GNU

Lab Seminar: 2022. 07. 05.

ImageNet Classification with Deep Convolutional Neural Networks (AlexNet)

Krizhevsky et.al. [NIPS 2012]

IDEALAB

Improving lives through learning

JongHyeon Kim
School of Computer Science/Department of AI Convergence Engineering
Gyeongsang National University (GNU)

GNU 경상국립대학교 Gyeongsang National University

Contents

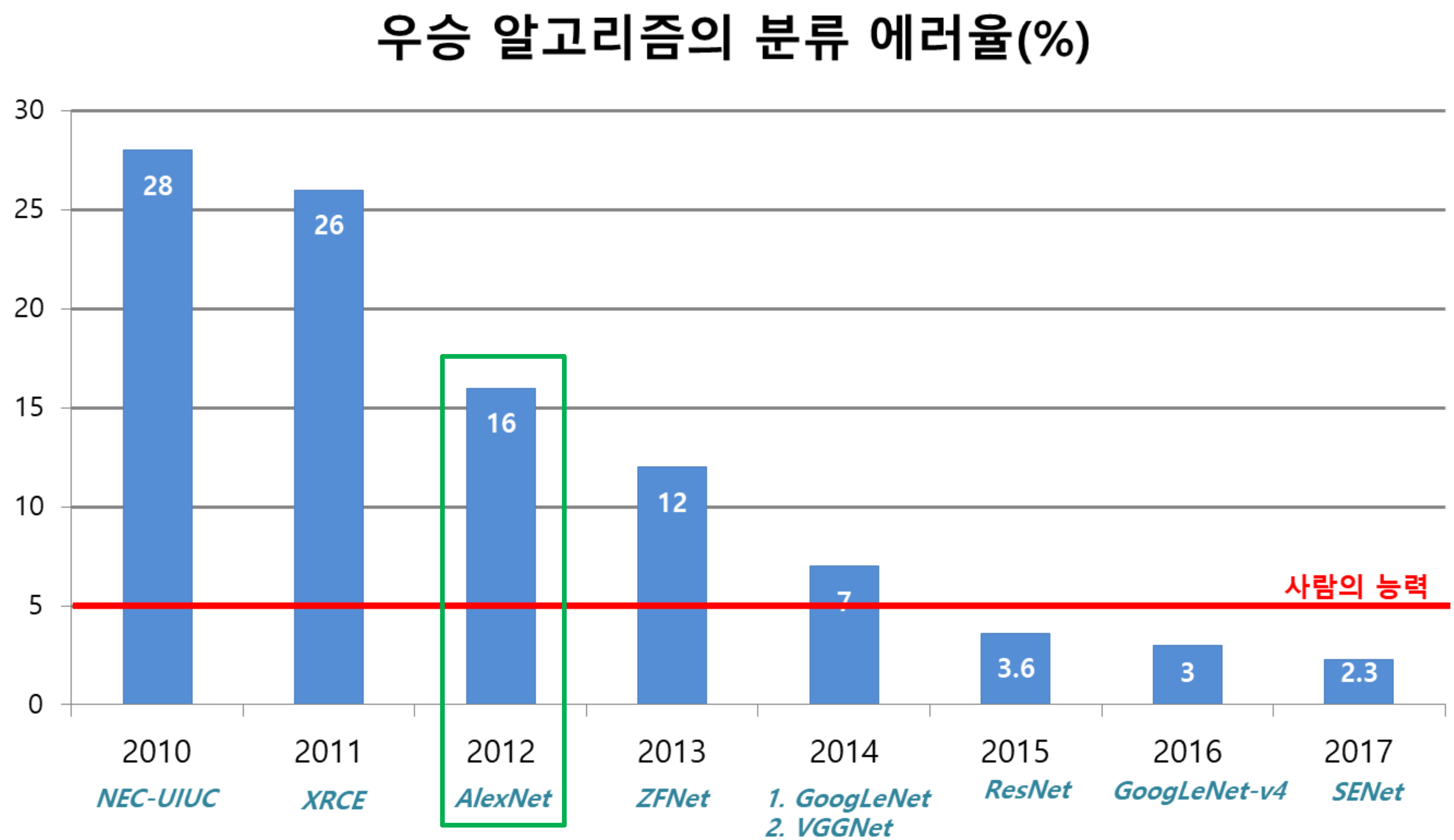
Introduction

- **ImageNet**
  - 15+ million labeled high-resolution images with 22000 categories
  - Metric: top-1, top-5 error rates
    - top-1 error rate: same as accuracy (number of prediction == label)
    - top-5 error rate: if label in 5 highest probability model outputs > True

- **ILSVRC(ImageNet Large-Scale Visual Recognition Challenge)**
  - Roughly 1000 images in each 1000 categories
  - 1.2 million training images | 50k validation images | 150k testing images

Introduction

- Trends of ILSVRC

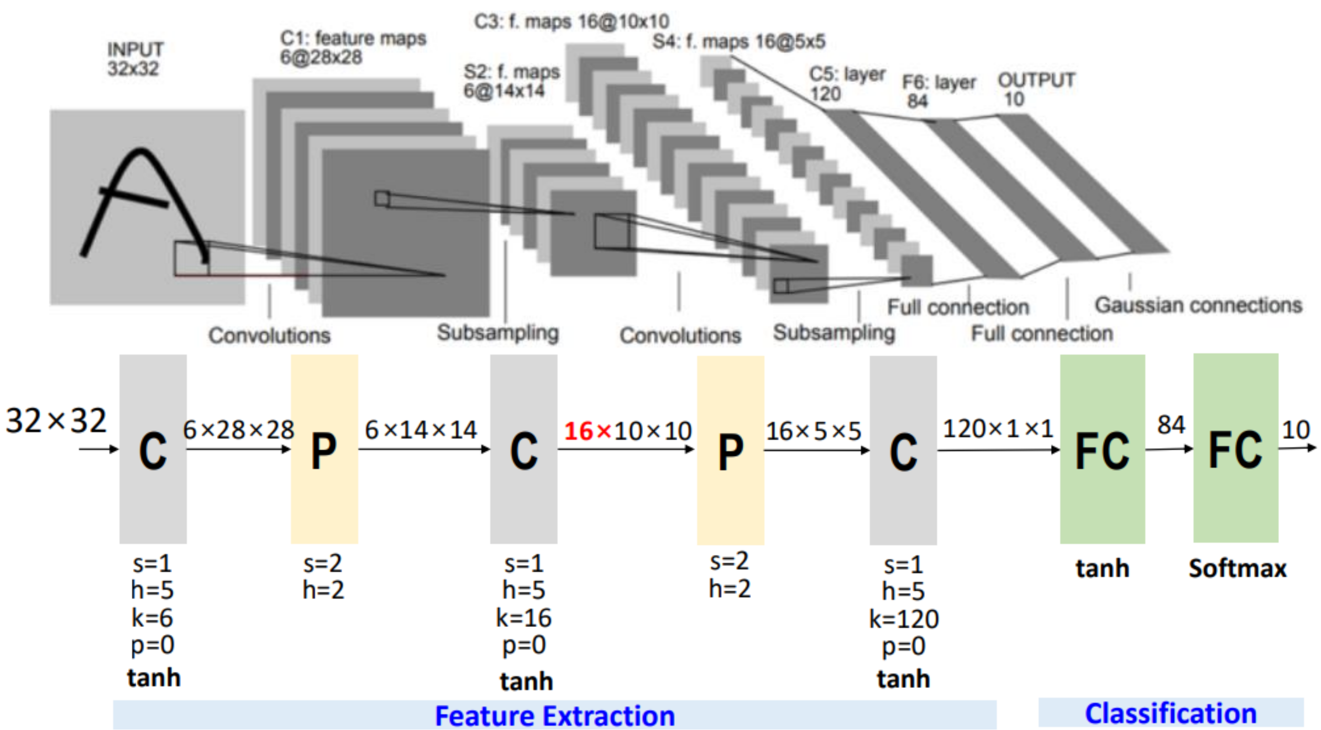


Introduction

- **CNN Architecture**
  - CNN can control model capacity with Depth(how many convolution layers) and Breadth(how many filters)
    - 5 convolutional Layers
    - 3 fully-connected layers
- **Achieved Best Results on ILSVCR-2010 and ILSVCR-2012**
- **5~6 days to train network with two GTX 580 3GB GPUs**
  - Distributed Training

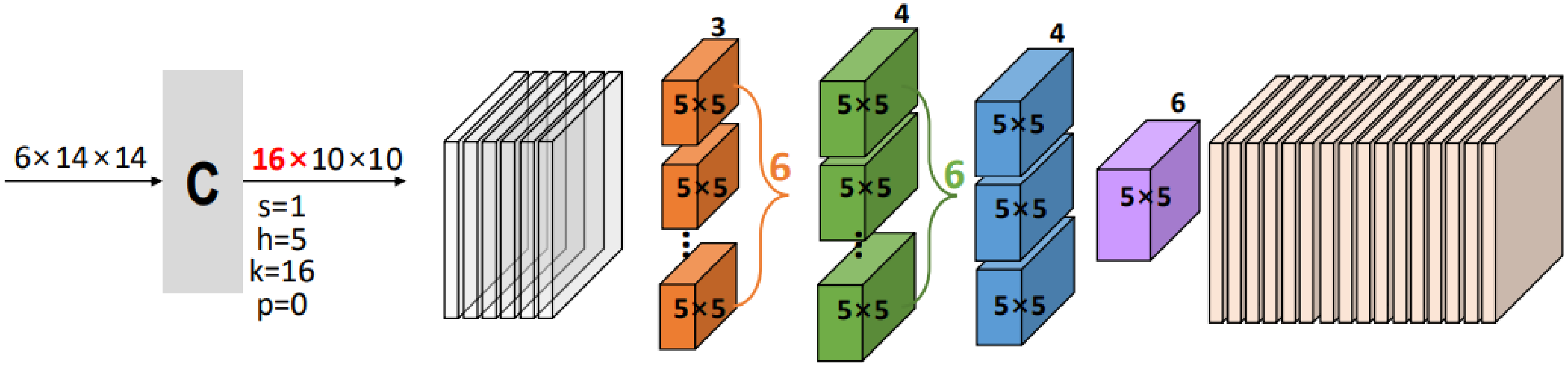
Related Work

- **Gradient-Based Learning Applied to Document Recognition (Lecun et.al. IEEE, 1998.); LeNet-5**
  - To recognize handwritten numbers (0~9)
    - Adapted CNN Architecture (due to limitations of FFNN)
    - 3 Convolutional Layers, 2 Pooling(avg) Layers, 2 Fully-Connected Layers



Related Work

- **Gradient-Based Learning Applied to Document Recognition (Lecun et.al. IEEE, 1998.); LeNet-5**
  - 3rd convolution layer

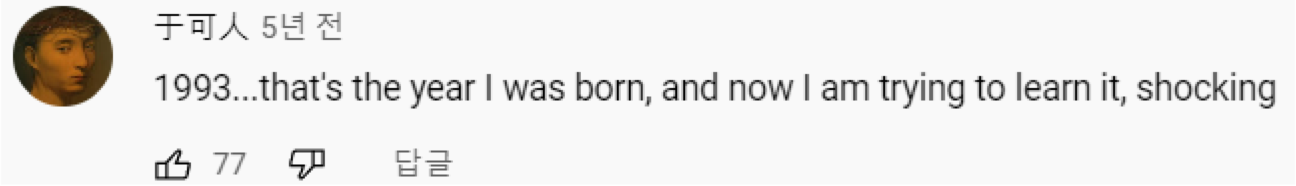


OUT

IN		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0																
	1																
	2																
	3																
	4																
	5																

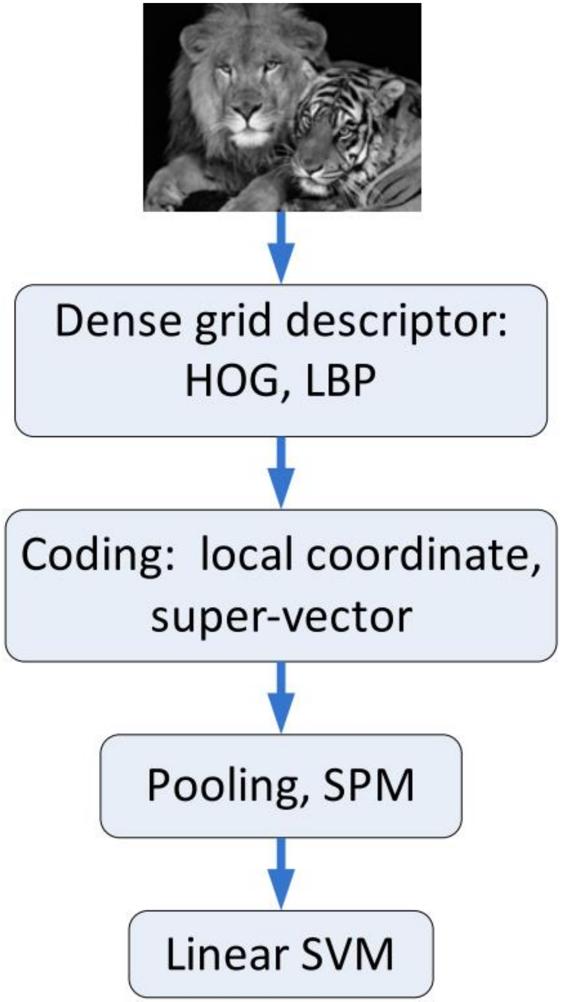
Related Work

- **Gradient-Based Learning Applied to Document Recognition (Lecun et.al. IEEE, 1998.); LeNet-5**



Related Work

- **Large-scale Image Classification: Fast Feature Extraction and SVM Training (Lin et al. CVPR, 2011.)**
  - Winner on ILSVRC 2010
    - Achieved top-5 error rate 28.19%

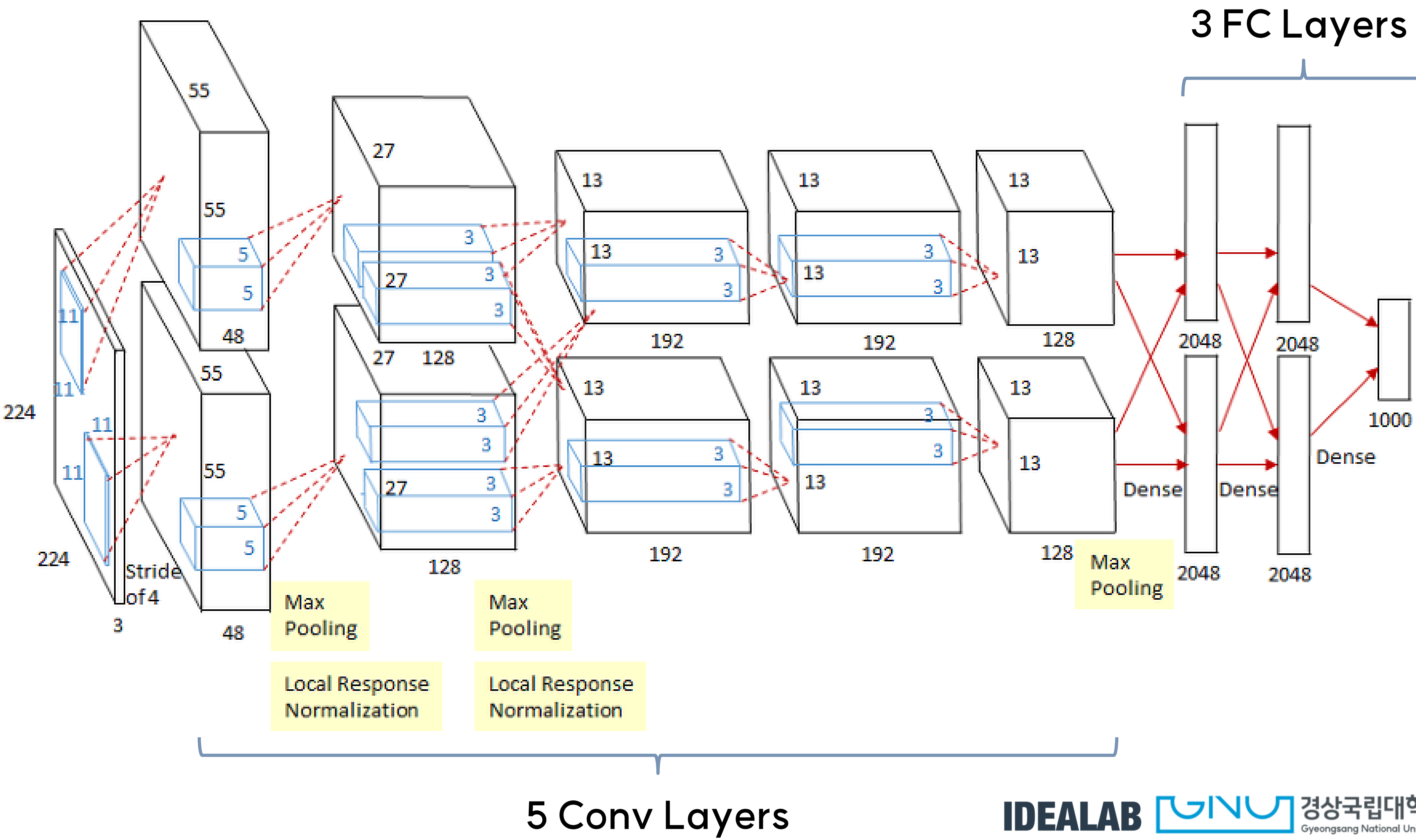


Dataset

- **Preprocessing**
  - For squared image: down-sampled to resolution of 256 x 256
  - For rectangular image
    - First, rescale shorter side to 256
    - Second, center cropped to 256 x 256
  - Subtracted mean value over the training set from each pixel
    - First, calculate mean value for each pixel from all training data (RGB 3 channels)
    - Second, subtract it from corresponding pixel value of each image
    - zero-centering

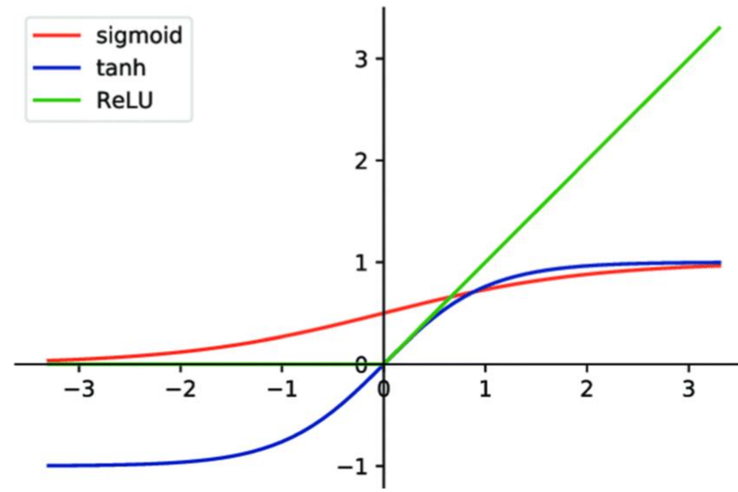
Architecture

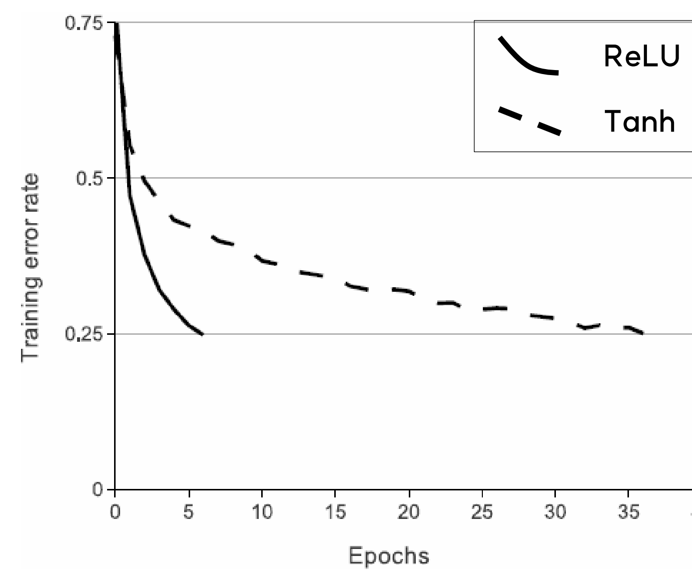
- 5 convolutional layers + 3 fully-connected layers



Details of The Architecture

- **ReLU Nonlinearity**
  - Standard way: saturating nonlinearities
    - Hyperbolic Tangent(Tanh): $\frac{e^x - e^{-x}}{e^x + e^{-x}}$
    - Sigmoid: $\frac{1}{1+e^{-x}}$
    - Vanishing gradient problem
    - Slow convergence
    - Computationally expensive
  - New way: non-saturating nonlinearity
    - Rectified Linear Unit(ReLU): $\max(0, x)$
    - Does not saturate (in the + region)
    - Computationally efficient
  - ReLU is x6 faster than Tanh on CIFAR-10 Dataset



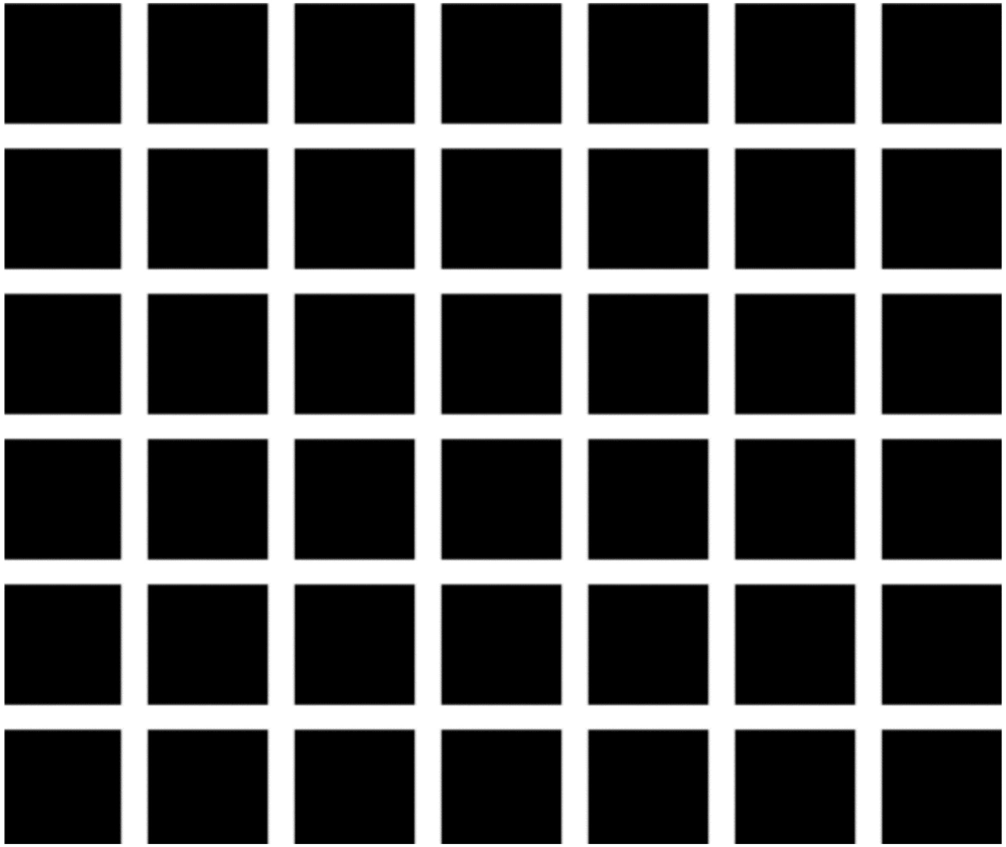


Details of The Architecture

- **Training on Multiple GPUs**
  - Not for training speed, due to model capacity and lots of training data
    - Spread network across two GPUs: puts half of the kernels(neurons) on each GPU
  - One Additional Trick: the GPUs communicate only in certain layers
    - 2^{nd}, 4^{th}, 5^{th} convolutional layers are trained on each distributed GPU
    - 3^{rd} convolutional layer, fully-connected layers are trained by gathering previous layer outputs (GPU-0 output + GPU-1 output)
  - Reduced top-1 error rate by 1.7%, top-5 error rate by 1.2%
  - Two-GPU net slightly less time to train than one-GPU net

Details of The Architecture

- **Local Response Normalization**
  - Motivated from lateral inhibition
    - the capacity of an excited neuron to reduce the activity of its neighbors

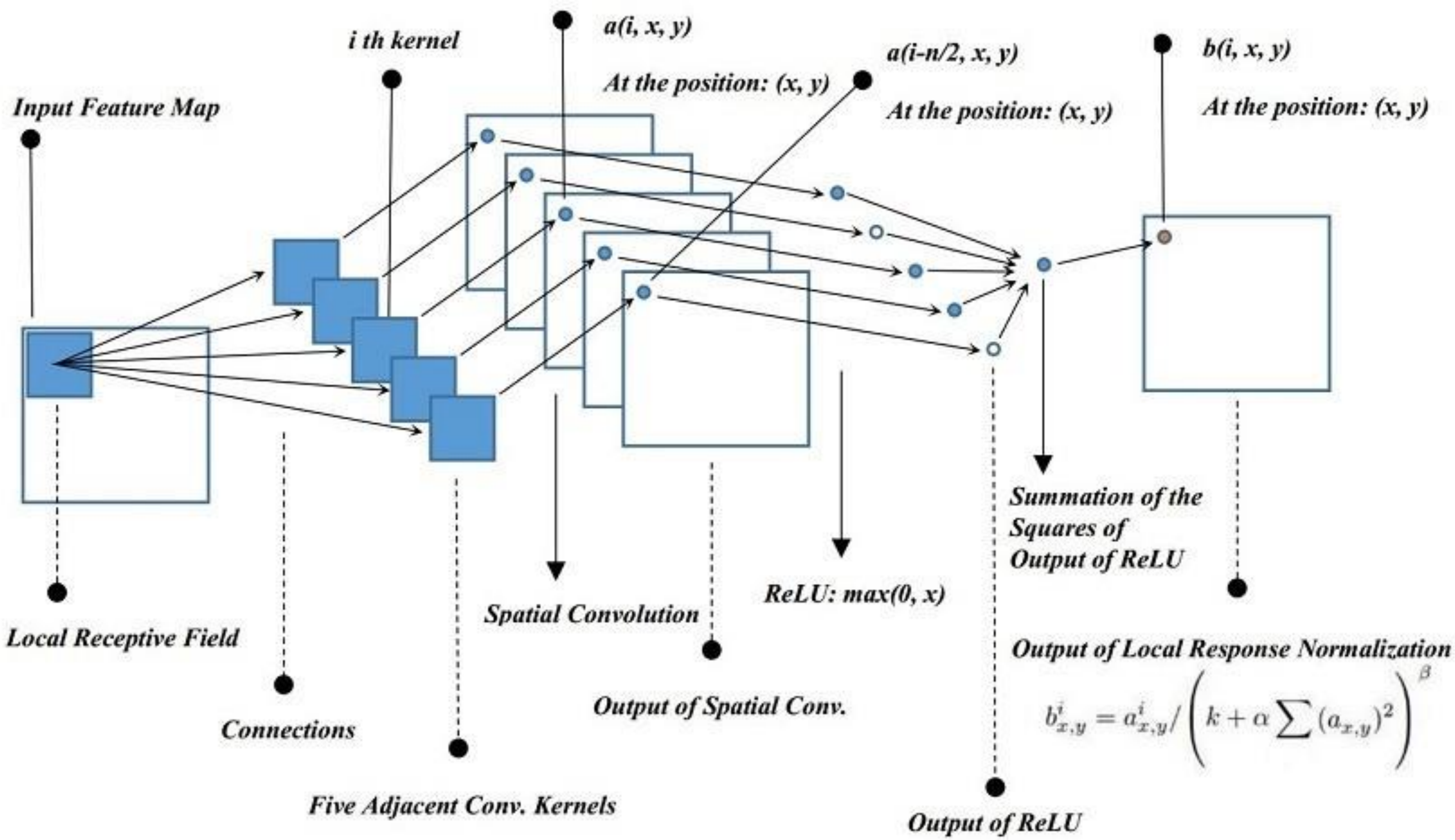


Details of The Architecture

- **Local Response Normalization**
  - ReLU does not require input normalization, but Adapted for generalization
    - excited neuron == large positive value; to refrain strong stimulus to near pixels
    - $$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j = \max(0, i - n/2)}^{min(N-1, i+n/2)} (a^j_{x,y})^2 \right)^\beta$$
    - Sum over n adjacent kernel maps at the same spatial position (x, y)
    - k, n, α, β (hyper-parameter): $k = 2$, $n = 5$, $\alpha = 10^{-4}$, $\beta = 0.75$
    - Apply after 1st, 2nd ReLU layers
    - Reduced top-1 error rate by 1.4%, top-5 error rate by 1.2%

Details of The Architecture

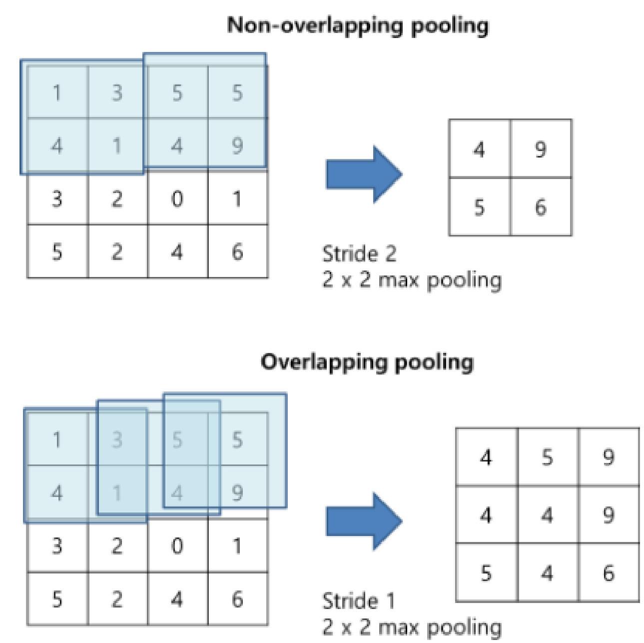
- **Local Response Normalization**



Details of The Architecture

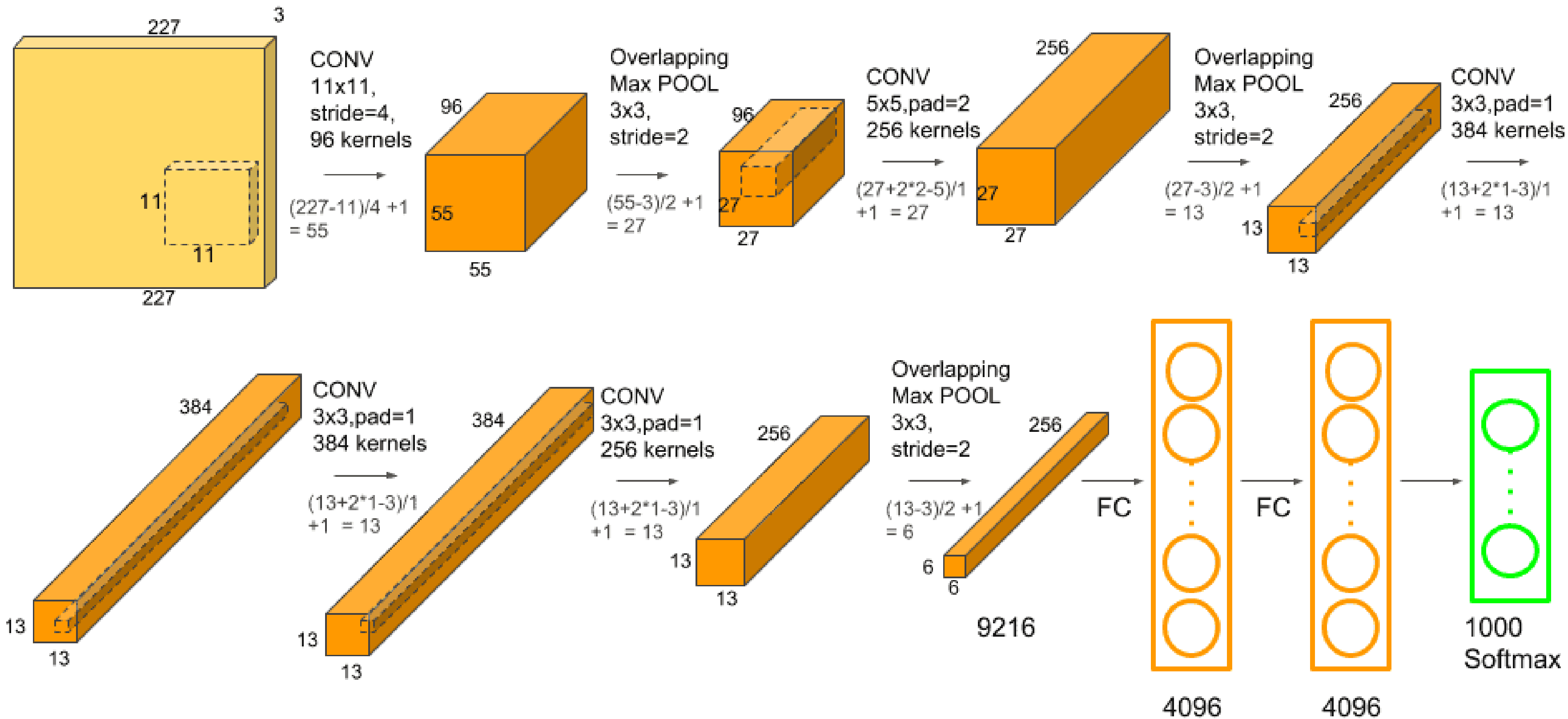
- **Overlapping Pooling**
  - At max pooling, adjust kernel size and stride
    - kernel size == stride: traditional local pooling
    - kernel size > stride: overlapping pooling

      – used kernel size = 3, stride = 2
    - Reduced top-1 error rate by 0.4%, top-5 error rate by 0.3%
  - Overlapping pooling prevents overfit



Details of The Architecture

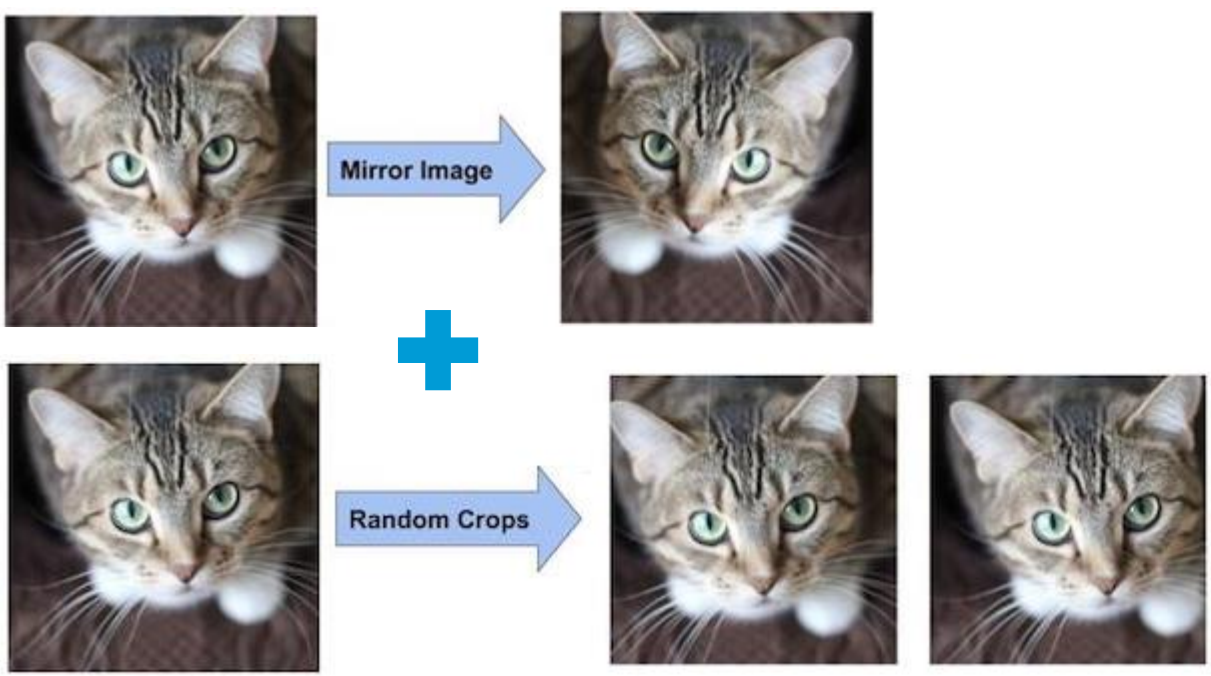
- **Overall Architecture**



Reducing Overfitting

- **Data Augmentation**
  - Image translation and horizontal reflection
    - Train Data: Random 224 x 224 patches (RandomCrop) + horizontal reflections from the preprocessed images (256 x 256)

      – increased training set by a factor of 2048
    - Test Data: averaging the ten Softmax probabilities (five 224 x 224 patches (4 corner + center patches) + horizontal reflections)



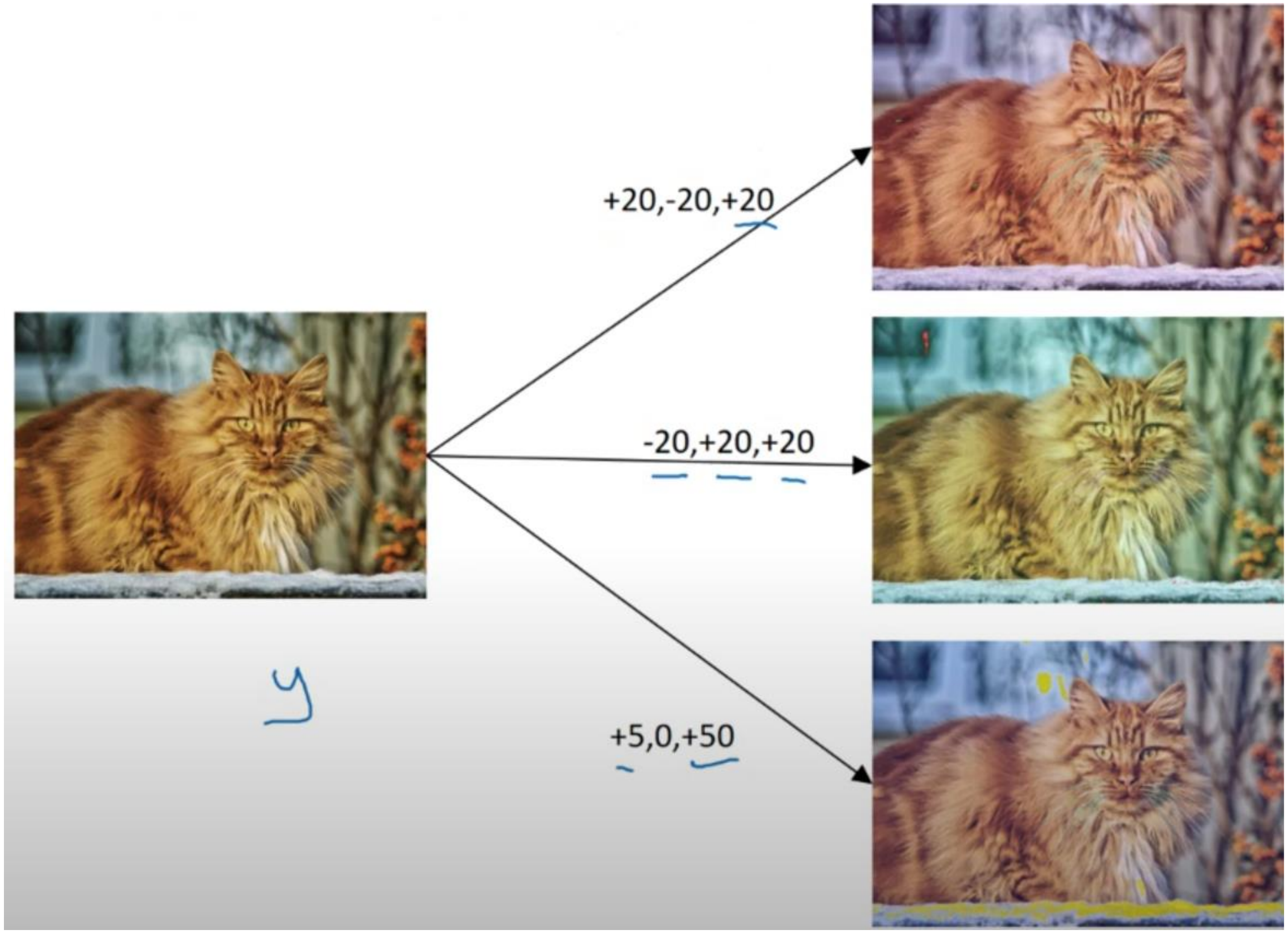
Reducing Overfitting

- **Data Augmentation**
  - Change the intensity of RGB channels (ColorJitter)
    - PCA on the set of RGB pixel values throughout training set

      – to obtain eigenvector, eigenvalue of training set
    - $I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B]$: RGB image pixel
    - expr: $[p_1, p_2, p_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$; p_i: ith eigenvector, λ_i: ith eigenvalue, α_i: random variable (Gaussian Distribution, mean=0.0, std=0.1)
    - Add expr to each training image
  - Reduces the top-1 error rate by over 1%

Reducing Overfitting

- **Data Augmentation**
  - Change the intensity of RGB channels (ColorJitter)

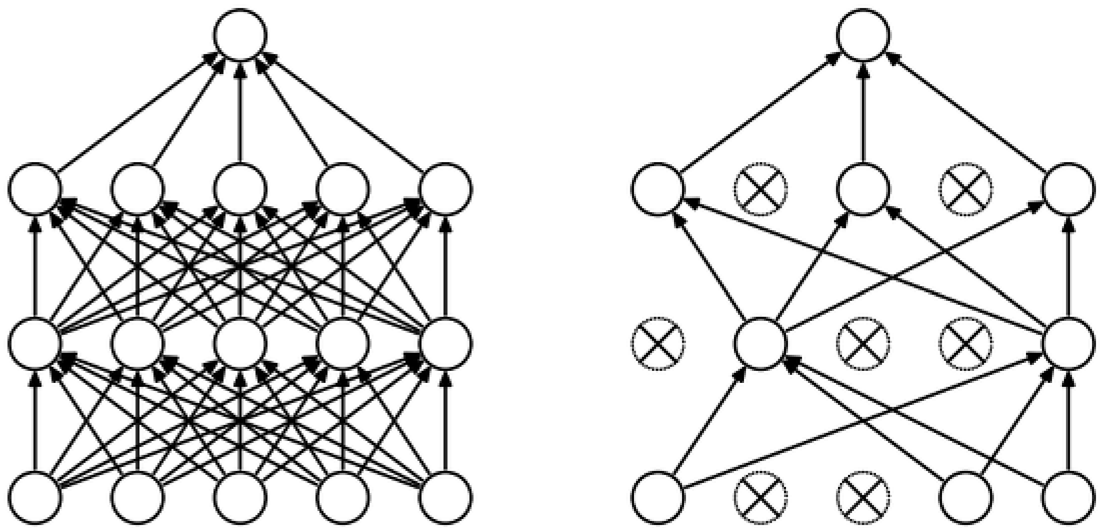


Reducing Overfitting

- **Dropout**
  - Train model with partly dropped input
    - Ensemble effect: Dropout is based on Bernoulli distribution
      - randomly zeroes some of elements of the input(activation outputs) on every mini-batch
    - Prevents Co-adaptation: Reduce dependencies to some neurons
      - e.g.) if neurons receive “bad” inputs, then the dependent neurons can be affected as well
      - affects to model performance > overfitting
  - Dropout should only work during training time

Reducing Overfitting

- **Dropout**



(a) Standard Neural Net (b) After applying dropout.

Input for Present Layer = Previous Layer Output

```
CLASS torch.nn.Dropout(p=0.5, inplace=False) [SOURCE]
```

During training, randomly zeroes some of the elements of the input tensor with probability p using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Details of Learning

- **Optimizer: SGD (momentum=0.9, weight decay=5e-4) with batch Size 128**

- **Parameter initialization**
  - All weights: zero-mean Gaussian distribution with standard deviation 0.01
  - 2^{nd}, 4^{th}, 5^{th} convolutional layers and fully-connected layers biases: 1
    - Accelerate activation function(ReLU): positive examples
  - Remaining layers biases: 0

- **Learning Rate**
  - Initial lr: 1e-2
  - Divided by 10 when validation error stopped improving

- **Trained 90 cycles(epochs) for 5~6 days**

Results

- ILSVRC-2010 (Test Error)

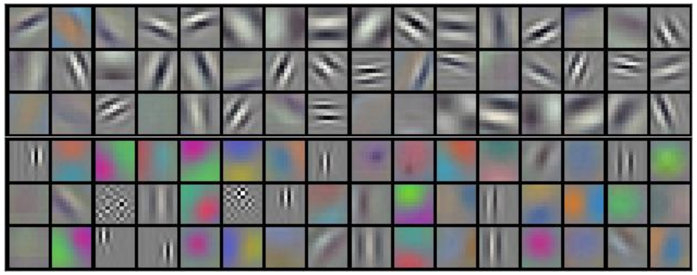
Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

- ILSVRC-2012

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

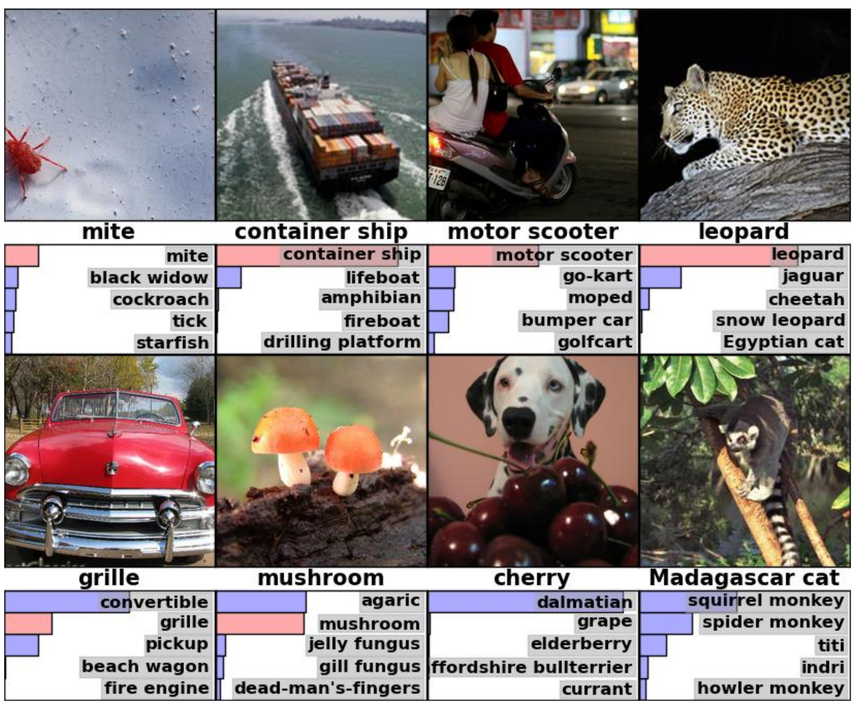
  - 1 CNN: proposed model
  - 5 CNNs: ensemble 5 similar CNNs
  - 1 CNN*: finetuned 1 model (proposed model + extra 6th convolutional layer + pretrain with ILSVRC-2011 dataset)
  - 7 CNNs*: finetuned 2 models(proposed model + pretrained with ILSVRC-2011 dataset) + 5CNNs

Appendix

- Each GPU learns different features
  - 

    GPU 0: color-agnostic features

    GPU 1: color-specific features

- Top-5 predictions (8 samples)
  - 

Appendix

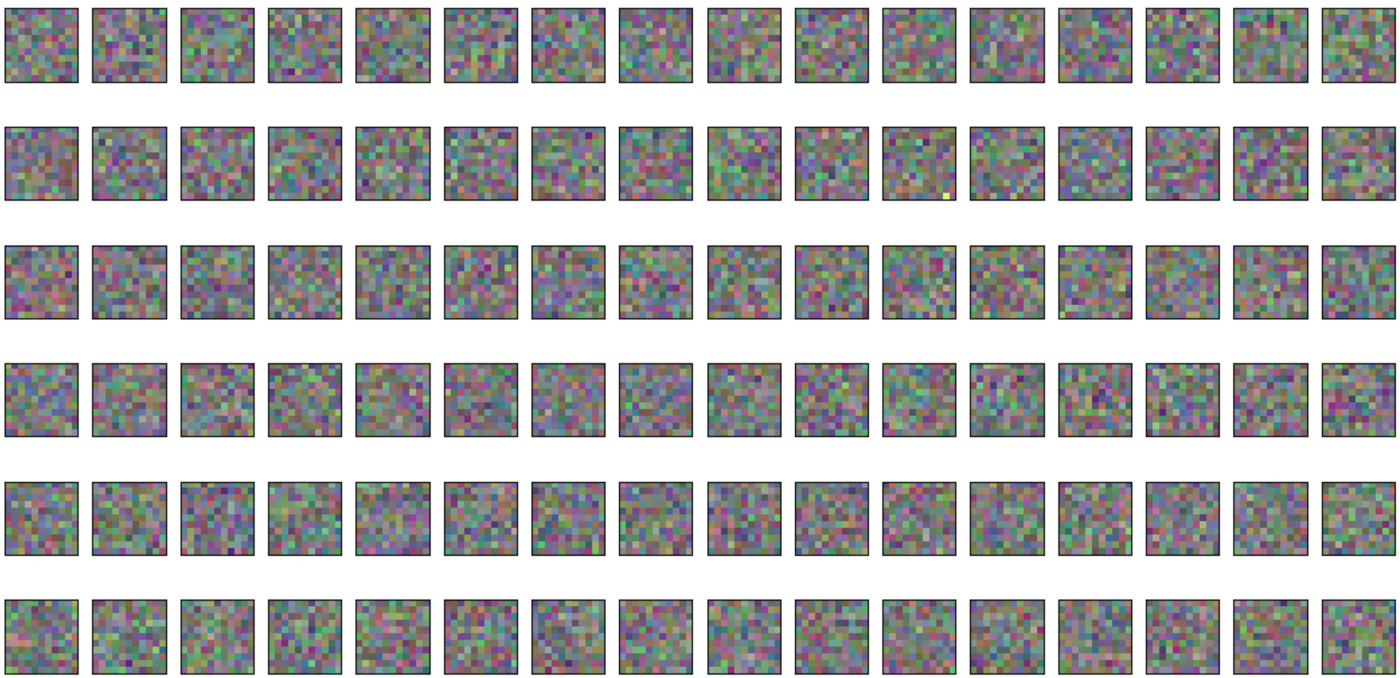
- **Compute similarity by Euclidean distance**
  - 
    - Calculate L2 distance between last fully-connected layer (before classifier, 4096-dim) and train data
    - Can reduce computational complexity by training autoencoder with fully-connected layer (4096-dim)

Conclusion

- **Depth is important for achieving results**
  - removing any of the middle layer results in a loss of about 2% for the top-1 performance of the network
- **Unsupervised pre-training expected to helpful, but did not use**
  - Limitation of computational power
- **Much deeper network, more training time improved the results**

Discussion

- **Each GPU learns different features**
  - How does it work? > is it reliable?



Epoch 0

Code Implementation

- Convolution Layers

```
self.conv = nn.Sequential(
        nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11, stride=4),
        # (bs, 3, 227, 227) > (bs, 96, 55, 55)
        nn.ReLU(),
        nn.LocalResponseNorm(size=5, alpha=1e-4, beta=0.75, k=2),
        nn.MaxPool2d(kernel_size=3, stride=2),  # (bs, 96, 55, 55) > (bs, 96, 27, 27)
        nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5, stride=1,
                        padding=2),  # (bs, 96, 27, 27) > (bs, 256, 27, 27)
        nn.ReLU(),
        nn.LocalResponseNorm(size=5, alpha=1e-4, beta=0.75, k=2),
        nn.MaxPool2d(kernel_size=3, stride=2),  # (bs, 256, 27, 27) > (bs, 256, 13, 13)
        nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3, stride=1,
                       padding=1),  # (bs, 256, 13, 13) > (bs, 384, 13, 13)
        nn.ReLU(),
        nn.Conv2d(in_channels=384, out_channels=384, kernel_size=3, stride=1,
                       padding=1),  # (bs, 384, 13, 13) > (bs, 384, 13, 13)
        nn.ReLU(),
        nn.Conv2d(in_channels=384, out_channels=256, kernel_size=3, stride=1,
                       padding=1),  # (bs, 384, 13, 13), (bs, 256, 13, 13)
        nn.MaxPool2d(kernel_size=3, stride=2),  # (bs, 256, 13, 13) > (bs, 256, 6, 6)
    )
```

Code Implementation

- Fully-Connected Layers

```
self.fc = nn.Sequential(
        nn.Flatten(),  # same as x.view(-1, 256 * 6 * 6)
        nn.Dropout(p=0.5),
        nn.Linear(in_features=(256 * 6 * 6), out_features=4096),
        nn.ReLU(),
        nn.Dropout(p=0.5),
        nn.Linear(in_features=4096, out_features=4096),
        nn.ReLU(),
        nn.Linear(in_features=4096, out_features=self.num_features),
    )
```

Code Implementation

- Weight Initialization

```python
def init_weights(self):
    for layer_num, layer in enumerate(self.conv):
        if isinstance(layer, nn.Conv2d):
            nn.init.normal_(layer.weight, mean=0.0, std=0.01),
            nn.init.zeros_(layer.bias)
            if (
                layer_num == 4 or layer_num == 10 or
                layer_num == 12
            ):  # second, fourth, fifth conv layers
                nn.init.ones_(layer.bias)
    for layer in self.fc:
        if isinstance(layer, nn.Linear):
            nn.init.normal_(layer.weight, mean=0.0, std=0.01),
            nn.init.ones_(layer.bias)
```

Code Implementation

- Model Summary

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 55, 55]          34,944
              ReLU-2           [-1, 96, 55, 55]               0
 LocalResponseNorm-3           [-1, 96, 55, 55]               0
         MaxPool2d-4           [-1, 96, 27, 27]               0
            Conv2d-5          [-1, 256, 27, 27]         614,656
              ReLU-6          [-1, 256, 27, 27]               0
 LocalResponseNorm-7          [-1, 256, 27, 27]               0
         MaxPool2d-8          [-1, 256, 13, 13]               0
            Conv2d-9          [-1, 384, 13, 13]         885,120
             ReLU-10          [-1, 384, 13, 13]               0
           Conv2d-11          [-1, 384, 13, 13]       1,327,488
             ReLU-12          [-1, 384, 13, 13]               0
           Conv2d-13          [-1, 256, 13, 13]         884,992
        MaxPool2d-14            [-1, 256, 6, 6]               0
          Flatten-15                 [-1, 9216]               0
          Dropout-16                 [-1, 9216]               0
           Linear-17                 [-1, 4096]      37,752,832
             ReLU-18                 [-1, 4096]               0
          Dropout-19                 [-1, 4096]               0
           Linear-20                 [-1, 4096]      16,781,312
             ReLU-21                 [-1, 4096]               0
           Linear-22                 [-1, 1000]       4,097,000
================================================================
Total params: 62,378,344
Trainable params: 62,378,344
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.59
Forward/backward pass size (MB): 14.47
Params size (MB): 237.95
Estimated Total Size (MB): 253.01
----------------------------------------------------------------
```

경상국립대학교
Gyeongsang National University

Improving lives through learning

IDEALAB