PATTERN RECOGNITION USING PYTHON

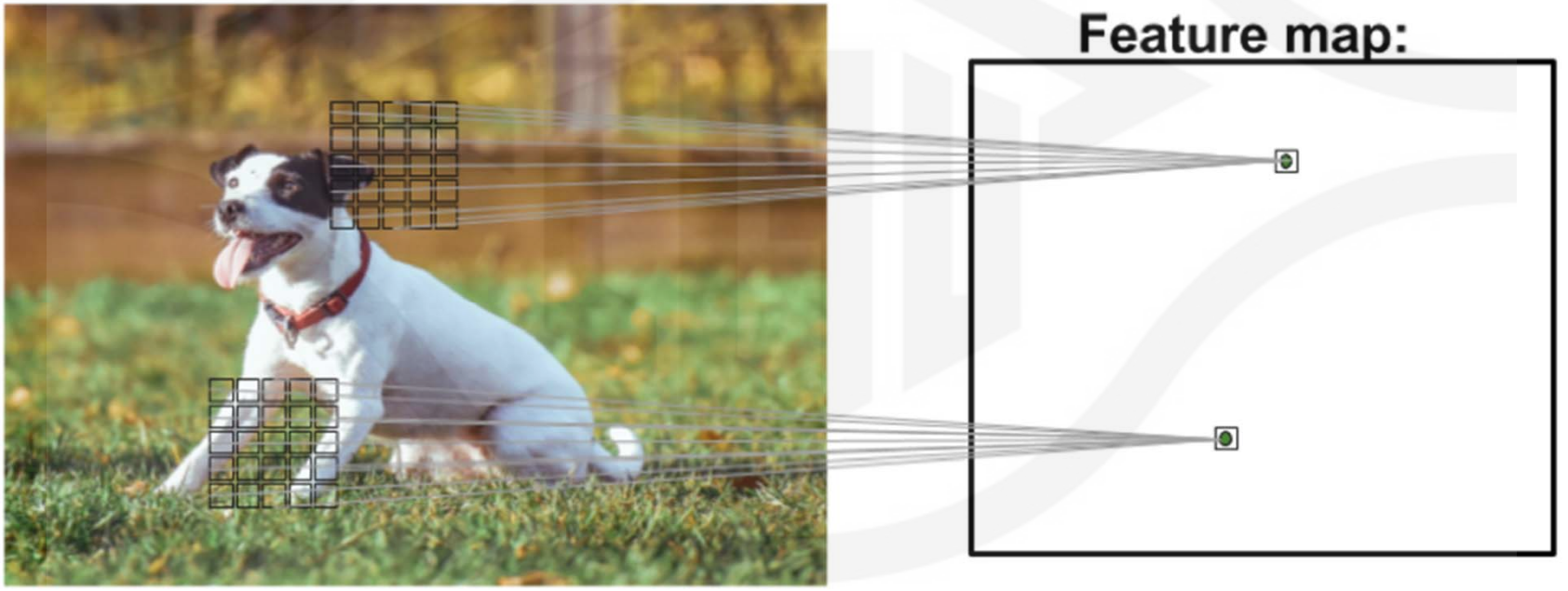
Convolutional Neural Networks

Wen-Yen Hsu
Dept Electrical Engineering
Chang Gung University, Taiwan

2019-Spring

Learning Hierarchies Feature

- CNN computes **feature maps** from an input image, where each element comes from a local patch of pixels in the input image

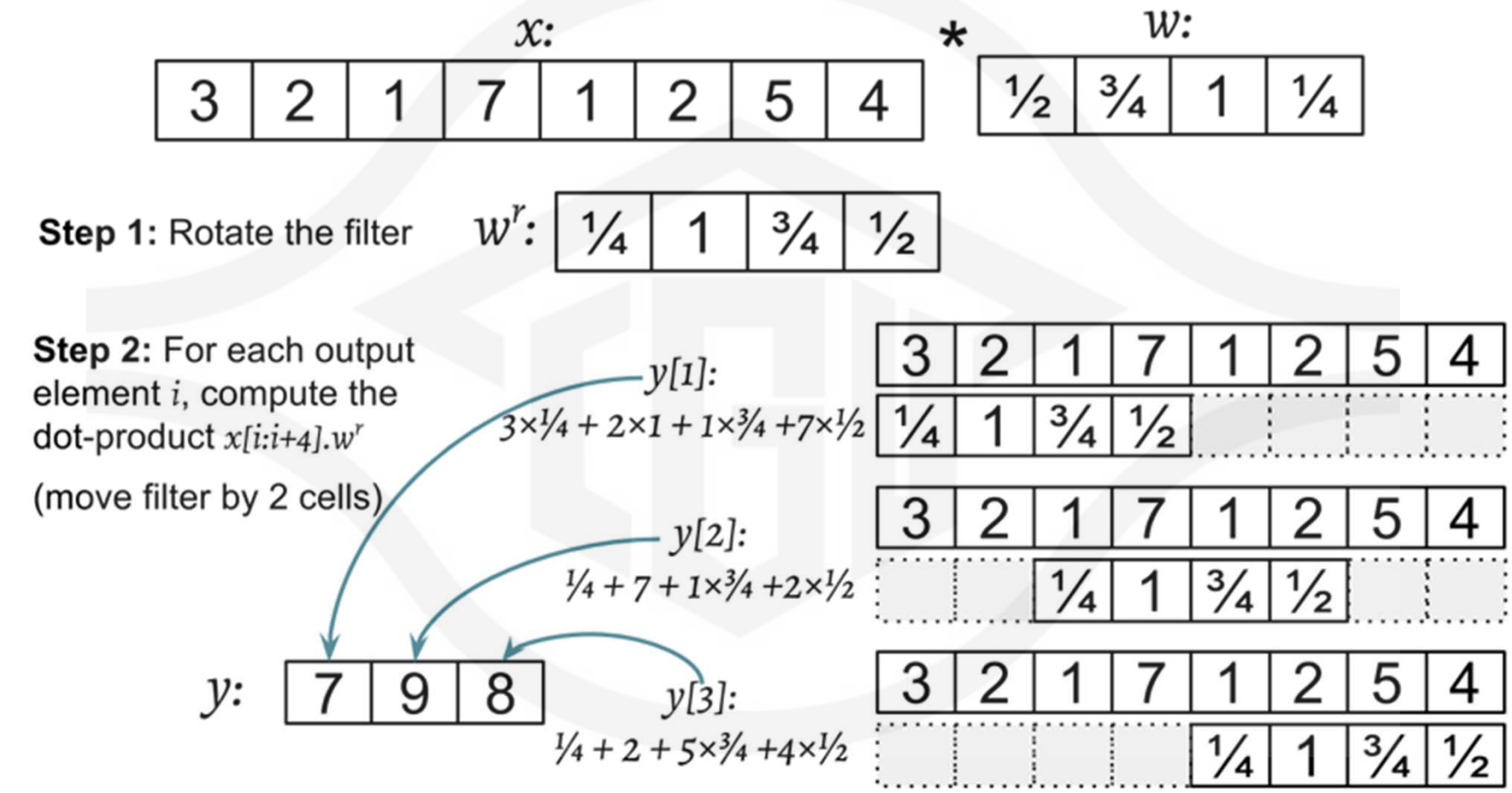


Discrete Convolution in One Dimension

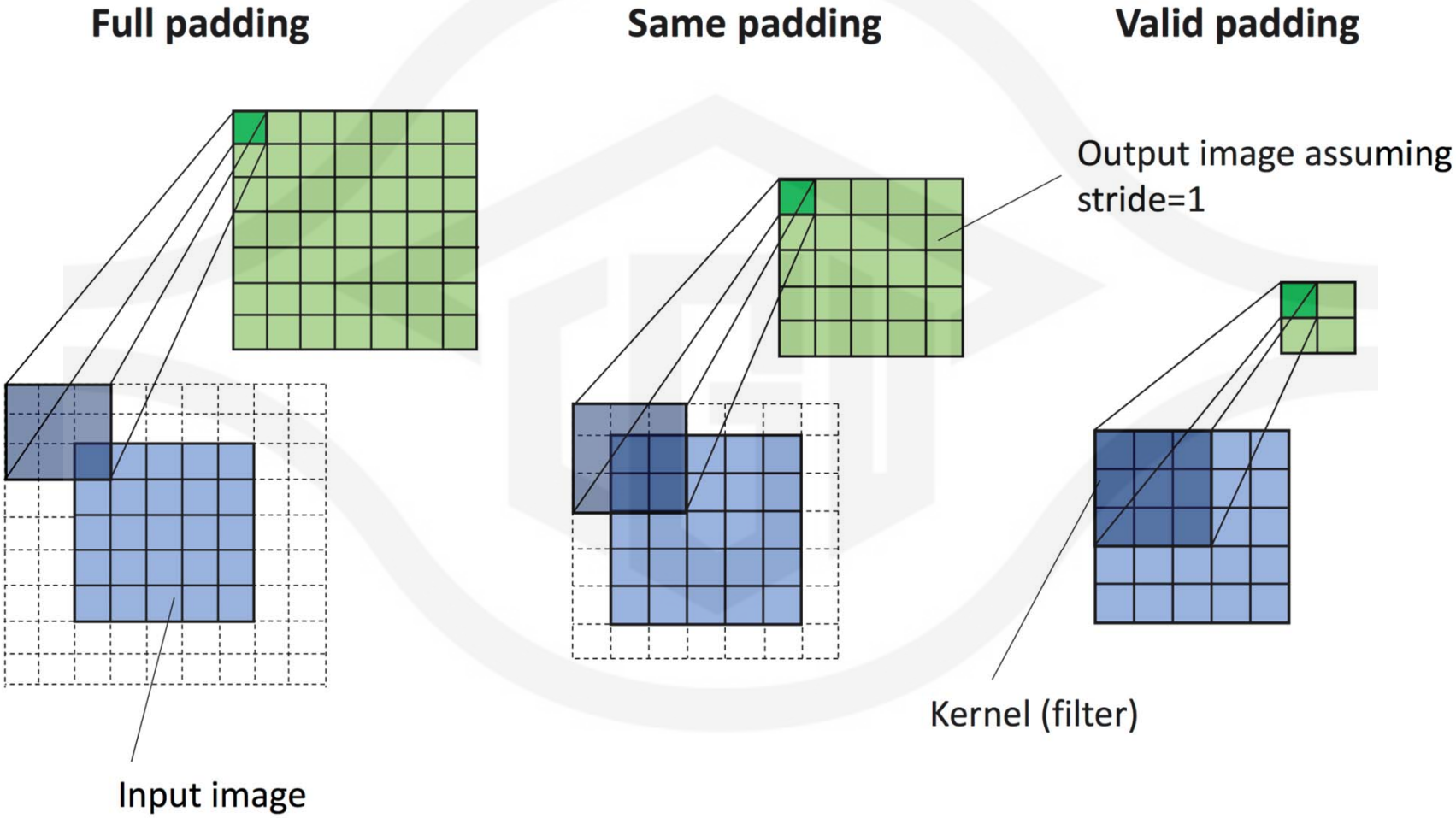
- Discrete convolution for two one-dimensional vectors x and w
- Vector x is our input or signal
- Vector w is called the **filter** or **kernel**

$$y = x * w \rightarrow y[i] = \sum_{k=-\infty}^{+\infty} x[i-k]\,w[k]$$

Performing Discrete Convolutions



Padding


Full padding
Same padding
Valid padding
Output image assuming stride=1
Kernel (filter)
Input image

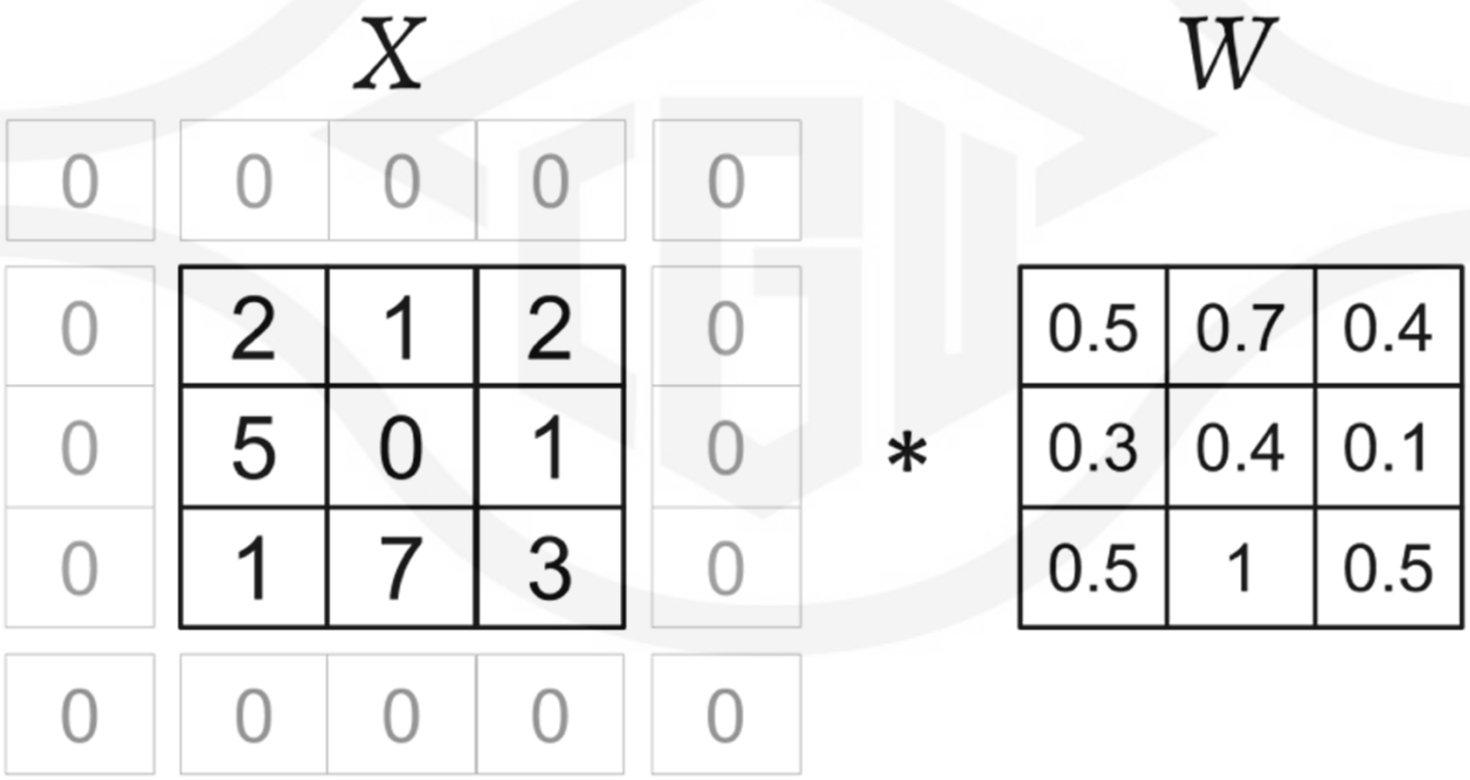
Size of Output

- Input vector x has size n and the filter w is of size m
- The size of the output resulting from x, w with padding p and stride s is determined as follows:

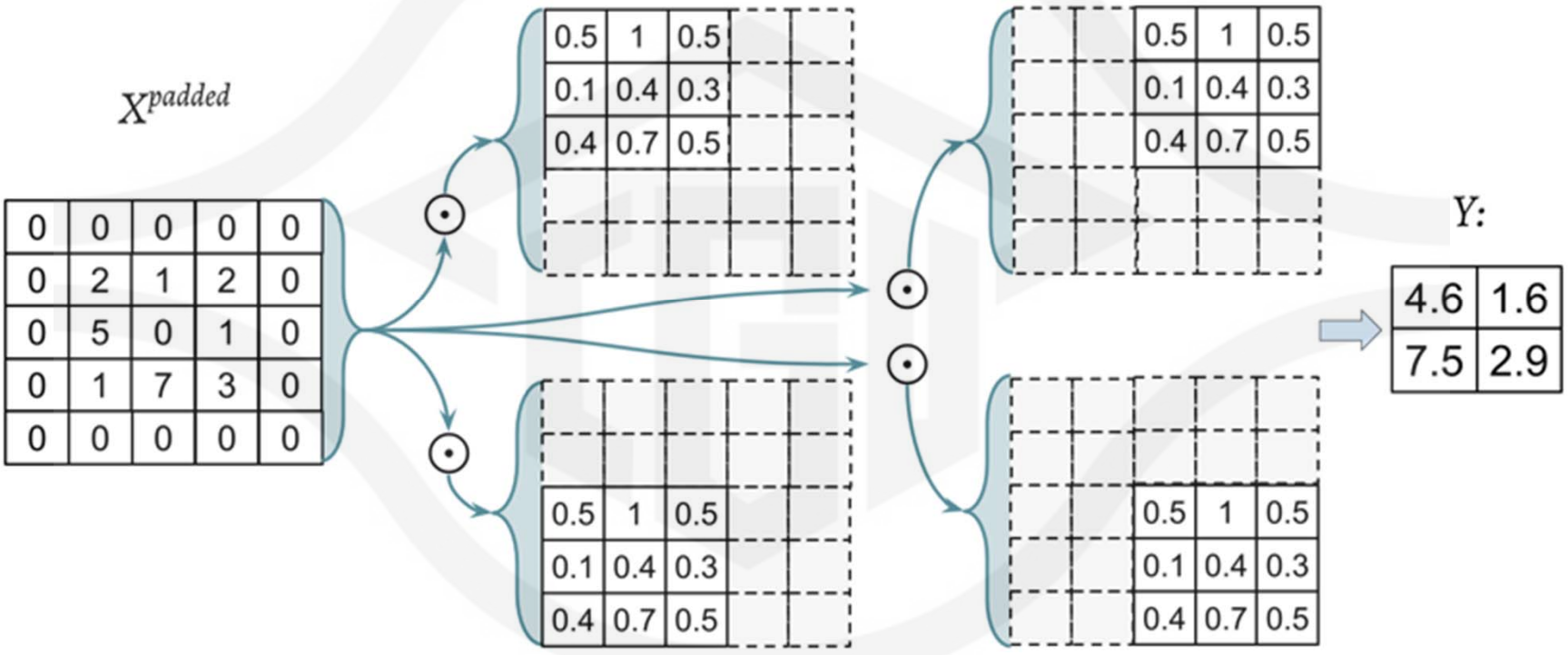
$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1$$

Discrete Convolution in 2D

$$Y = X * W \rightarrow Y[i,j] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} X[i-k_1, j-k_2] W[k_1, k_2]$$



Performing Convolutions

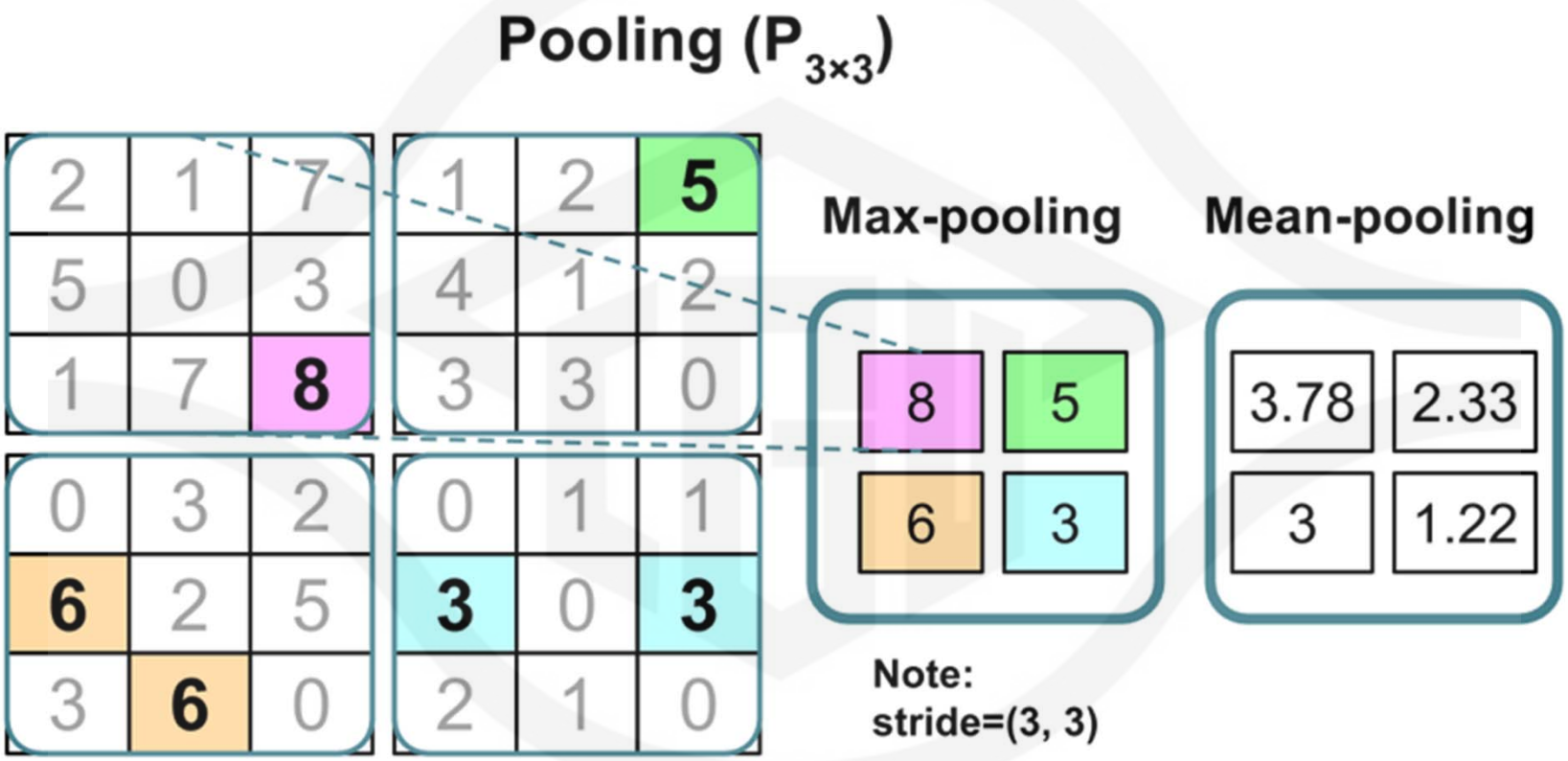


Convolution

- Convolution from scratch

```
def conv2d(X, kernel, pad, stride):
    X_pad = np.pad(X, ((pad, pad), (pad, pad)), 'constant',
constant_values=0)
    kernel_cov = kernel[::-1, ::-1]
    k_x = kernel.shape[0]
    k_y = kernel.shape[1]
    s_x = stride
    s_y = stride
    opt_x = int(((X_pad.shape[0]-k_x)/s_x)+1)
    opt_y = int(((X_pad.shape[1]-k_y)/s_y)+1)
    cov = np.zeros((opt_x, opt_y))
    for i in range(opt_x):
        for j in range(opt_y):
            cov[i][j] =
np.sum(np.multiply(X_pad[i*s_x:i*s_x+k_x, j*s_y:j*s_y+k_y],
kernel_cov))
    return cov
```

Sub-sampling



Performing Max Pooling

$$X_1 = \begin{bmatrix} 10 & 255 & 125 & 0 & 170 & 100 \\ 70 & 255 & 105 & 25 & 25 & 70 \\ 255 & 0 & 150 & 0 & 10 & 10 \\ 0 & 255 & 10 & 10 & 150 & 20 \\ 70 & 15 & 200 & 100 & 95 & 0 \\ 35 & 25 & 100 & 20 & 0 & 60 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} 100 & 100 & 100 & 50 & 100 & 50 \\ 95 & 255 & 100 & 125 & 125 & 170 \\ 80 & 40 & 10 & 10 & 125 & 150 \\ 255 & 30 & 150 & 20 & 120 & 125 \\ 30 & 30 & 150 & 100 & 70 & 70 \\ 70 & 30 & 100 & 200 & 70 & 95 \end{bmatrix}$$

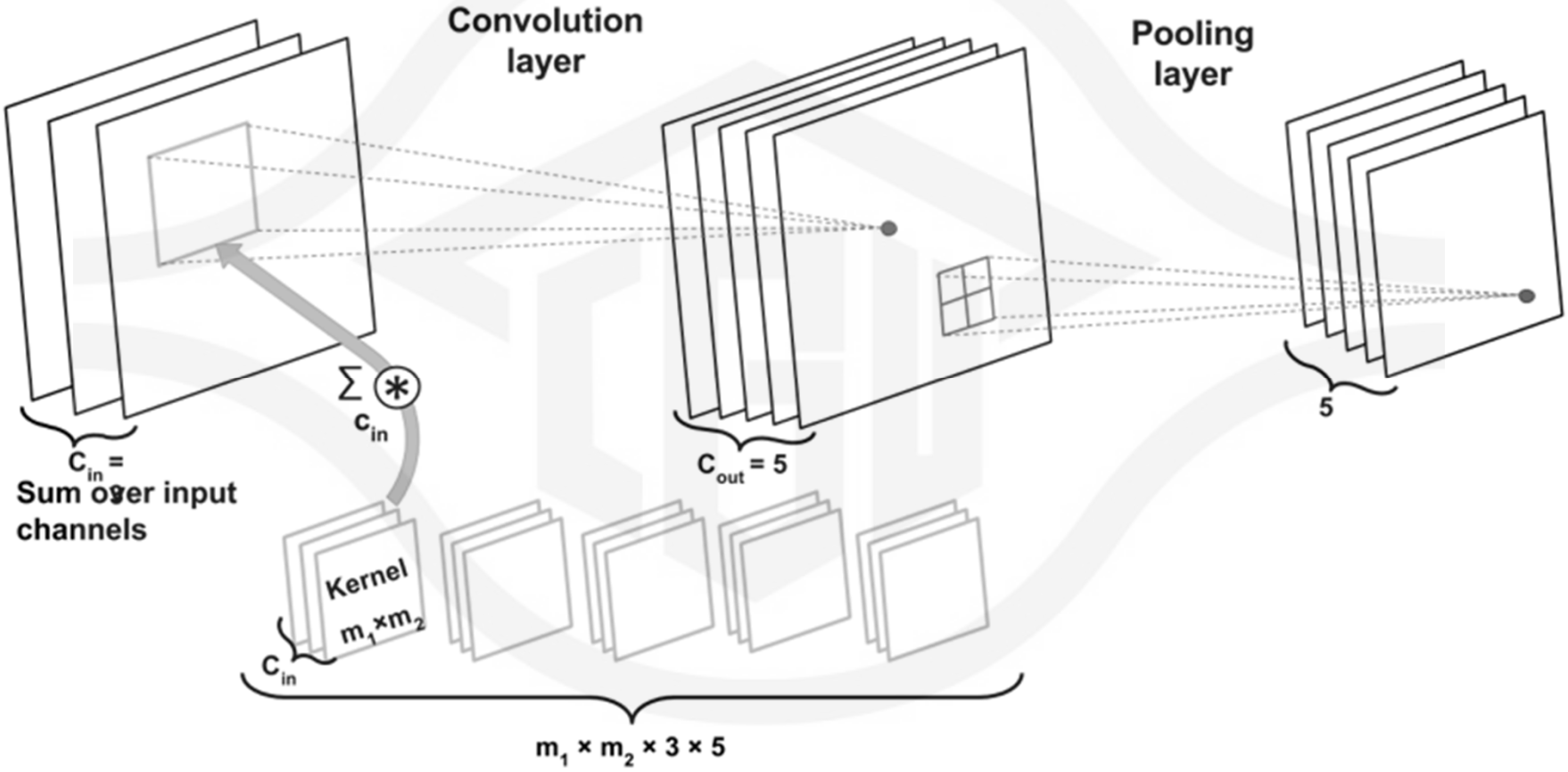
$$\xrightarrow{\max\text{-pooling } P_{2\times 2}} \begin{bmatrix} 255 & 125 & 170 \\ 255 & 150 & 150 \\ 70 & 200 & 95 \end{bmatrix}$$

Max Pooling

- Max pooling from scratch

```
def max_pool2d(X, pool_size, stride):
    p_x = pool_size
    p_y = pool_size
    s_x = stride
    s_y = stride
    opt_x = int(((X.shape[0]-p_x)/s_x)+1)
    opt_y = int(((X.shape[1]-p_y)/s_y)+1)
    mp = np.zeros((opt_x, opt_y))
    for i in range(opt_x):
        for j in range(opt_y):
            mp[i][j] = np.max(X[i*s_x:i*s_x+p_x,
j*s_y:j*s_y+p_y])
    return mp
```

Combine Convolution & Max Pooling

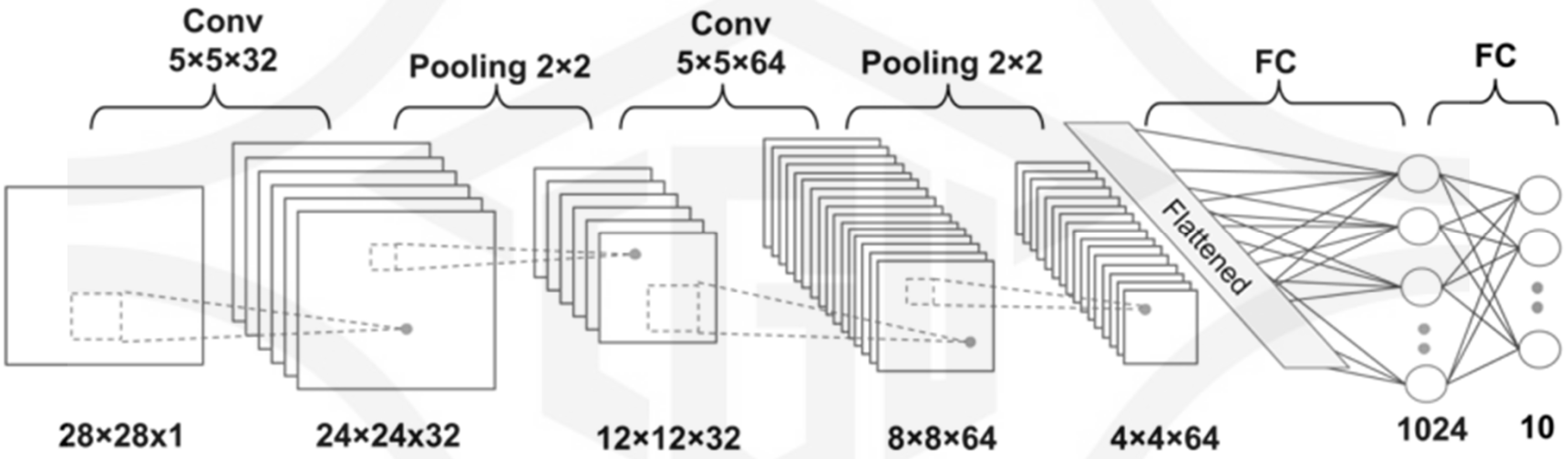


Convolution & Max Pooling in PyTorch

- Define CNN in pytorch

```
 def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 1 output channels, 5x5
square convolution
        self.conv1 = nn.Conv2d(1, 1, 5)
    def forward(self, x):
        x = self.conv1(x)
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(x, (2, 2))
        return x
```

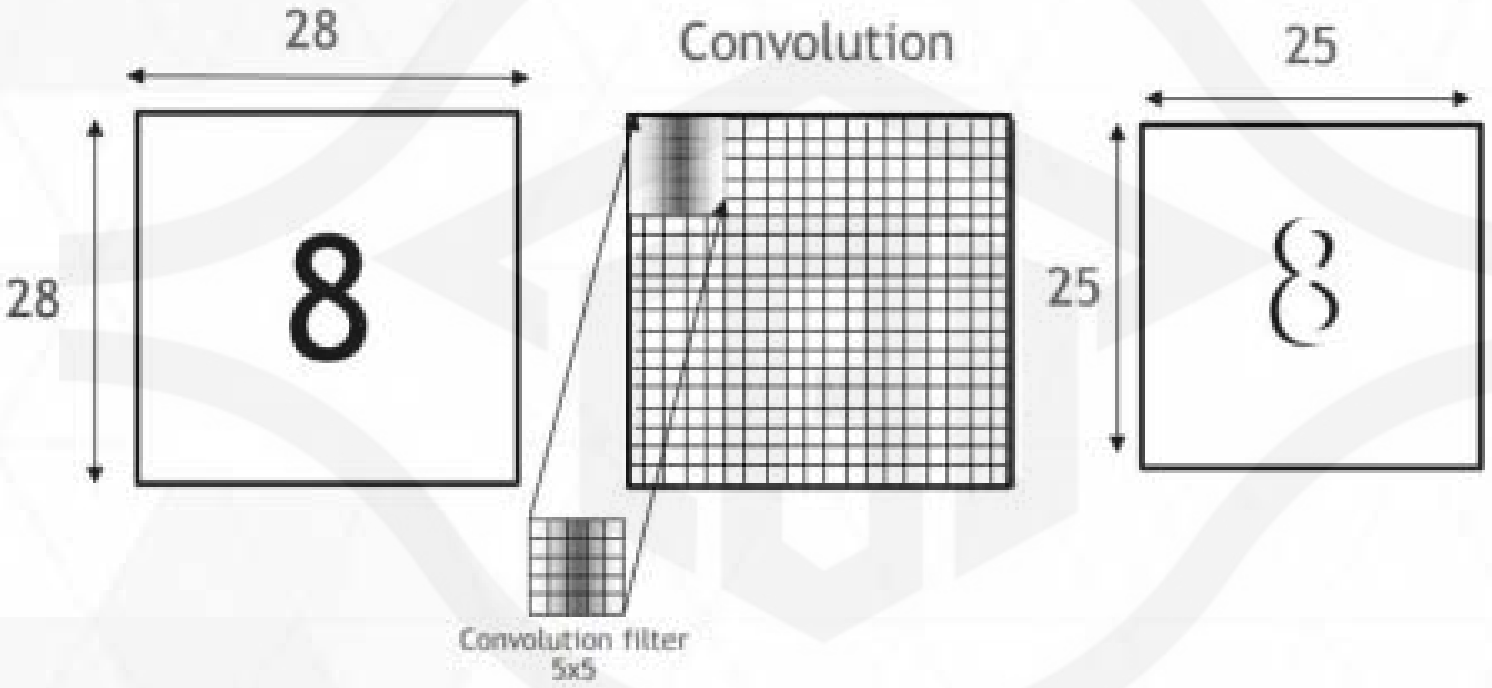
Multilayer CNN Architecture



Classical CNN : LeNet (Yann Lecun, 1989)

http://yann.lecun.com/exdb/lenet/

Convolution in LeNet



Zero padding, input is reduced by the radius of the kernel
Input[28x28]*filter[5x5]=FeatureMap[25x25]

Digits Classification with PyTorch CNN

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        # 1 input image channel, 20 output channels, 5x5 square
convolution
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 50, 5)
        self.fc1 = nn.Linear(800, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.tanh(self.conv1(x))
        x = F.max_pool2d(x, (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.tanh(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.tanh(self.fc1(x))
        x = self.fc2(x)

        return x
```

CNN Visualization

https://www.youtube.com/watch?v=0pnaB7ptu50 What a Convolution Neural Network sees in the image? - Visualizing a classic CNN

Feature Extraction

- ***Traditional Machine Learning***

Feature Extractor

Trainable Classifier

Hand engineered

Trainable

- ***Multilayer neural nets***

Trainable

Low-Level Features

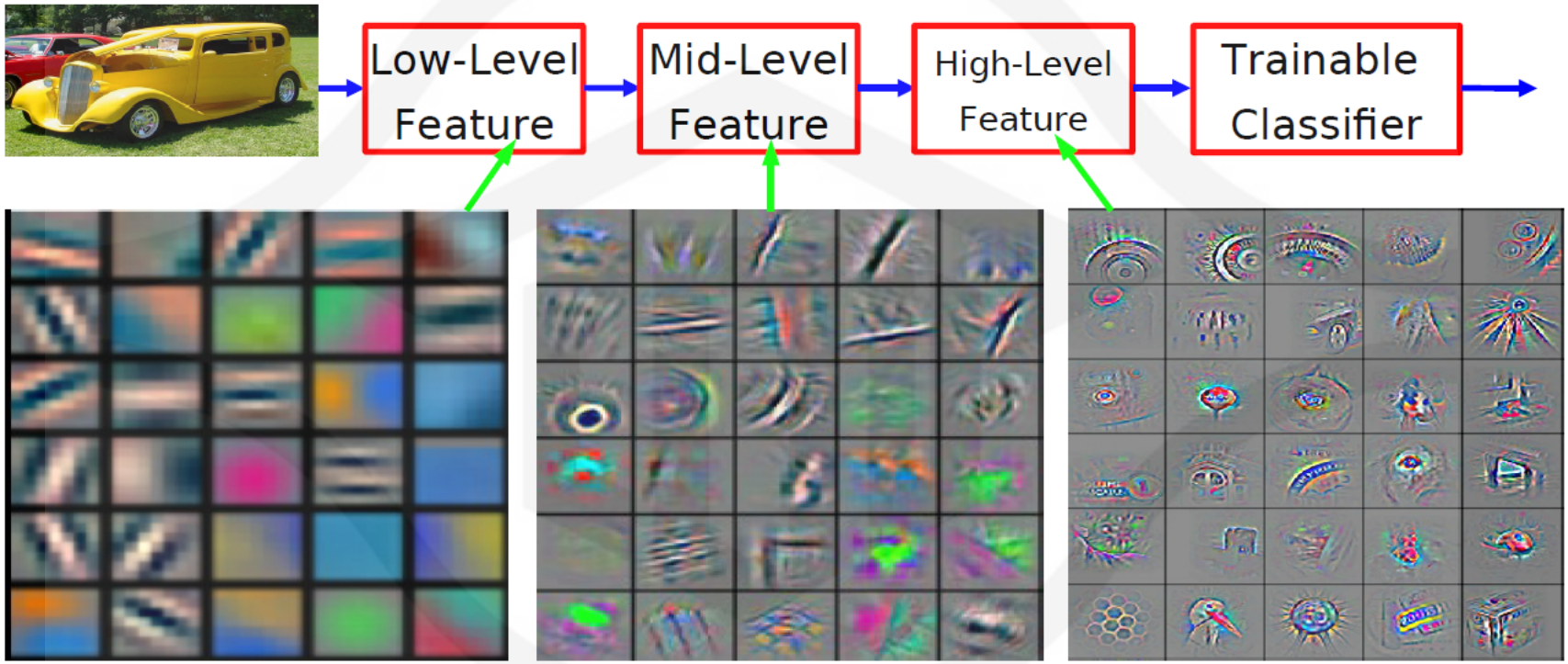
Mid-Level Features

High-Level Features

Trainable Classifier

Hierarchically Feature Representation

Natural is data is compositional => it is efficiently representable hierarchically

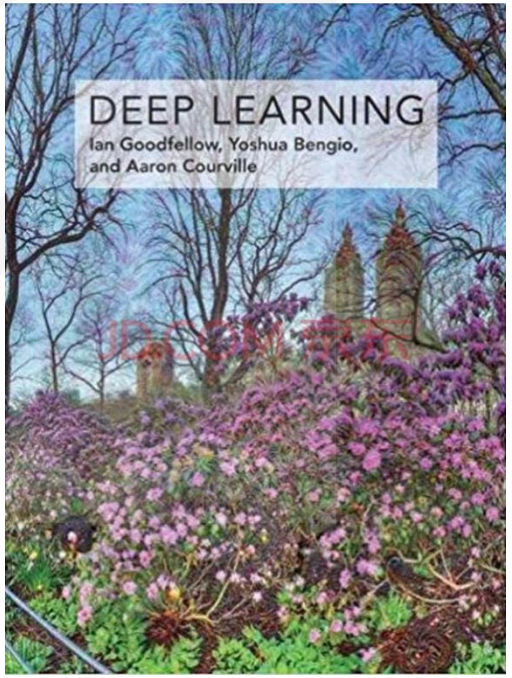


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

The Era of Deep Learning (2012~)

- Reform neural networks from the 1980s
  - computer vision application(CNN)
  - sequence model application (RNN)
- New training tricks and activation functions
- Import GPU parallelization computation
- Big data is ready

- As a basic element expand to other areas
  - Transfer Learning
  - Reinforcement Learning
  - Generative Adversarial Network



Deep Learning Authority

- 2018 Turing Award (Nobel Prize of Computing)

NYTIMES.COM

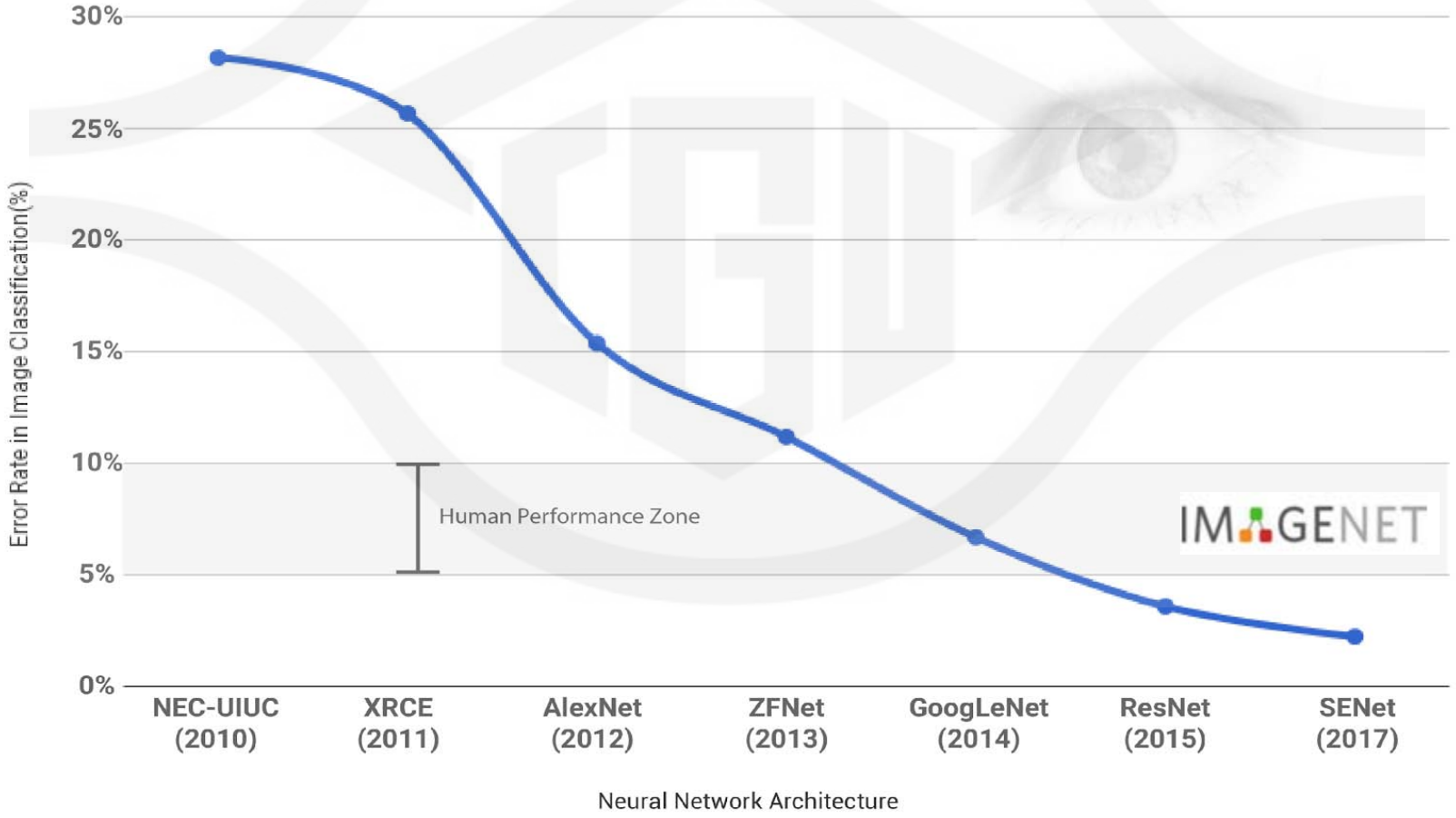
Three Pioneers in Artificial Intelligence Win Turing Award

For their work on neural networks, Geoffrey Hinton, Yann LeCun an…

https://amturing.acm.org/

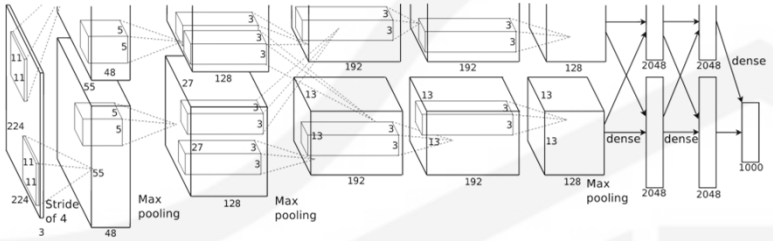
ILSVRC 2010-2017 (Fei-Fei Li)

- (**I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**ompetition) dataset scale : 15 Million

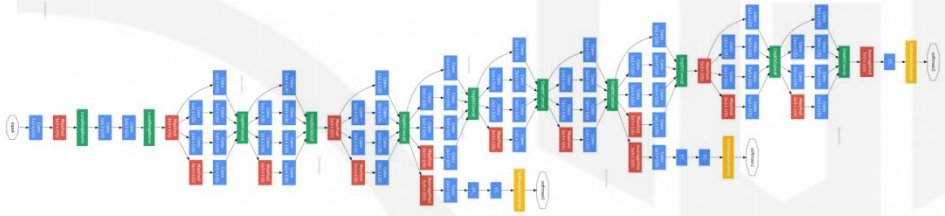


Depth Inflation

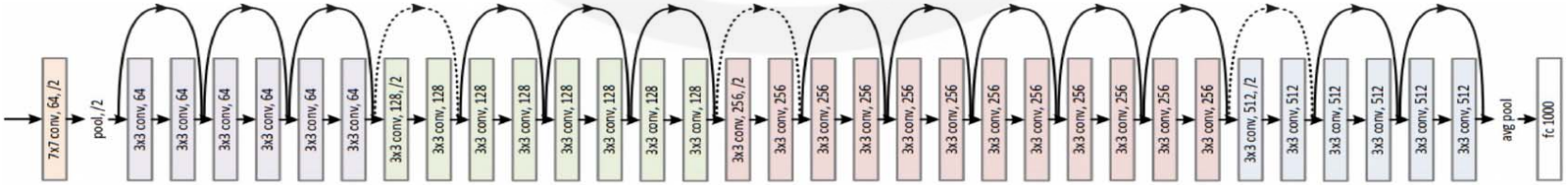
- AlexNet **8-Layer** (ILSVRC 2012)



- GoogLeNet **22-Layer** (ILSVRC 2014)

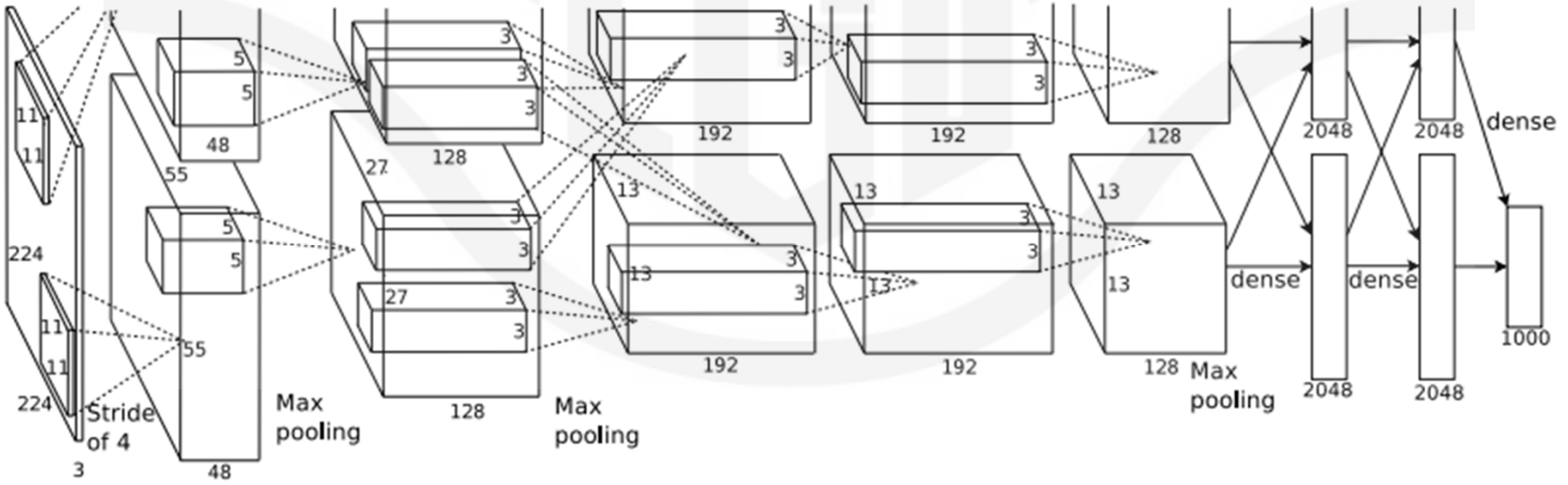


- ResNet **152-Layer** (ILSVRC 2015)



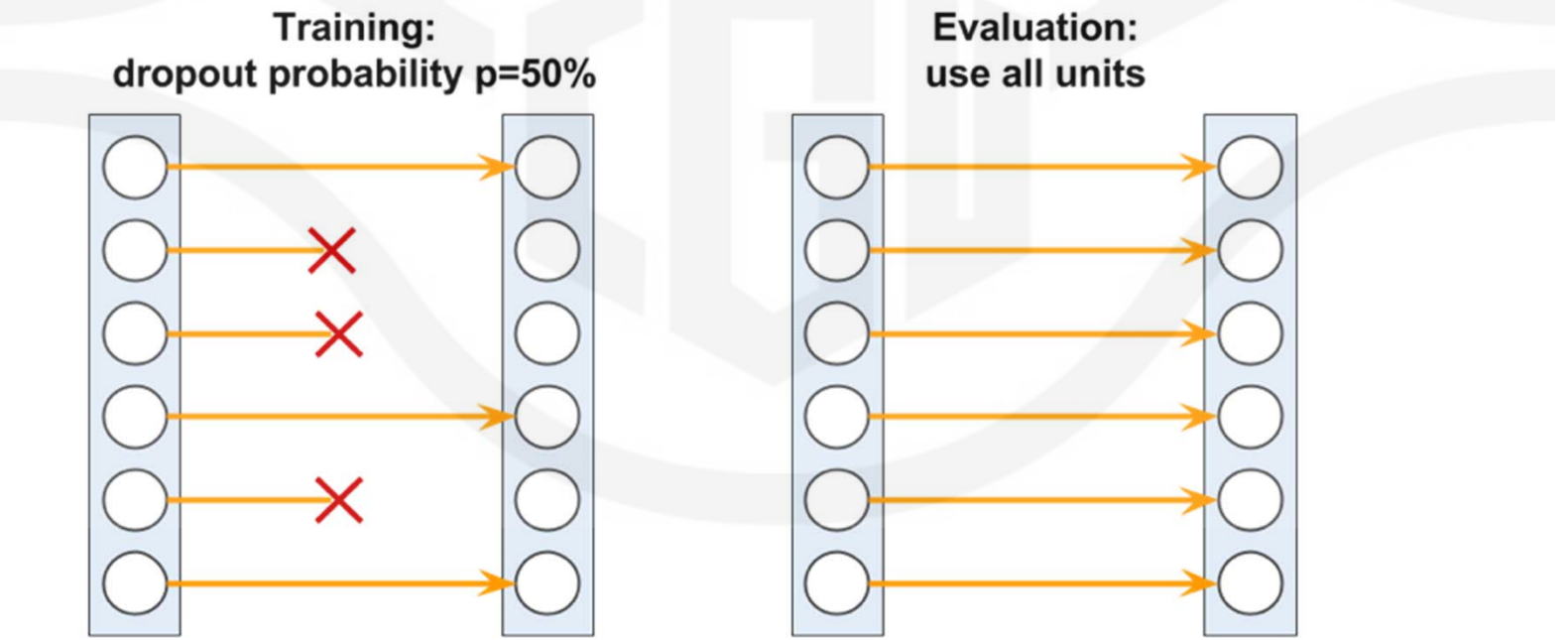
AlexNet

- Use **Dropout** technology
- Replaced Sigmoid and Tanh with **ReLu**
- Run two weeks of training with **two GTX 580 GPUs**
- Prove more layer can make the performance of CNN better



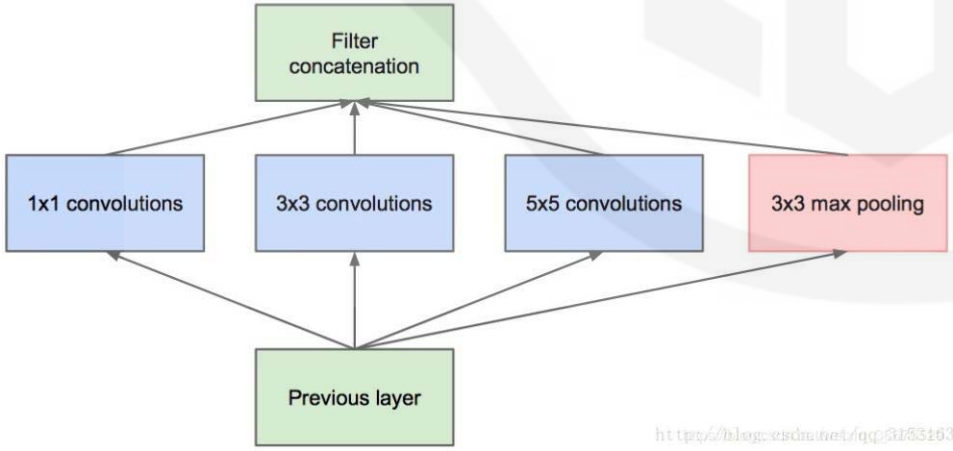
Regularizing with Dropout

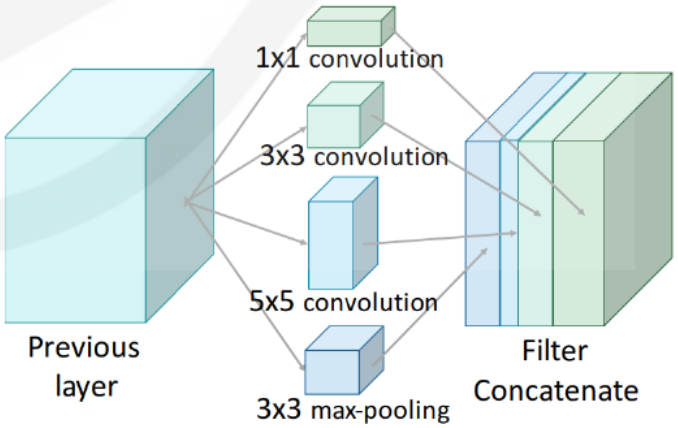
- Dropout is usually applied to the hidden units of higher layers. During the training phase of a neural network, a fraction of the hidden units is **randomly** dropped at every iteration with probability p_{drop}



GoogLeNet

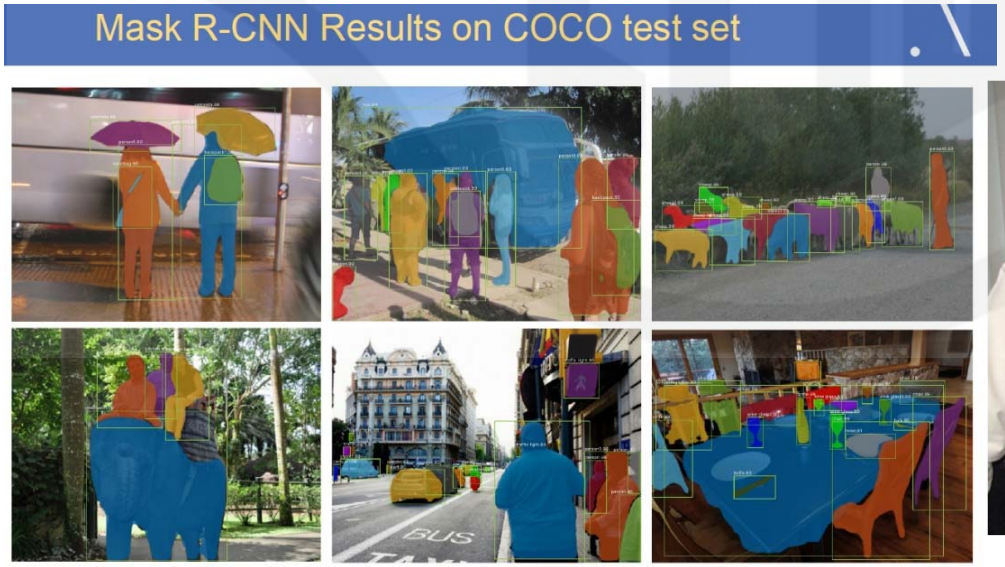
- **Inception** network in network architecture (parallel different filter sizes of convolution and max-pooling)
- **GAP** (Global Average Pooling) replaces the FC layer
- The number of **hyperparameters** is **less than** Alexnet, the computational **efficiency** is **quite good**, but the **accuracy** is **higher**





Kaiming He 何恺明

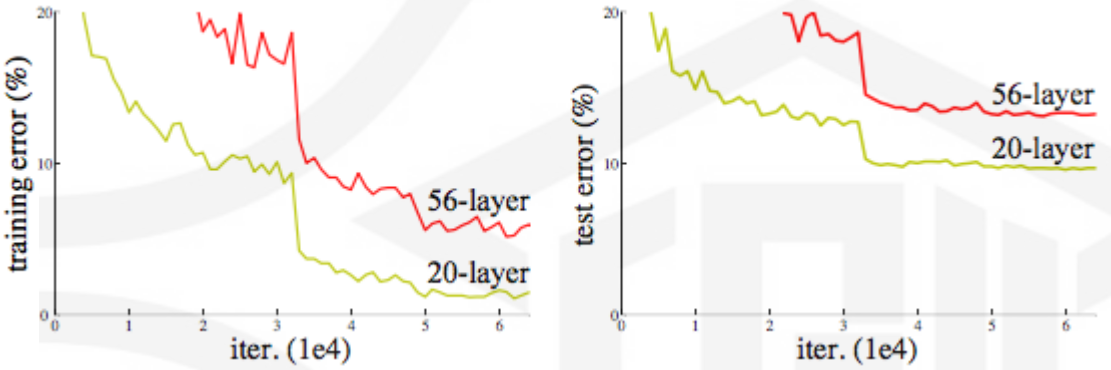
- Haze Removal (2009 CVPR Best Paper Award)
- Parametric ReLU (torch.nn.functional.prelu()) (2015)
- He initialization (torch.nn.init.kaiming_uniform_()) (2015)
- Residual Network (2015 ILSVRC champion)
  (2016 CVPR best paper award)
- Focal Loss (2017 ICCV Best Student Paper Award)
- Mask R-CNN (2017 ICCV Best Paper Award)

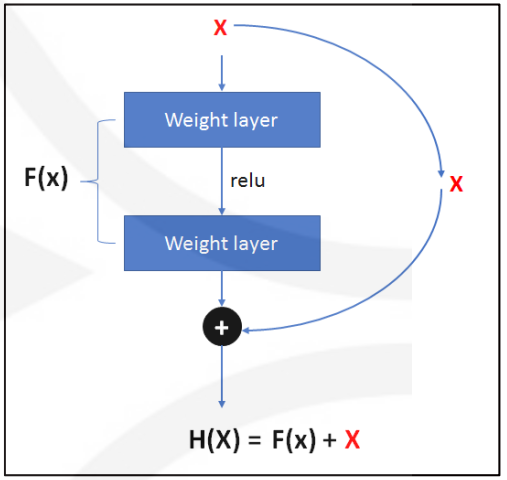


http://kaiminghe.com/

ResNet (The deeper, the better ?)

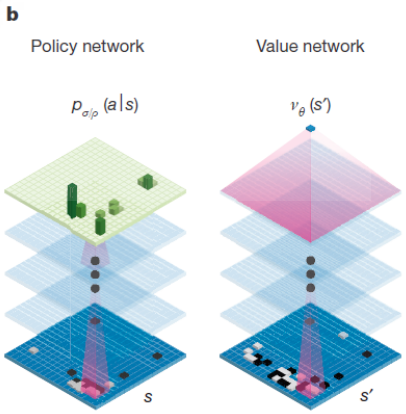
- Propose **Residual Block** solve **degradation** problem





- AlphaGo using ResNet to construct policy or value network (RL)

b, Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.



Tricks in Deep Learning

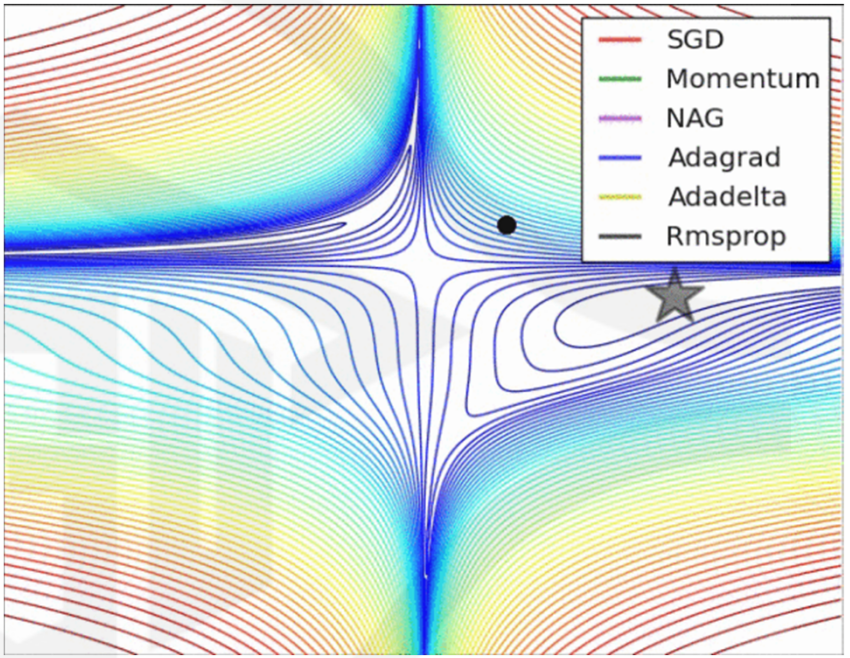
- New type activation functions (ReLu, PReLu, Selu)
- Regularization (Dropout)
- Initialization (Xavier init, Kaiming init)
- Internal Covariate Shift (Batch Normalization)
- Optimizer with adaptive learning rates(Momentum, AdaGrad, RMSProp, Adam)

Adaptive Learning Rates

- Momentum

$$V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + V_t$$



- AdaGrad

$$W \leftarrow W - \eta \frac{1}{\sqrt{n+\epsilon}} \frac{\partial L}{\partial W}$$

$$n = \sum_{r=1}^{t} \left(\frac{\partial L_r}{\partial W_r}\right)^2$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{\sum_{r=1}^{t} \left(\frac{\partial L_r}{\partial W_r}\right)^2 + \epsilon}} \frac{\partial L}{\partial W}$$

PyTorch Apply Tricks

```
class LeNet_modify(nn.Module):
    def __init__(self):
        super(LeNet_modify, self).__init__()
        self.prelu = nn.PReLU()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.bn1 = nn.BatchNorm2d(20)
        self.conv2 = nn.Conv2d(20, 50, 5)
        self.bn2 = nn.BatchNorm2d(50)
        self.fc1 = nn.Linear(800, 500)
        self.bn3 = nn.BatchNorm1d(500)
        self.dropout1 = nn.Dropout(p=0.5)
        self.fc2 = nn.Linear(500, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.prelu(x)
        x = F.max_pool2d(x, (2, 2))
        x = F.max_pool2d(self.prelu(self.bn2(self.conv2(x))), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = self.dropout1(self.prelu(self.bn3(self.fc1(x))))
        x = self.fc2(x)
        return x
```

Training and Testing Phase

- Training Phase

```
for epoch in range(epoch_time):
    loss_average = np.zeros(1)
    model.train()
    for step, (batch_x, batch_y) in enumerate(loader):
        optimizer.zero_grad()
        prediction = model(batch_x)
        loss = loss_func(prediction, batch_y)
        loss.backward()
        optimizer.step()
```

- Testing Phase

```
model.eval()
y_test_hat_tensor = model(X_test_tensor)
```

PyTorch Apply Tricks (Initialization and Optimizer)

- Initialization

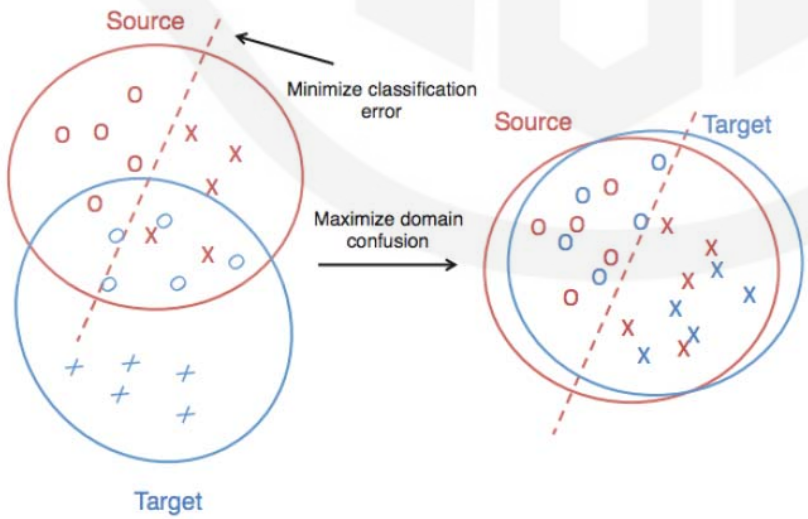
```
def init_weights(m):
    if type(m) == nn.Linear:
        t.nn.init.kaiming_uniform_(m.weight, mode='fan_in')
    if type(m) == nn.Conv2d:
        t.nn.init.kaiming_uniform_(m.weight, mode='fan_in')
model = LeNet_modify()
model.apply(init_weights)
```

- Optimizer

```
optimizer = t.optim.Adam(model.parameters(), lr = learning_rate)
```

Transfer Learning (Optional)

- Using labeled data from **source domain** and transferring the learned knowledge to a **target domain**
- Using a CNN on a very **large dataset**, and then further fine-tuning it on the **target dataset** which is **relatively small** in size (Pre-trained models)
- The key advantage of transfer learning is that it **removes the need** to create a large dataset required to train the CNN



PyTorch with ILSVRC Model (Optional)

- Pre-trained models

```
import torchvision.models as models

resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenetv2(pretrained=True)
```

- Normalization

```
T.Normalize(mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225])
```

Reference

- Sebastian Raschka, Vahid Mirjalili. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow. Second Edition. Packt Publishing, 2017.
- Vishnu Subramanian. Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch. Packt Publishing, 2018.