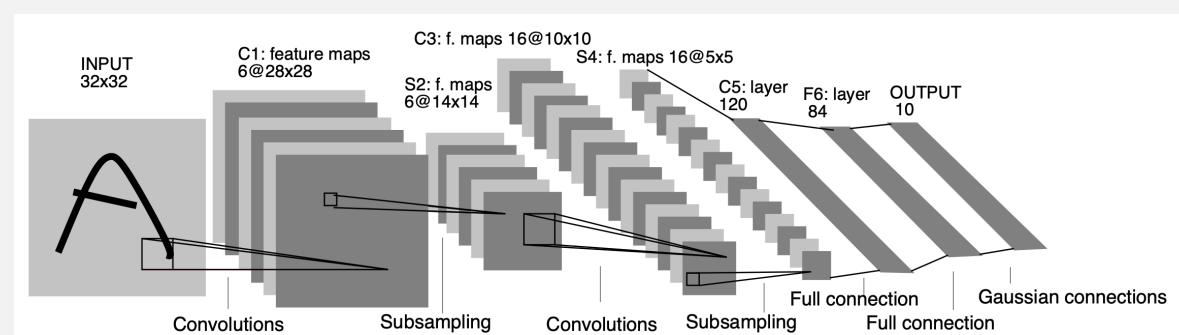
A  $32 \times 32 \times 3$  image



Input: Input in RGB

A  $5 \times 5 \times 3$  filter

**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

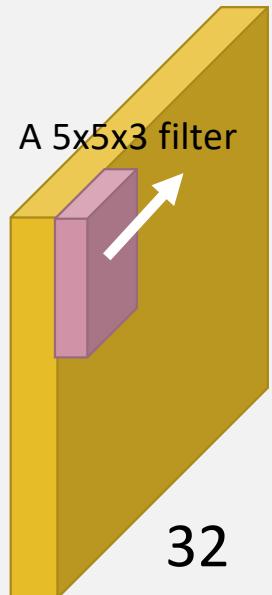


Architecture of LENET-5

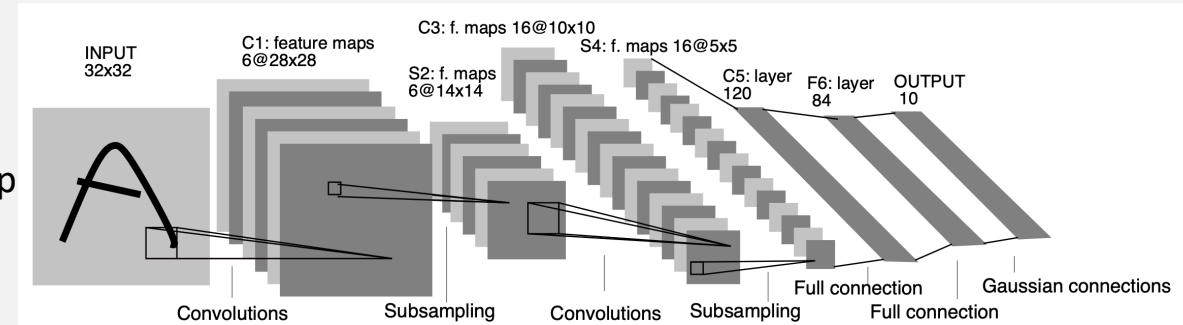
# CONVOLUTIONAL NEURAL NETWORKS

- Convolution layer

A 32x32x3 image



An output  
=Activation map



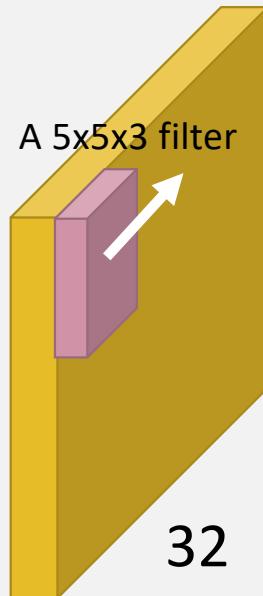
shift kernel 1 pixel: 192000  
Stride = 1, no padding  
192000

1 Node depth how

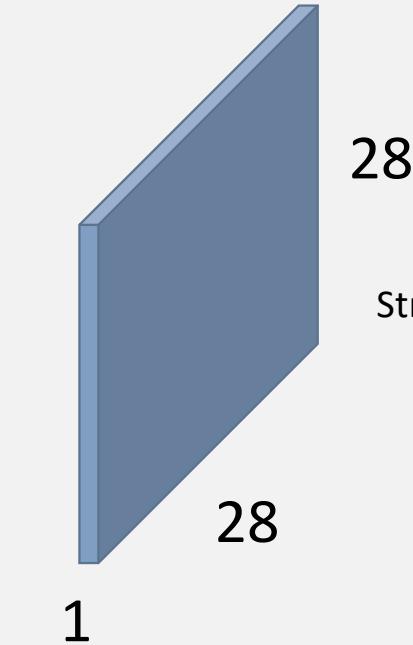
# CONVOLUTIONAL NEURAL NETWORKS

- Convolution layer

A  $32 \times 32 \times 3$  image

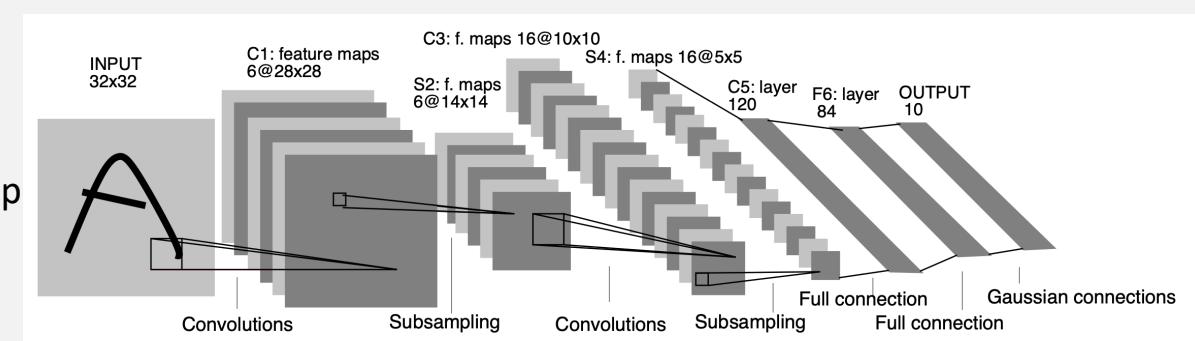


An output  
=Activation map

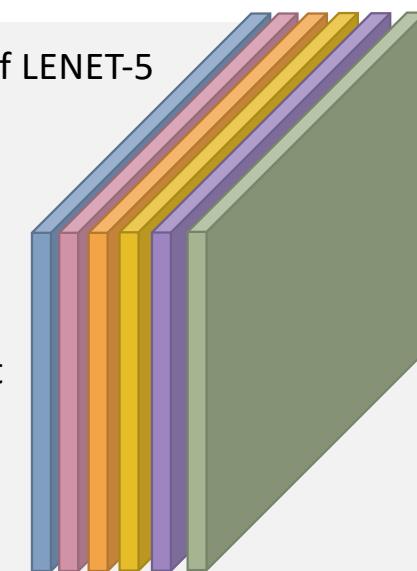


Stride = 1, no padding

For 6  $5 \times 5$  filters, we get  
the activation maps  
 $6 \times 28 \times 28$

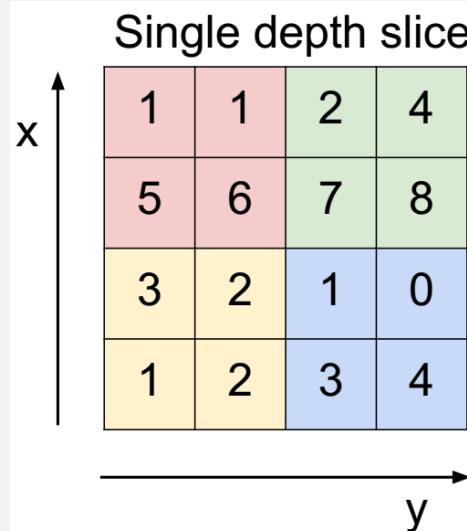
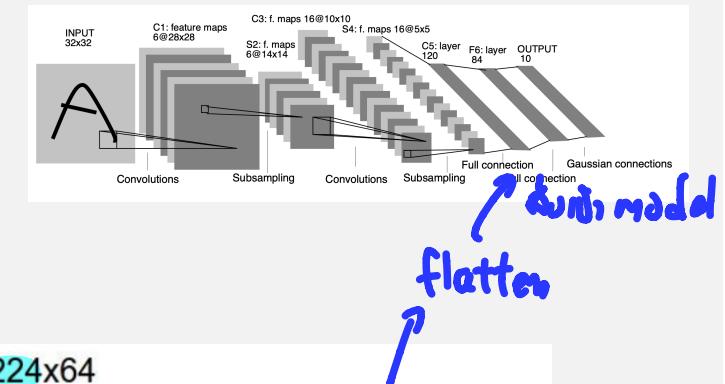


Architecture of LENET-5

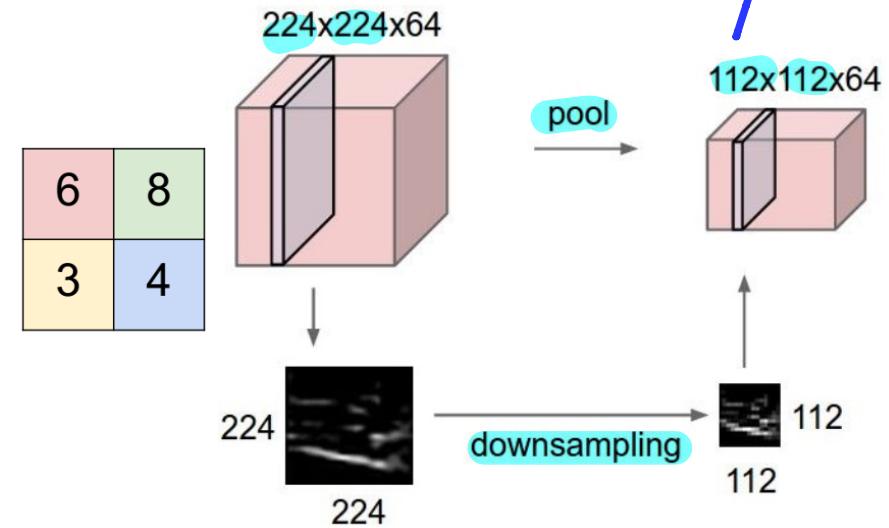


# CONVOLUTIONAL NEURAL NETWORKS

- Pooling layer
  - makes the representations smaller and more manageable
  - operates over each activation map independently

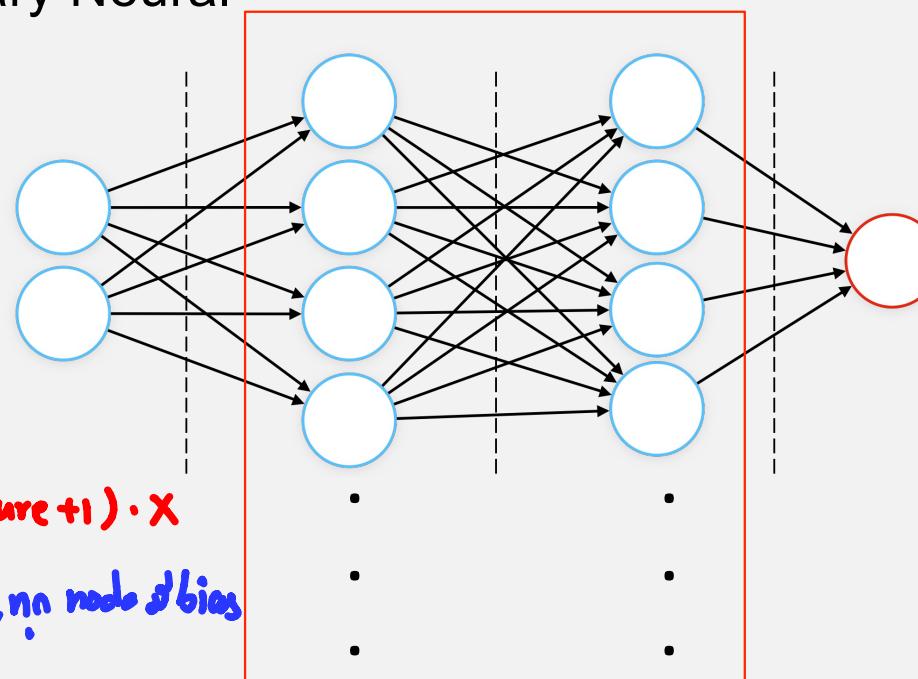
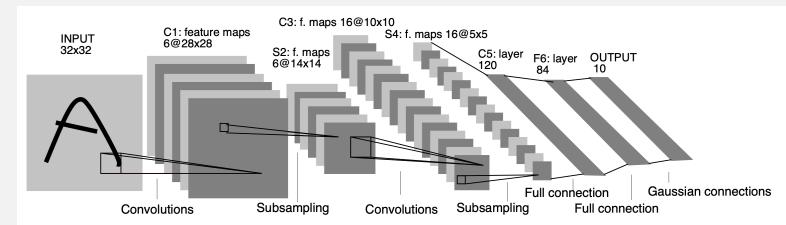


max pool with 2x2 filters  
and stride 2



# CONVOLUTIONAL NEURAL NETWORKS

- Fully Connection layer / Dense / Linear
  - Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



# CONVOLUTIONAL NEURAL NETWORKS

## Key parameters / functions

- **Losses functions** or objective functions, or optimization score function
  - Class loss - used to calculate the cross-entropy for classification problems. Multiple versions include binary cross-entropy,  $loss = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i)$ ;  $y_i$  is class indicator and  $p_i$  is probability of sample  $i$ .
  - categorical cross-entropy,  $loss = \sum_{c=1}^M -y_{i,c} \log(p_{i,c})$ ;  $y_{i,c}$  is class indicator and  $p_{i,c}$  is the probability of sample  $i$  in class  $c$
- A **metric function** is the results from evaluating a metric are not used when training the model, e.g., ‘accuracy’
- An **epoch** is one complete pass through the whole training set
- **Batch size** is the number of training instances observed before optimization performs update *ுননিত স্বীকৃত ওজন অপারেশন*

Reference: <https://pytorch.org/tutorials/>

# DEEP LEARNING FRAMEWORK

Everyone's situation and needs are different, so it boils down to which features matter the most for your AI project. For easy reference, here's a chart that breaks down the features of Keras vs PyTorch vs TensorFlow.

	Keras	PyTorch	TensorFlow
API Level	High	Low	High and Low
Architecture	Simple, concise, readable	Complex, less readable	Not easy to use
Datasets	Smaller datasets	Large datasets, high performance	Large datasets, high performance
Debugging	Simple network, so debugging is not often needed	Good debugging capabilities	Difficult to conduct debugging
Does It Have Trained Models?	Yes	Yes	Yes

	Keras	PyTorch	TensorFlow
Popularity	Most popular	Third most popular	Second most popular
Speed	Slow, low performance	Fast, high-performance	Fast, high-performance
Written In	Python	Lua	C++, CUDA, Python

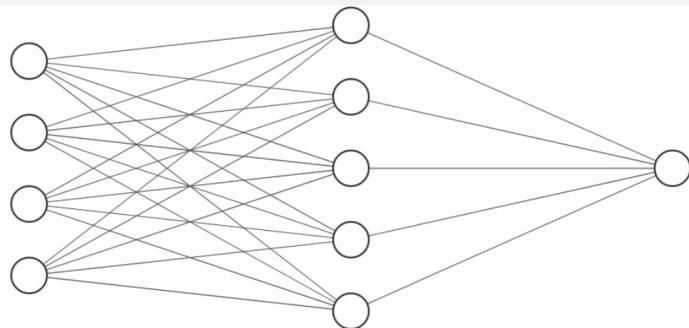
<https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article>

## **TRADITIONAL VS. MODERN CASE STUDY: MNIST DATASET**

- Apply a simple convolutional neural networks and compared the results with previous approach

# CREATING A MODEL USING PYTORCH

- Implementing a sequential model
  - use the nn.Sequential() package to create a deep learning model by passing layers in order



```
from torch import nn
# define a two-layer model
model = nn.Sequential(
    nn.Linear(4, 5),
    nn.ReLU(),
    nn.Linear(5, 1),
)
print(model)
```

Output:

```
Sequential(
    (0): Linear(in_features=4, out_features=5, bias=True)
    (1): ReLU()
    (2): Linear(in_features=5, out_features=1, bias=True) )
```

# CREATING A MODEL USING PYTORCH

- Defining models using `nn.Module`
  - specify the layers in the `__init__` method of the class
  - In `forward` method, we apply the layers to inputs
  - This method provides **better flexibility** for building customized models.
  - To accelerate operations in the neural network, we move it to the GPU if available.

```
Net(  
    (conv1): Sequential(  
        (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))  
        (1): ReLU()  
        (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) )  
    (conv2): Sequential(  
        (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))  
        (1): ReLU()  
        (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) )  
    (out): Linear(in_features=1568, out_features=10, bias=True)  
)
```

Output:

```
class Net(nn.Module):  
    def __init__(self):  
        super(Net, self).__init__()  
        self.conv1 = nn.Sequential(  
            nn.Conv2d(  
                in_channels=1,  
                out_channels=16,  
                kernel_size=5,  
                stride=1,  
                padding=2,  
            ),  
            nn.ReLU(),  
            nn.MaxPool2d(kernel_size=2),  
        )  
        self.conv2 = nn.Sequential(  
            nn.Conv2d(16, 32, 5, 1, 2),  
            nn.ReLU(),  
            nn.MaxPool2d(2),  
        )  
        # fully connected layer, output 10 classes  
        self.out = nn.Linear(32 * 7 * 7, 10)  
    def forward(self, x):  
        x = self.conv1(x)  
        x = self.conv2(x)  
  
        # flatten the output of conv2 to (batch_size, 32 * 7 *  
        x = x.view(x.size(0), -1)  
        output = F.log_softmax(self.out(x), dim=1)  
        return output  
model = Net().to(device)  
print(model)
```

The use of log probabilities improves numerical stability, when the probabilities are very small,

# TRAINING PROCESS

In a single training loop, the model makes predictions on the training dataset (fed to it in batches), and backpropagates the prediction error to adjust the model's parameters.

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(dataloader)
            print(f"loss: {loss:.7f} [{current:.5d}/{size:.5d}]")
```

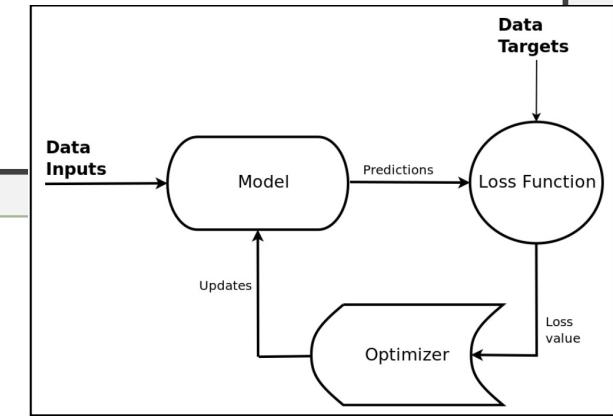
Sets the gradients of all optimized torch.Tensor s to zero.

compute the gradients with respect to the model parameters

update the model parameters

```
def val(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y.type(torch.float).sum().item())
    test_loss /= num_batches
    correct /= size
    print(f"Val Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8.5f}")

    Context-manager that disabled gradient calculation / not update internal weights
```



Loading and processing data

Training Dataset

Validation Dataset

Test Dataset

Reference: [https://pytorch.org/tutorials/beginner/basics/quickstart\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html)

# TRAINING PROCESS

```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")

epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-----")
    train(train_dataloader, model, loss_fn, optimizer)
    val(val_dataloader, model, loss_fn)
print("Done!")
```

Reference: [https://pytorch.org/tutorials/beginner/basics/quickstart\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html)

# CREATING A MODEL USING PYTORCH

- Save model (for later use) and load model

```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

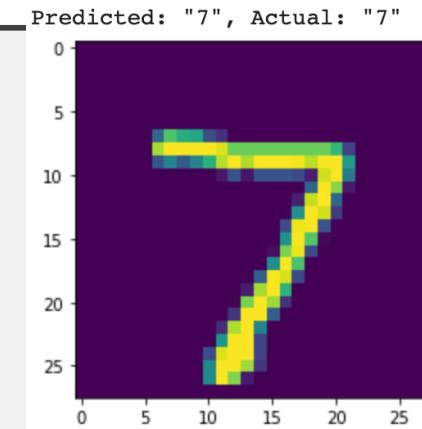
  

```
model = Net().to(device)
model.load_state_dict(torch.load("model.pth"))
```

- Evaluation

```
model.eval()
x, y = test_data[0][0], test_data[0][1]
print(x.shape)
with torch.no_grad():
    pred = model(x.unsqueeze(0).to(device))
    predicted, actual = pred[0].argmax(0), y
    print(f'Predicted: "{predicted}", Actual: "{actual}"')

plt.imshow(x[0])
plt.show()
```



`model.eval()` is a kind of switch for some specific layers/parts of the model that behave differently during training and inference (evaluating) time.

- For example, Dropouts Layers, BatchNorm Layers etc. You need to turn off them during model evaluation, and `.eval()` will do it for you.
- In addition, the common practice for evaluating/validation is using `torch.no_grad()` in pair with `model.eval()` to turn off gradients computation:

# IMPROVING MODEL GENERIZATION

- **Dropout** is a technique where you remove (or "drop out") units in a neural net to simulate training large numbers of architectures simultaneously. Importantly, dropout can drastically reduce the chance of overfitting during training.

- Number of layers / epoch /
- Number of filters
- Batch normalization

200/1 depth

```
self.conv2 = nn.Sequential(  
    nn.Conv2d(16, 32, 5, 1, 2),  
    nn.BatchNorm2d(32),  
    nn.Dropout(0.25),  
    nn.ReLU(),  
    nn.MaxPool2d(2),  
)
```

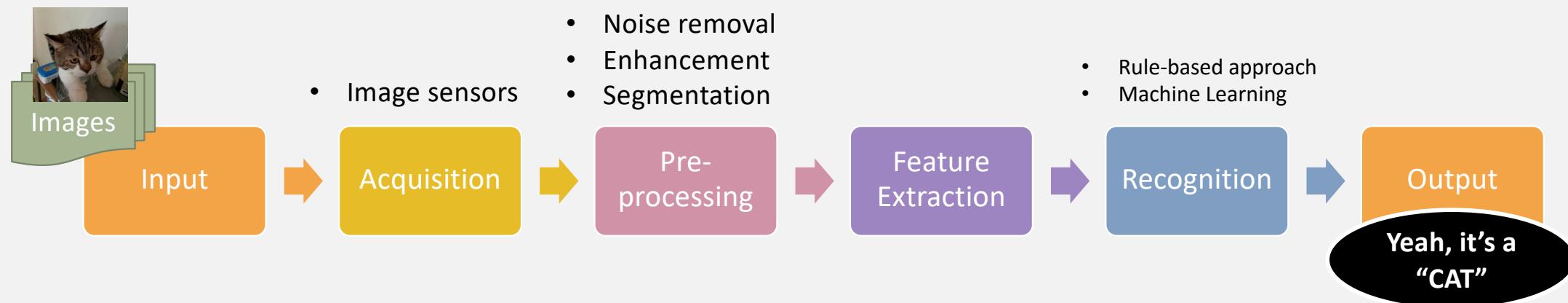
- Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension)
- The mean and standard-deviation are calculated per-dimension over the mini-batches and  $\gamma$  and  $\beta$  are learnable parameter vectors of size  $C$  (where  $C$  is the input size)

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

math  $M=0$   
 $\text{Var} \approx 1$

shifting  
scale params  
params

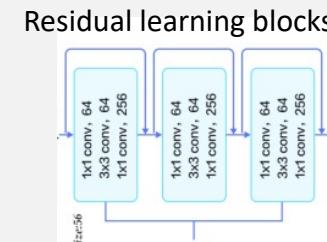
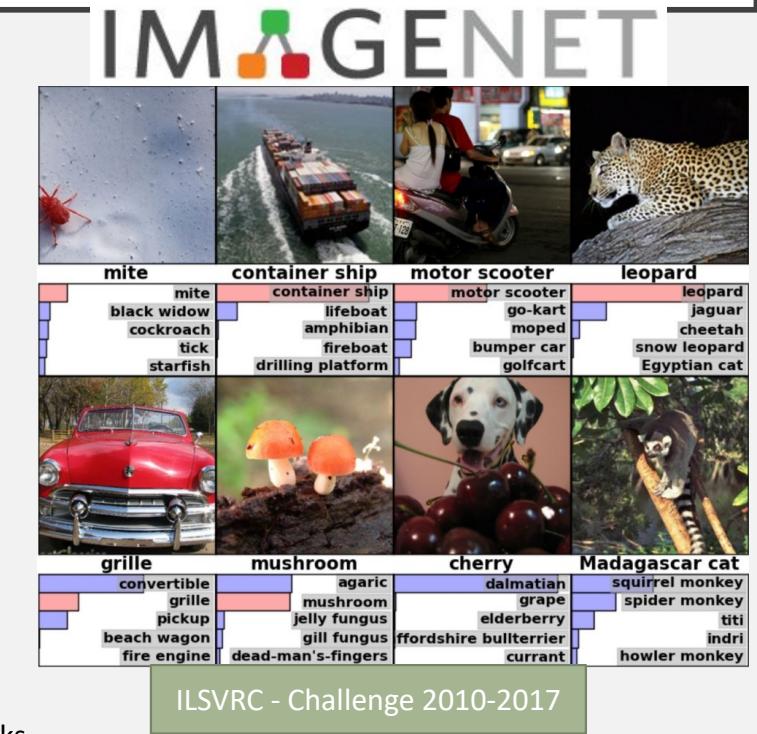
# RECOGNITION PERFORMANCE



- Specific to problems
- Feature Selection / Self learning features
- Dataset Quality and Quantity
  - Data augmentation
- Noise removal
- Enhancement
- Segmentation
- Rule-based approach
- Machine Learning
- Error analysis + Literature review
- Improve current model (parameters, and architecture) or use alternative models

# IMAGE CLASSIFICATION ADVANCEMENT

- ImageNet project is a large visual database
  - 1,000 object classes (categories).
  - > 1M train labeled high resolution images
- Deeper, wider or more flexible layers, introducing dropout, data augmentation, GPU parallel computation
  - AlexNet, ZFNet, GoogLeNet and VGG were the winner and runner-up of the ILSVRC-2014, ResNet,
  - EfficientNet (Scaling CNNs)
- Transfer learning concept

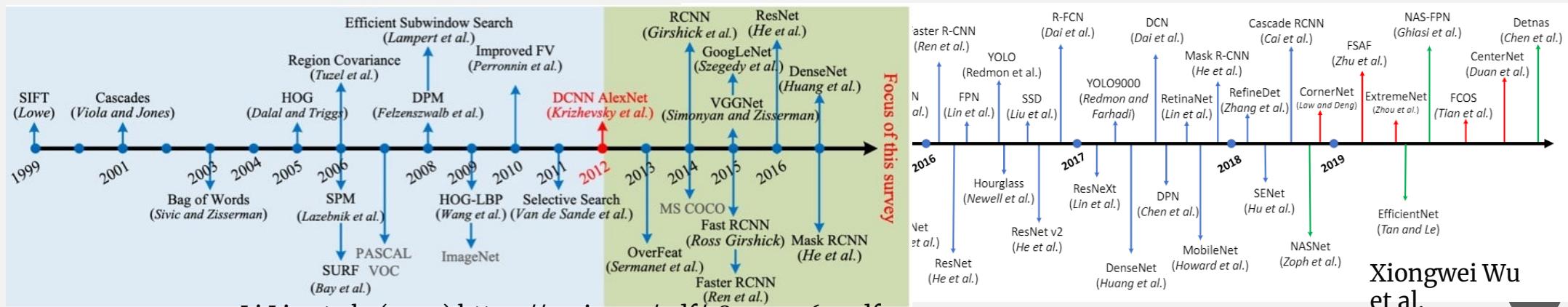


Challenge 2017

- I. Object localization for 1000 categories.
- II. Object detection for 200 fully labeled categories.
- III. Object detection from video for 30 fully labeled categories.

# OBJECT RECOGNITION TRENDS

- Deeper, wider or more flexible layers, introducing dropout, data augmentation, GPU parallel computation
  - AlexNet, ZFNet, GoogLeNet and VGG were the winner and runner-up of the ILSVRC-2014, ResNet,
  - EfficientNet (Scaling CNNs)
- Transfer learning concept
- More work on object detection / segmentation



Li Liu et al., (2019) <https://arxiv.org/pdf/1809.02165.pdf>