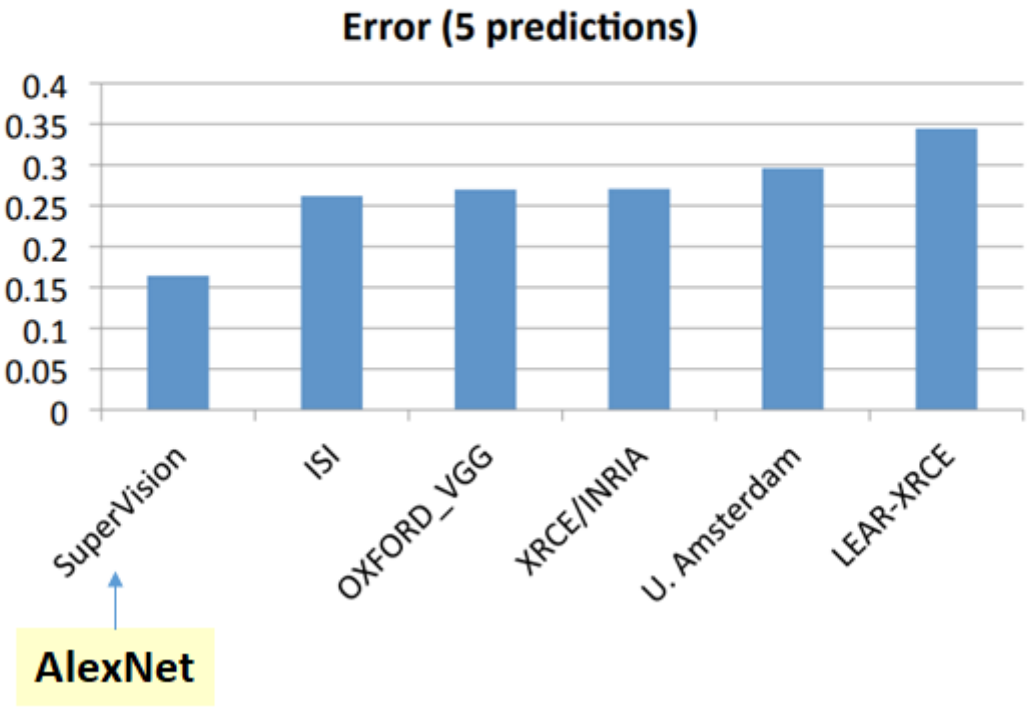
AlexNet

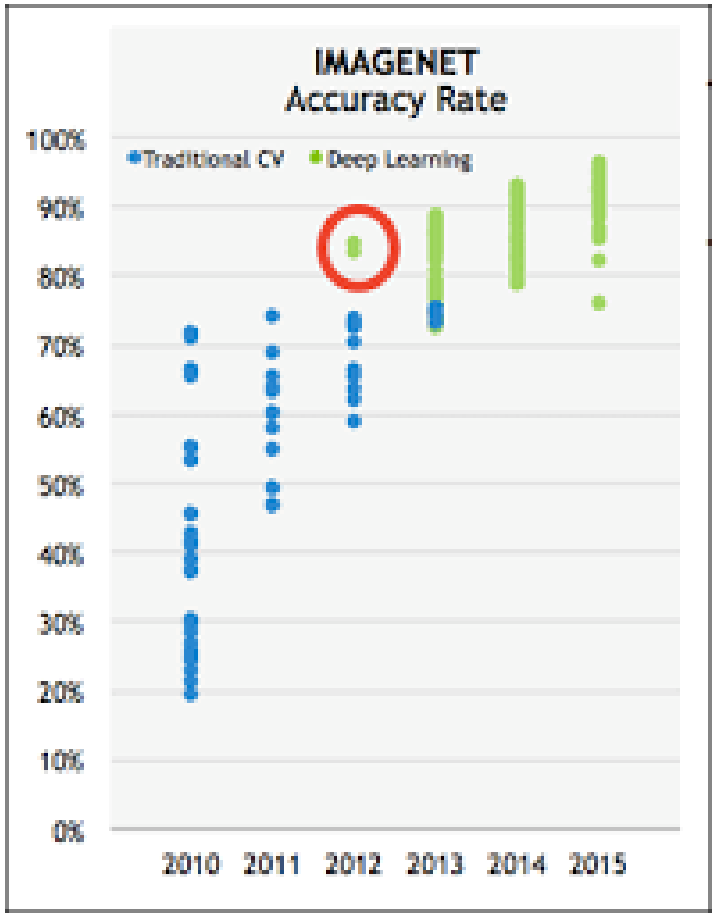
Kyeong Hwan Moon

01 AlexNet

Introduction

Ranking of the best results from each team





01 Dataset

Inspection


n02097047 (196)


n01682714 (40)


n03134739 (522)


n04254777 (806)


n02859443 (449)


n02096177 (192)


n02107683 (239)

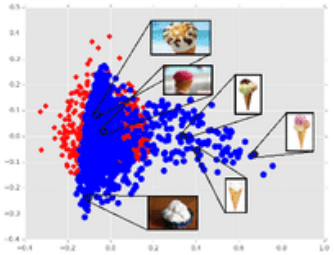

n01443537 (1)


n02264363 (318)


Icecream - ILSVRC12 (IN)


Icecream - WIN

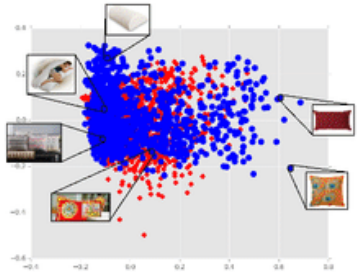

Icecream - WINC


ILSVR red, WINC blue


Pillow - ILSVRC12 (IN)

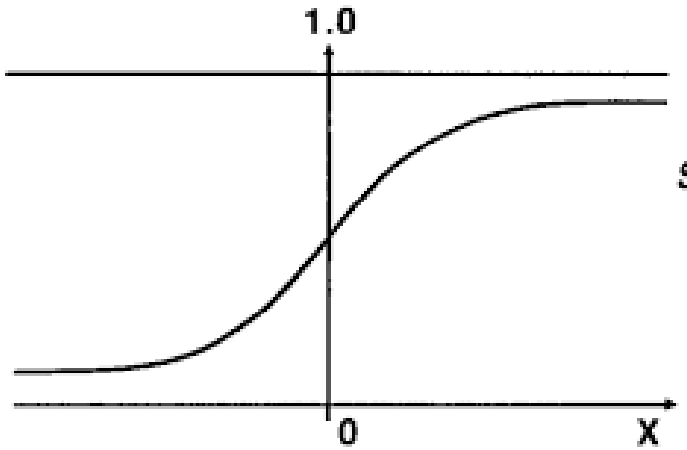

Pillow - WIN


Pillow - WINC


ILSVR red, WINC blue

01 Relu

sigmoid



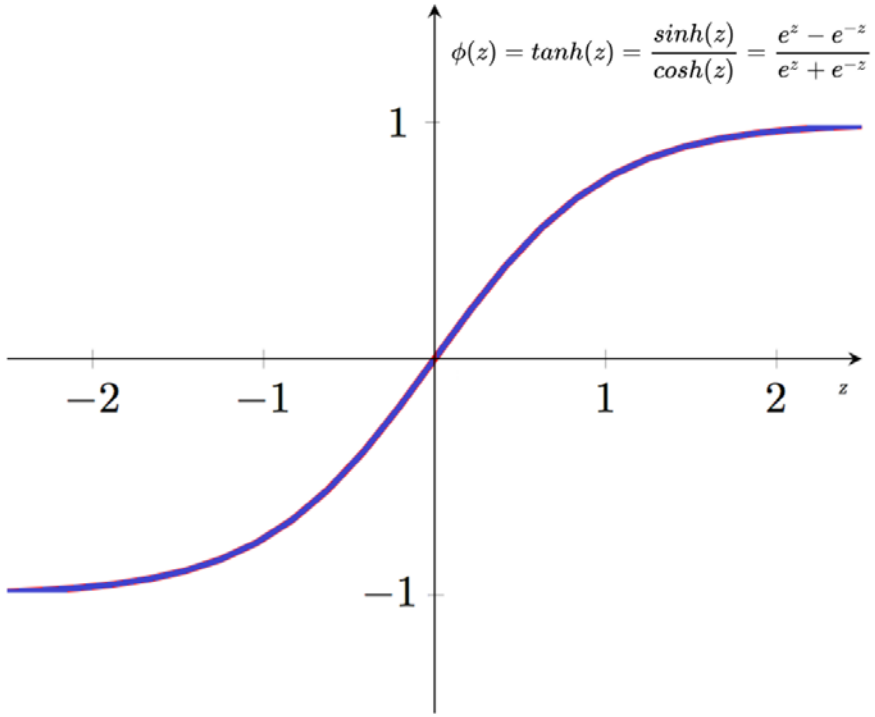
Sigmoid

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

- Returns 0 or 1
- Use when classifying binary data
- Easy to solve outlier problem

01 Relu

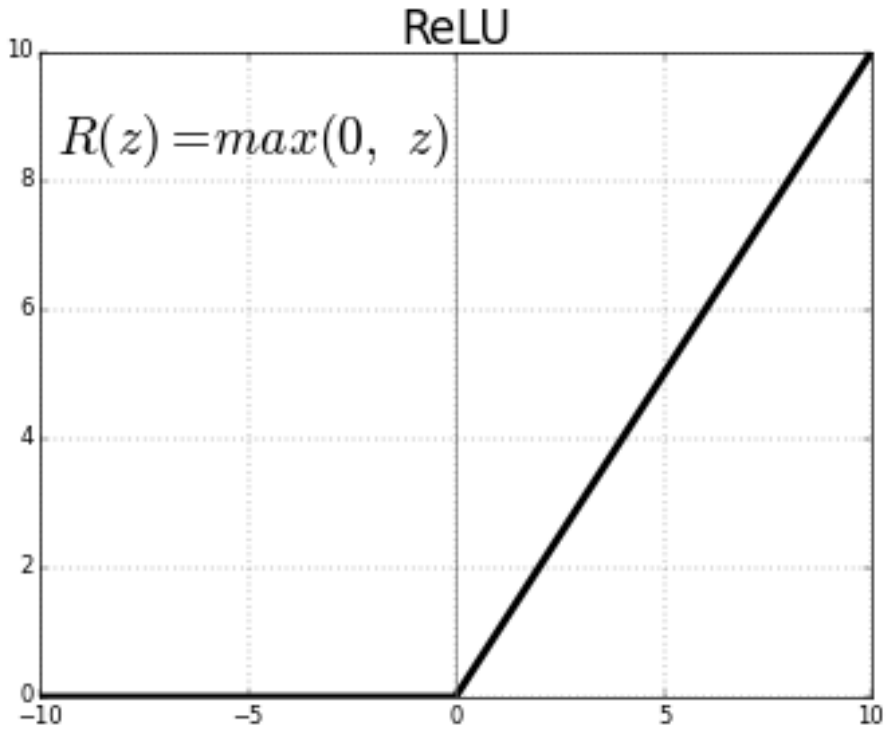
tanh(x)



- Median is 0 -> solve bias problem occurred in sigmoid
- But vanishing gradient problem still remaining

01 ReLU

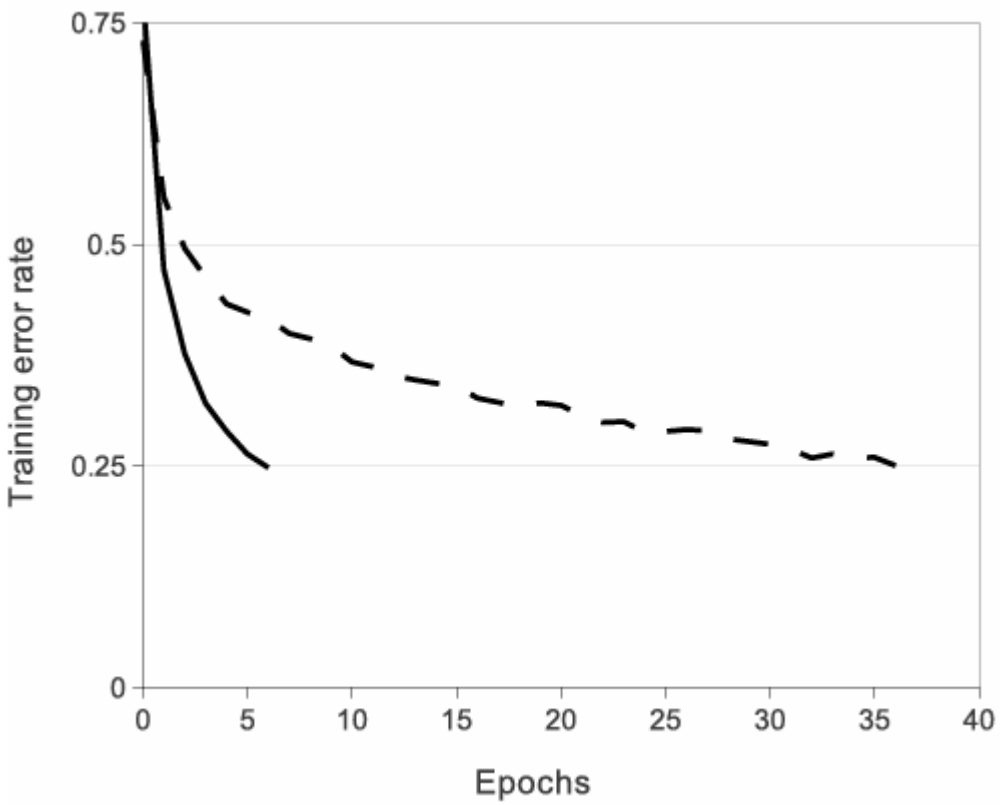
ReLU



- X > 0 -> Straight line with gradient 1
  X <= 0 -> 0
- Fast training
- Decrease cost of training

01 ReLU

Training Speed



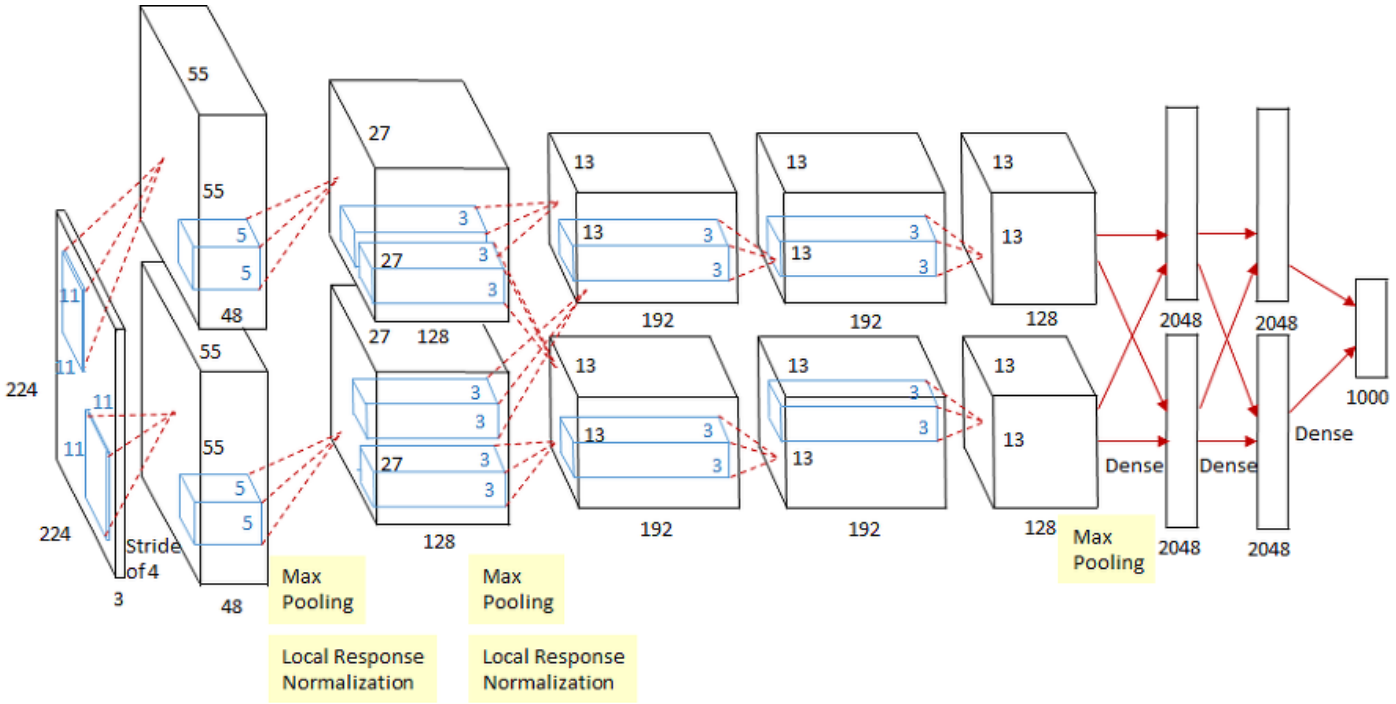
- ReLU is faster than tanh(x) when iterations required to reach 25% training error

** - - - : tanh(x)
----- : ReLU

01

GPU

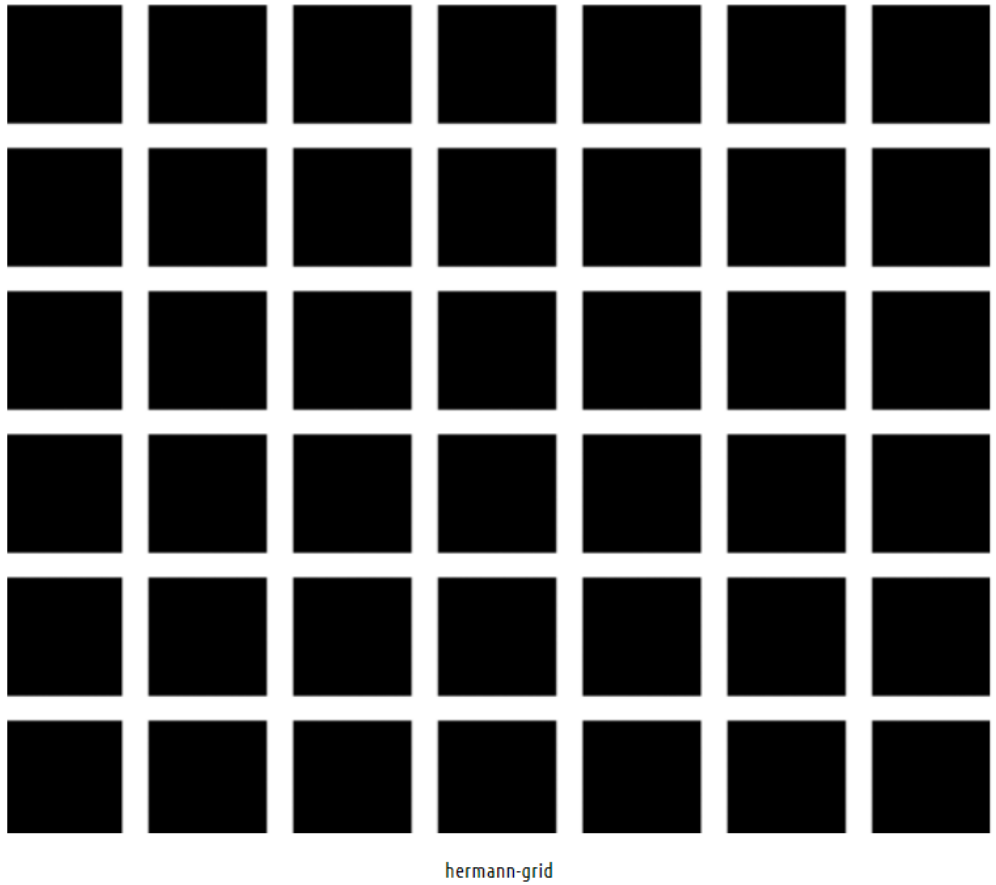
Why multiple GPU?



- Memory of each GPU was 3GB. But the training dataset was too big to fit on GPU.
- GPU are able to read from and write to one another's memory directly
- GPUs communicate in certain layers

01 Local Response Normalization

Lateral inhibition



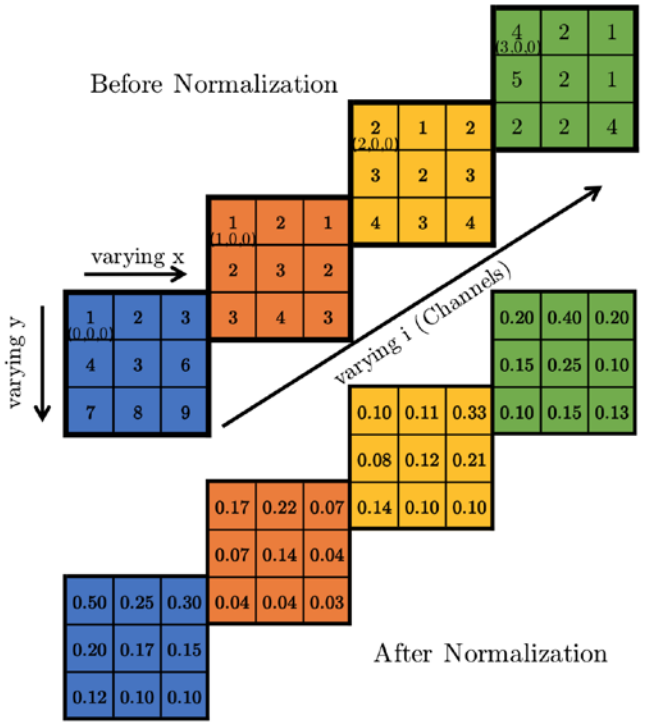
hermann-grid

- The dots are shown because of lateral inhibition (white color is inhibited -> shown inhibited color)

01 Local Response Normalization

LRN

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

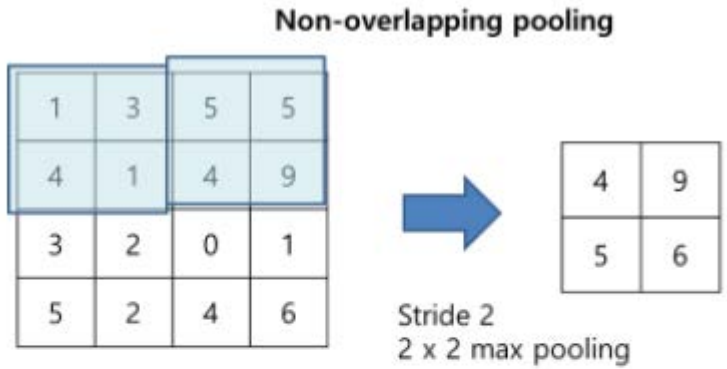


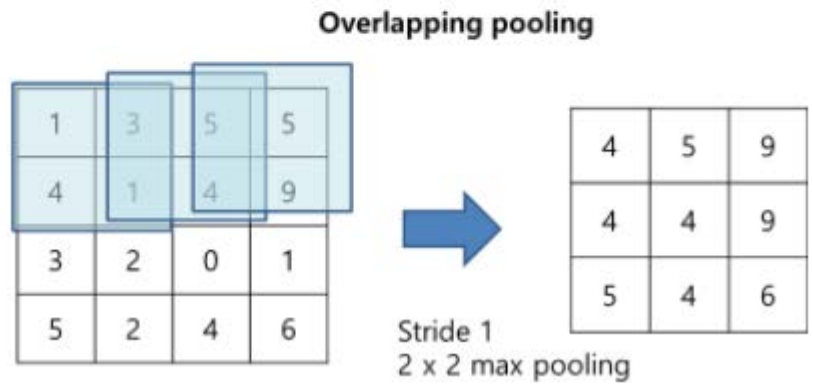
- N : The total number of kernels in the layer
- k = 2
- n = 5
- α = 10−4
- β = 0.75

- If sigma function(right side) is low, b will be high. On contrast if sigma function is high, b will be low
- It helps to generalize the learning

01 Overlapping Pooling

Overlapping Pooling

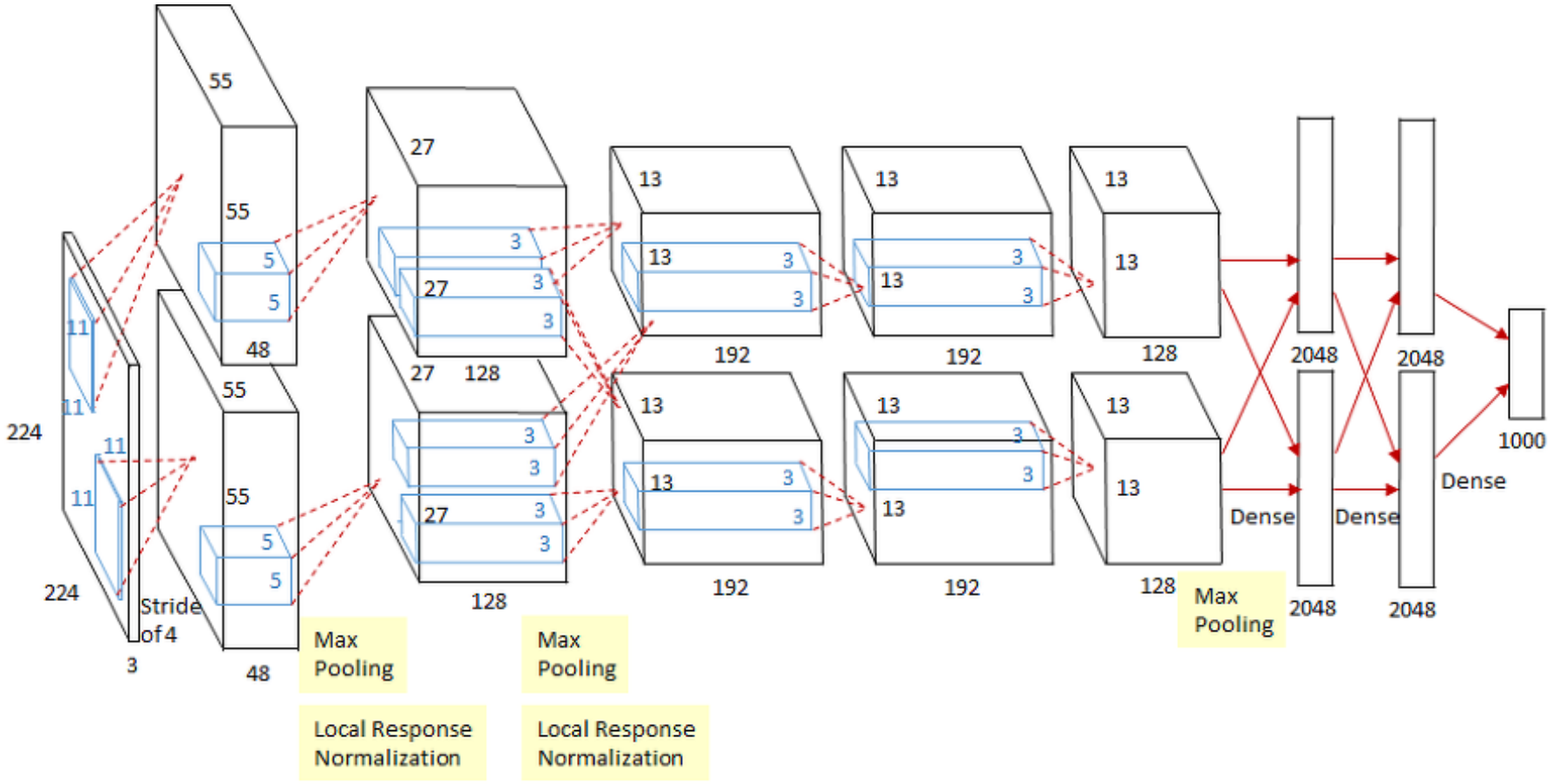




- This scheme reduces the top-1 and top-5 error rates by 0.4% and 0.3%

01 AlexNet

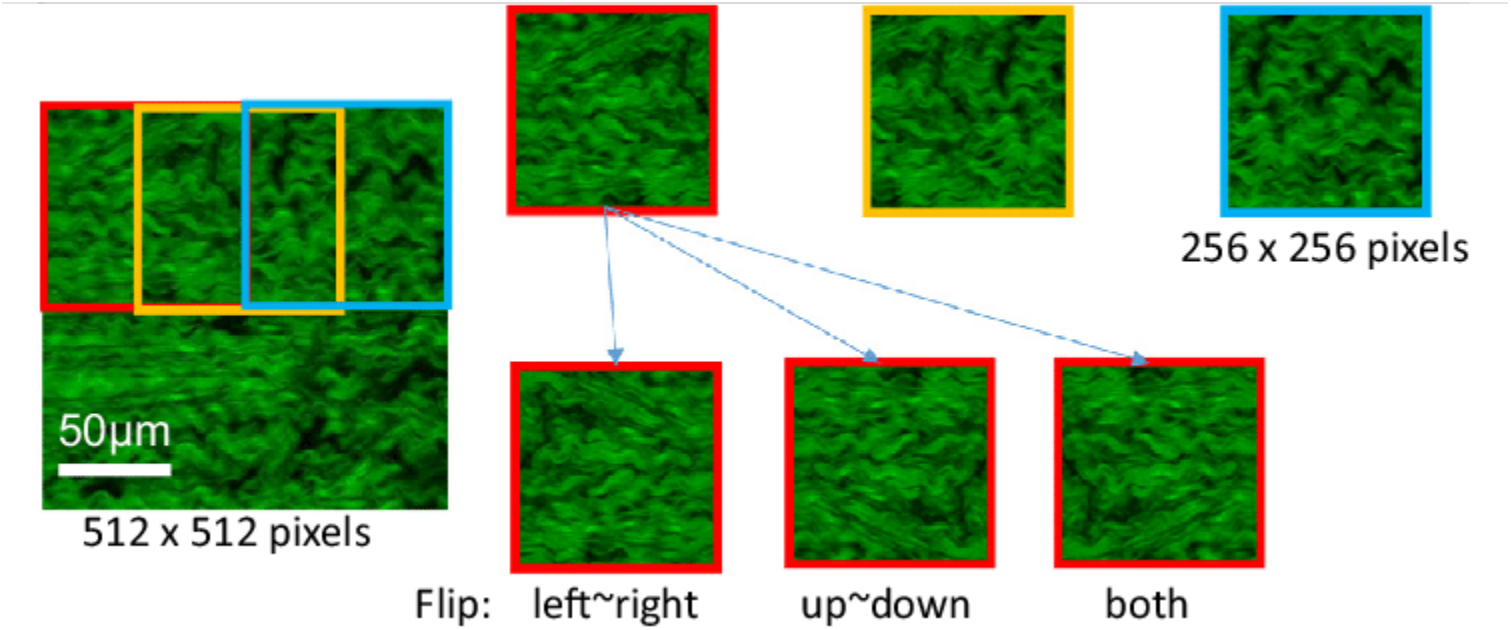
AlexNet



01 Reducing Overfitting

Data Augmentation

extracting random 224×224 patches

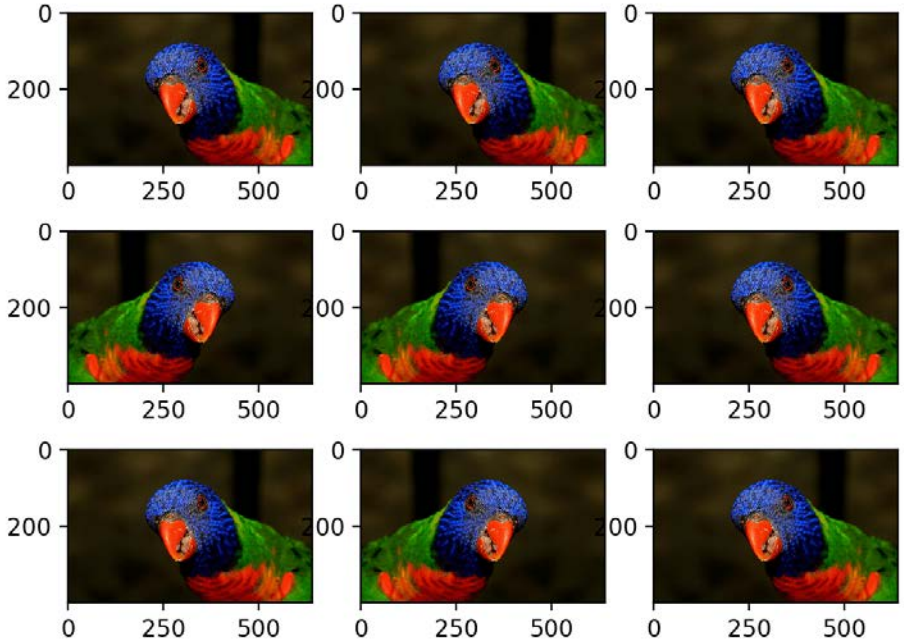


** Size of image is different compared with original AelxNet data augmentation

01 Reducing Overfitting

Data Augmentation

horizontal reflections

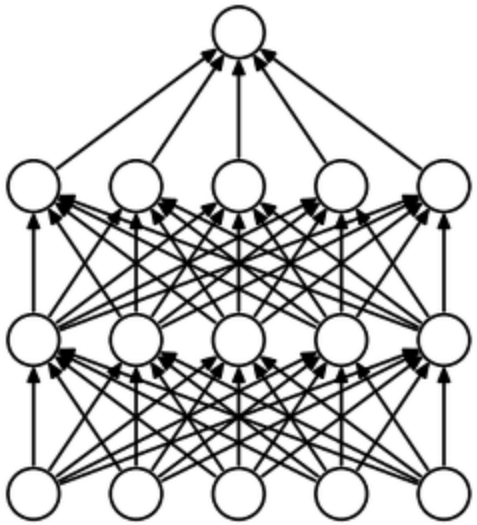


Color jittering

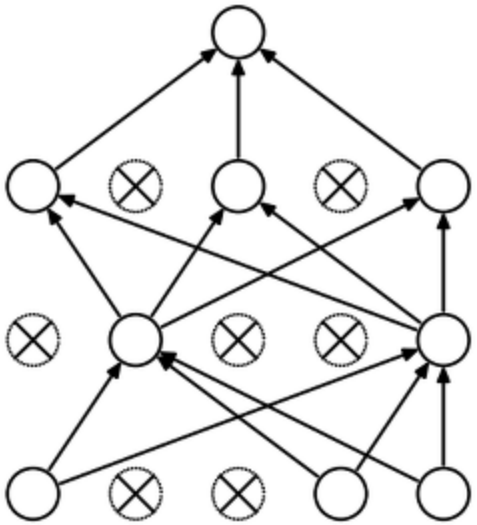
$$[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

01 Dropout

Dropout



(a) Standard Neural Net



(b) After applying dropout.

01 Code

```
class AlexNet(nn.Module):
  def __init__(self, classes=1000):
    """
    classes : number of classes to classify
    """
    super().__init__()
    self.classes = classes

    # conv layers
    self.net = nn.Sequential(
        nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11, stride=4),
        nn.ReLU(),
        nn.LocalResponseNorm(size=5, alpha=0.0001, beta=0.75, k=2),
        nn.MaxPool2d(kernel_size=3, stride=2),

        nn.Conv2d(96, 256, 5, padding=2),
        nn.ReLU(),
        nn.LocalResponseNorm(size=5, alpha=0.0001, beta=0.75, k=2),
        nn.MaxPool2d(kernel_size=3, stride=2),

        nn.Conv2d(256, 384, 3, padding=1),
        nn.ReLU(),

        nn.Conv2d(384, 384, 3, padding=1),
        nn.ReLU(),

        nn.Conv2d(384, 256, 3, padding=1),
        nn.ReLU(),

        nn.MaxPool2d(kernel_size=3, stride=2),
    )
```

```
    # fully connected layers
    self.fc = nn.Sequential(
        nn.Dropout(p=0.5, inplace=True),

        nn.Linear(in_features=(256 * 6 * 6), out_features=4096),
        nn.ReLU(),
        nn.Dropout(p=0.5, inplace=True),

        nn.Linear(in_features=4096, out_features=4096),
        nn.ReLU(),

        nn.Linear(in_features=4096, out_features=classes),
    )

    self.initialize_bias_weights()

  # initialize bias and weights
  def initialize_bias_weights(self):
    for layer in self.net:
      if isinstance(layer, nn.Conv2d):
        nn.init.normal_(layer.weight, mean=0, std=0.01)
        nn.init.constant_(layer.bias, 0)

    for idx in [4, 10, 12]:
      nn.init.constant_(self.net[idx].bias, 1)

  def forward(self, x):
    x = self.net(x)
    x = x.view(-1, 256 * 6 * 6)
    x = self.fc(x)
    return x
```

Thank You