Convolutional Neural Networks 2

Xiaolong Wang

This Class

- Regularization in Training Deep Networks
- Development of ConvNets
- Data Augmentation, Batch Normalization

Batch Normalization

Batch Normalization

- Explicitly enforce each layer to have zero-mean and unit-variance outputs

- A basic version of batch norm:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

Why is it important to maintain the magnitude of activations?

$$\frac{\partial e}{\partial W_k} = \frac{\partial e}{\partial h_k} \boxed{\frac{\partial h_k}{\partial W_k}}$$ activations

$$\frac{\partial e}{\partial h_{k-1}} = \frac{\partial e}{\partial h_k} \frac{\partial h_k}{\partial h_{k-1}}$$

Compute $\frac{\partial h_k}{\partial W_k}$

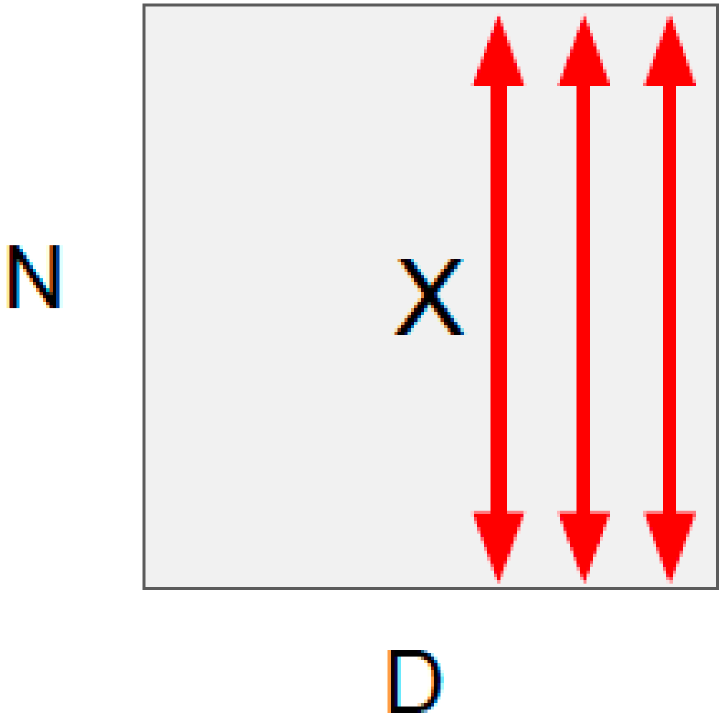
Layer k

Compute $\frac{\partial h_k}{\partial h_{k-1}}$

$$\frac{\partial e}{\partial h_k}$$

Batch Normalization for FC layer

Input: $x \in \mathbb{R}^{N \times D}$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Compute mean for each channel $\mu \in \mathbb{R}^D$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2$$

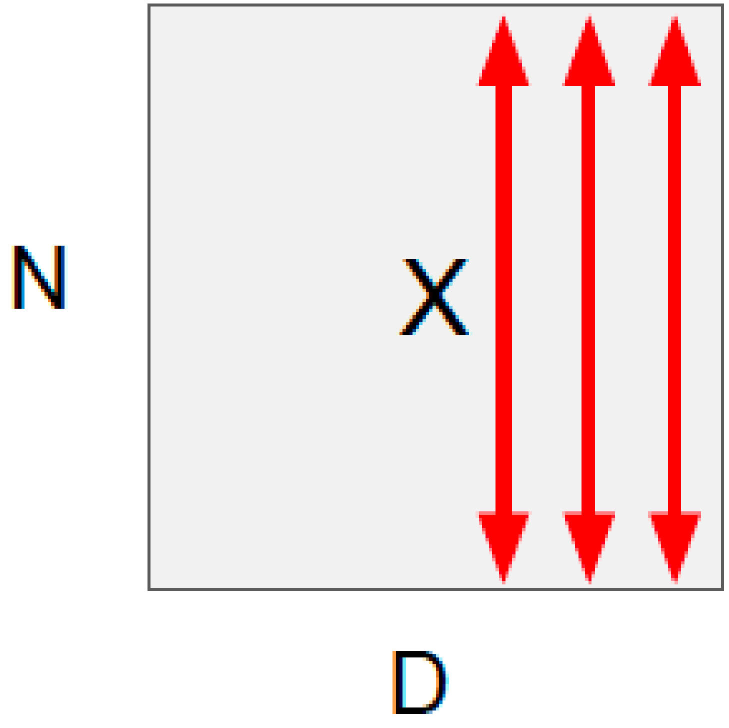
Compute variance for each channel $\sigma^2 \in \mathbb{R}^D$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalize $x \in \mathbb{R}^{N \times D}$

Batch Normalization for FC layer

Input: $x \in \mathbb{R}^{N \times D}$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Compute mean for each channel $\mu \in \mathbb{R}^{D}$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2$$

Compute variance for each channel $\sigma^2 \in \mathbb{R}^{D}$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

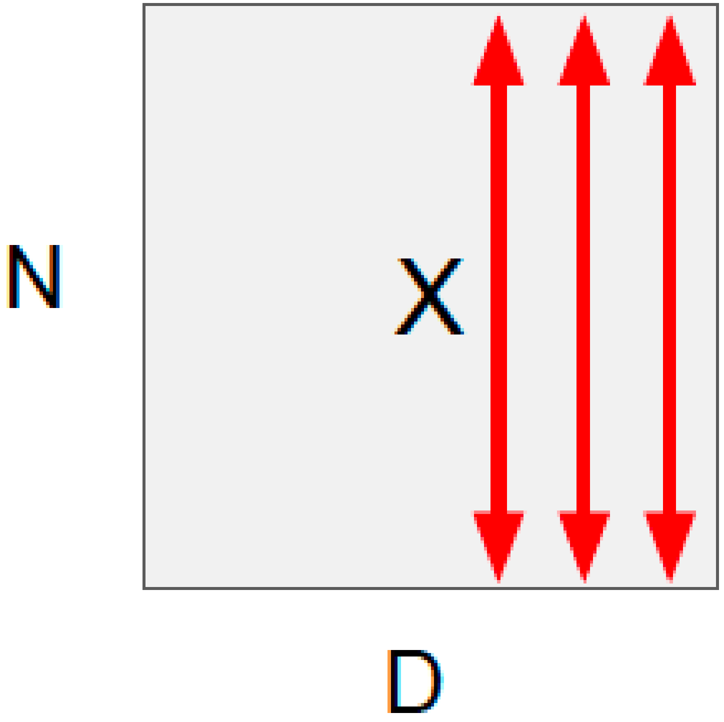
Normalize $x \in \mathbb{R}^{N \times D}$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Scale with learnable parameters $\gamma \in \mathbb{R}^{D}, \beta \in \mathbb{R}^{D}$

During Test Time

Input: $x \in \mathbb{R}^{N \times D}$



$$\cancel{\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}}$$

A running average of μ during training

$$\cancel{\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2}$$

A running average of σ^2 during training

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

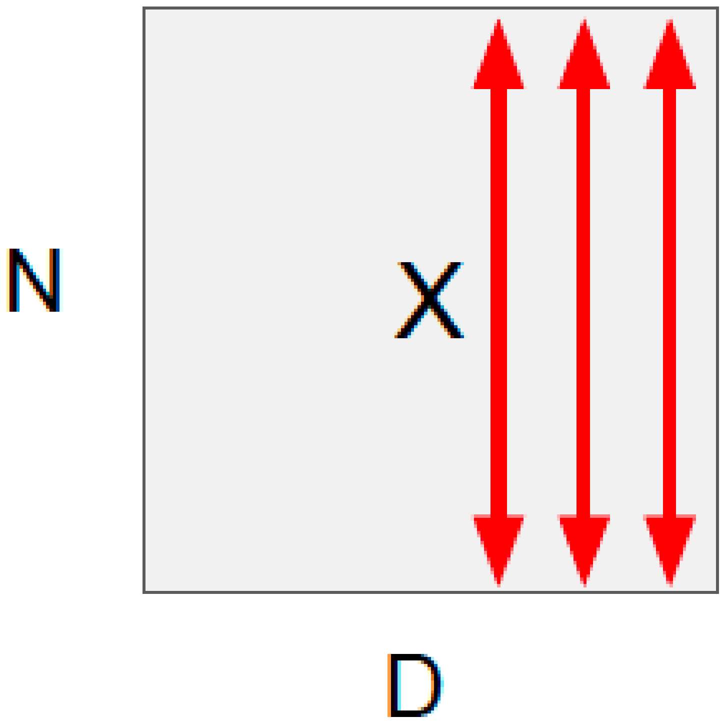
Normalize $x \in \mathbb{R}^{N \times D}$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Scale with learnable parameters $\gamma \in \mathbb{R}^D, \beta \in \mathbb{R}^D$

During Test Time

Input: $x \in \mathbb{R}^{N \times D}$



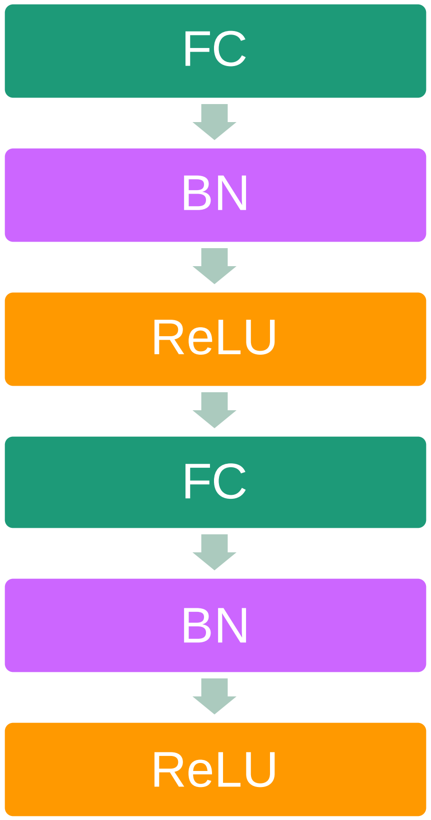
A running average of μ during training:

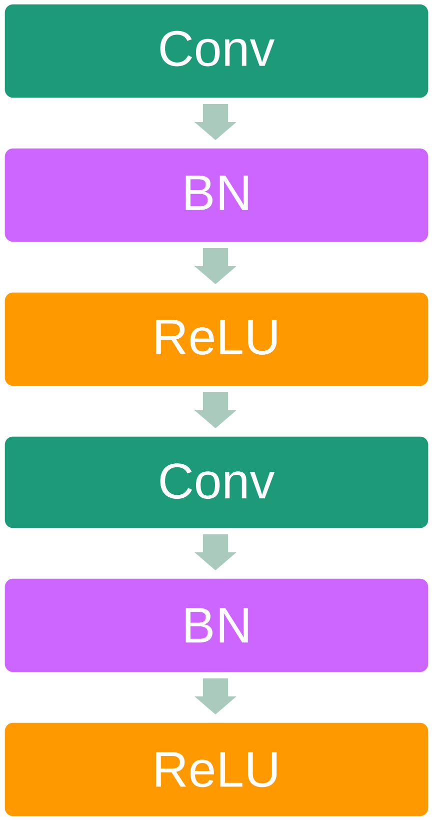
$$\hat{\mu}_t = \alpha \hat{\mu}_{t-1} + (1-\alpha)\mu_{t-1}$$

A running average of σ^2 during training:

$$\hat{\sigma}_t^2 = \alpha \hat{\sigma}_{t-1}^2 + (1-\alpha)\sigma_{t-1}^2$$

Batch Normalization in Deep Networks





Batch Normalization for ConvNets

MLPs

$\mathbf{x}: N \times D$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times D$

$\gamma, \beta: 1 \times D$

$\mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \beta$

ConvNets

$\mathbf{x}: N \times C \times H \times W$

Normalize

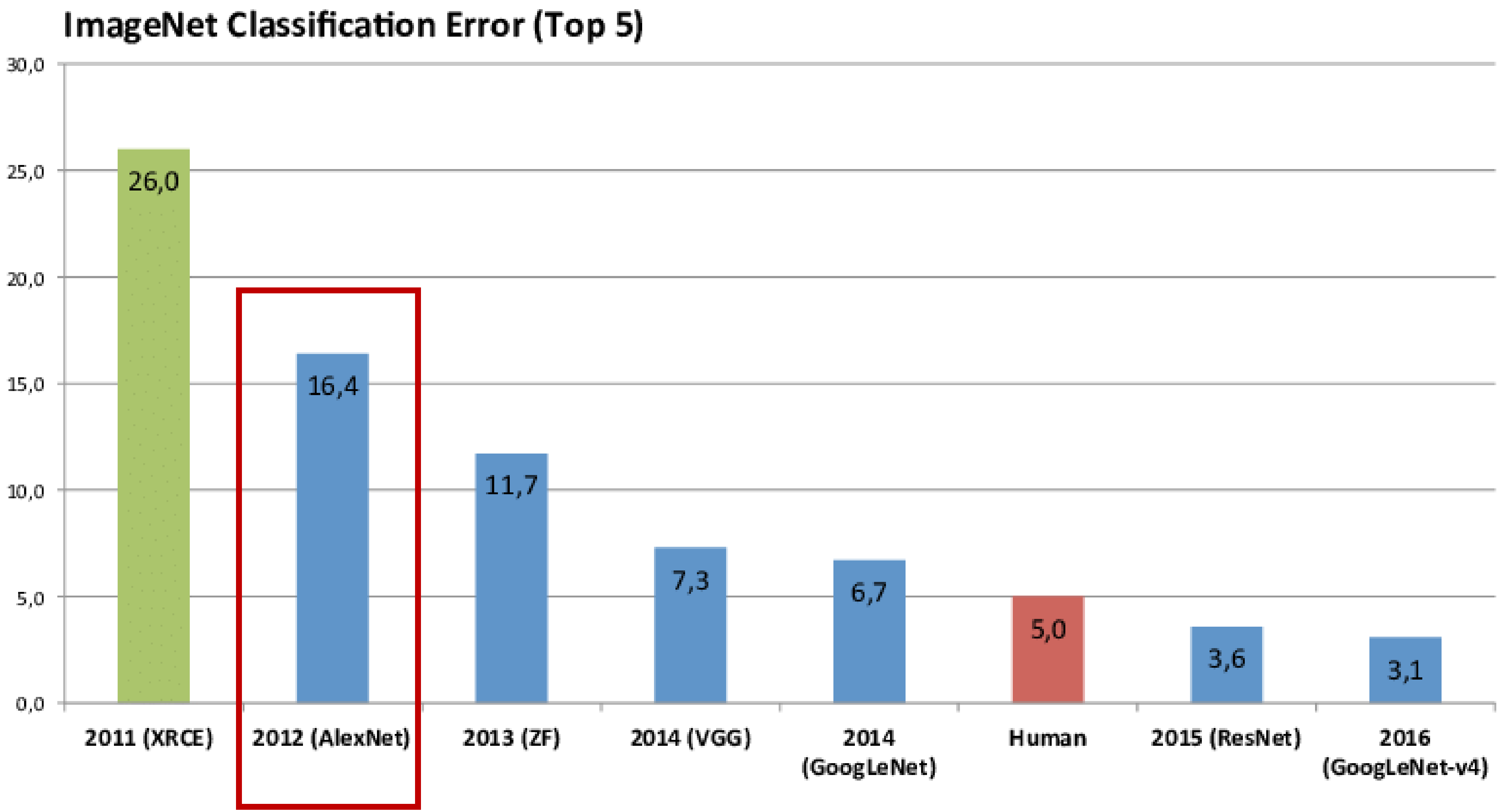
$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

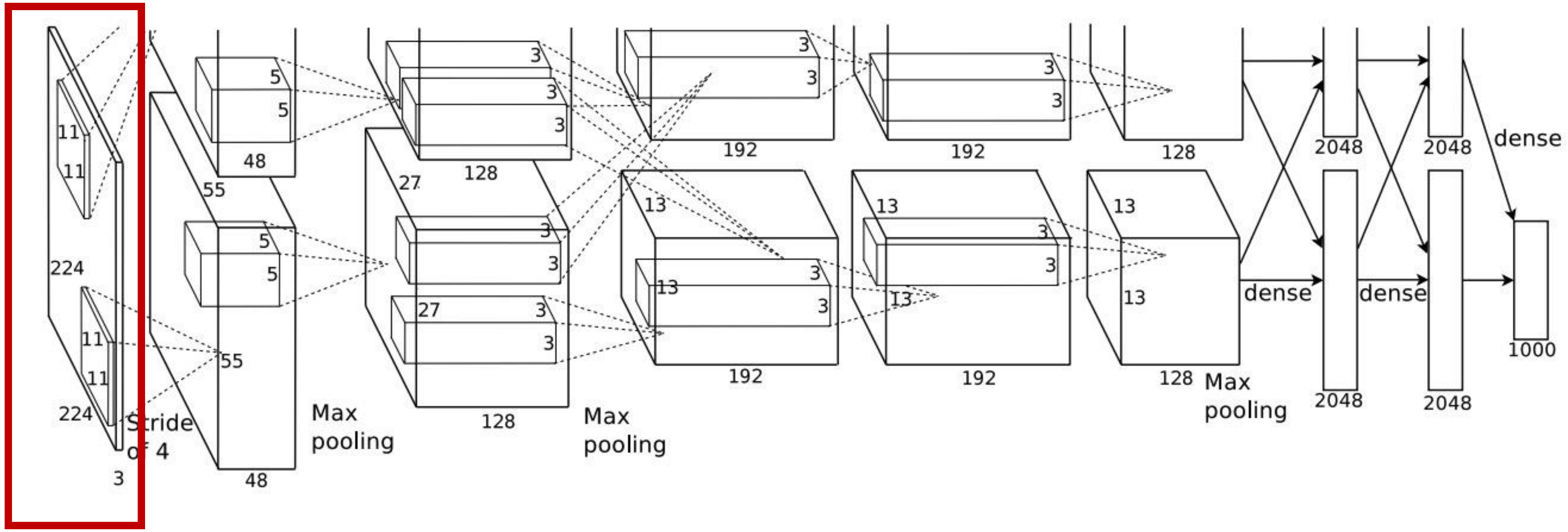
$\mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \beta$

CNN Architectures

ImageNet Performance



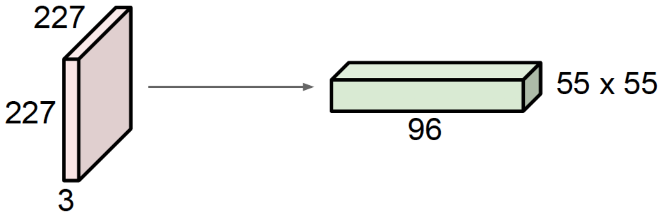
AlexNet



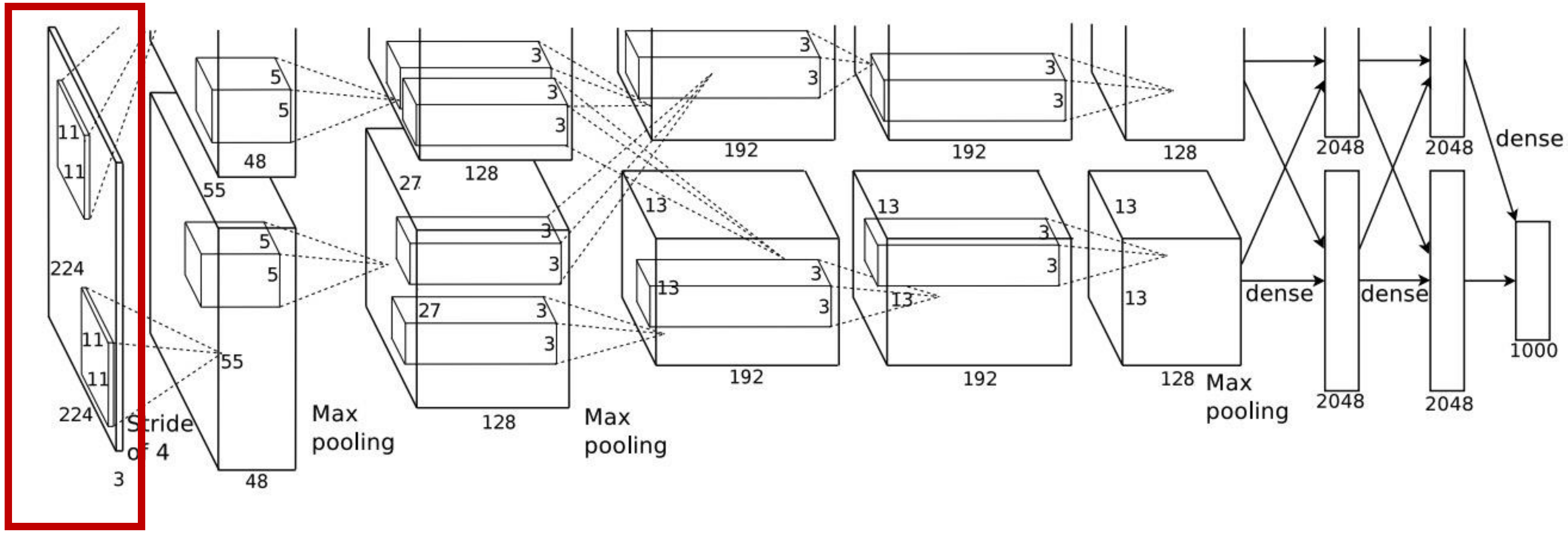
Conv1 -> Maxpool -> Conv2 -> Maxpool -> Conv3 -> Conv4 -> Conv5 -> Maxpool -> FC6 -> FC7 -> FC8

- Input: 227 x 227 x 3 image

- First layer (Conv1): 96 11x11 filters applied at stride 4
  - Output size of first layer: (227 - 11) / 4 + 1 = 55

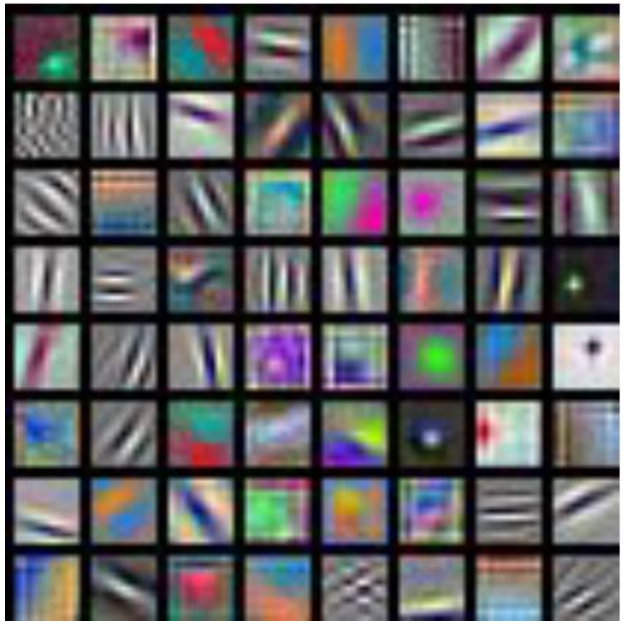


AlexNet

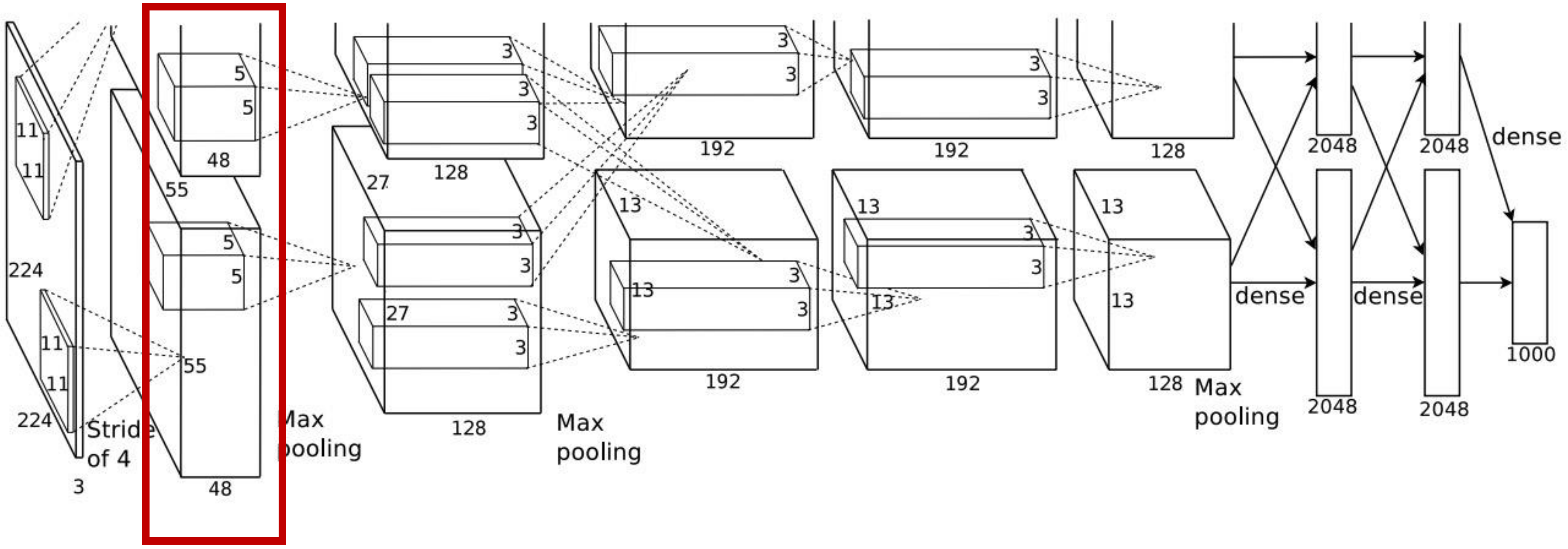


Conv1 -> Maxpool -> Conv2 -> Maxpool -> Conv3 -> Conv4 -> Conv5 -> Maxpool -> FC6 -> FC7 -> FC8

- Learned filters for Conv1

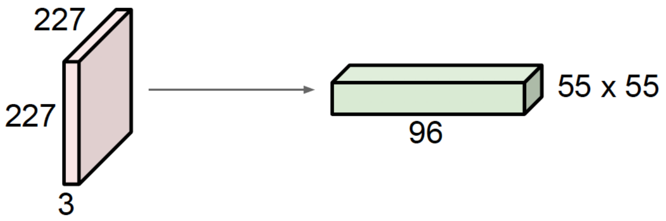


AlexNet



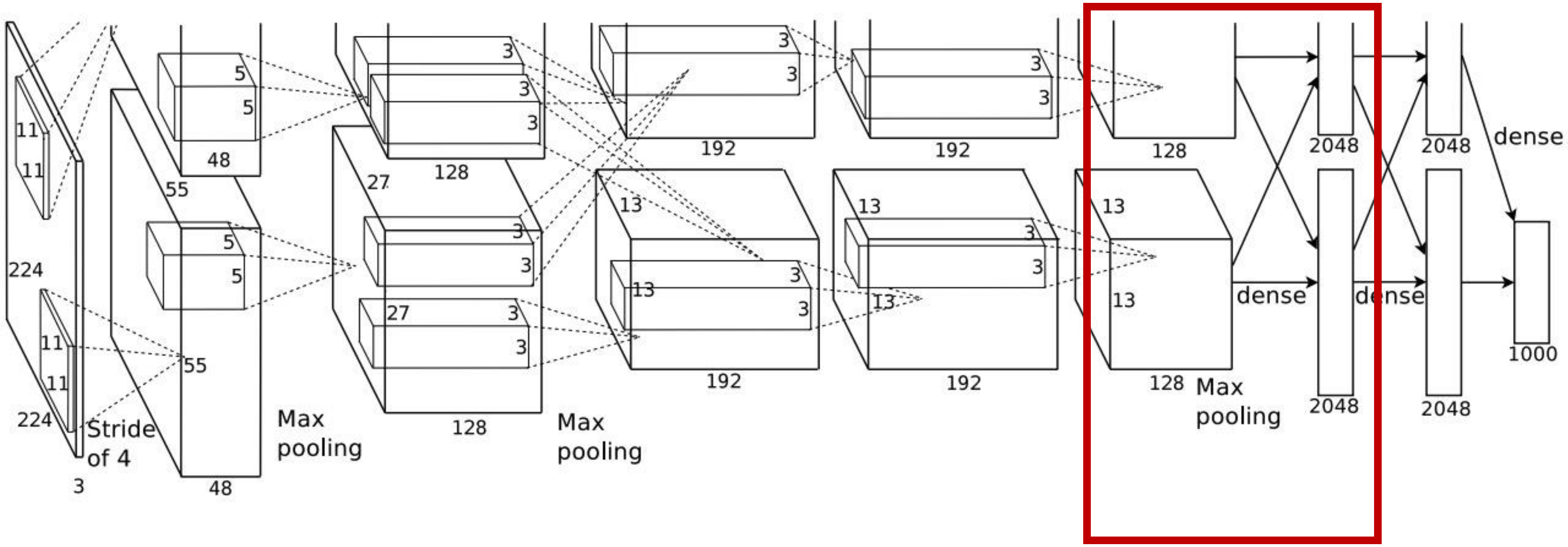
Conv1 -> Maxpool -> Conv2 -> Maxpool -> Conv3 -> Conv4 -> Conv5 -> Maxpool -> FC6 -> FC7 -> FC8

- Input: 55 x 55 x 96 feature map



- Second layer (Maxpool): 3 x 3 filters applied at stride 2
  - Output size of second layer: (55 - 3) / 2 + 1 = 27

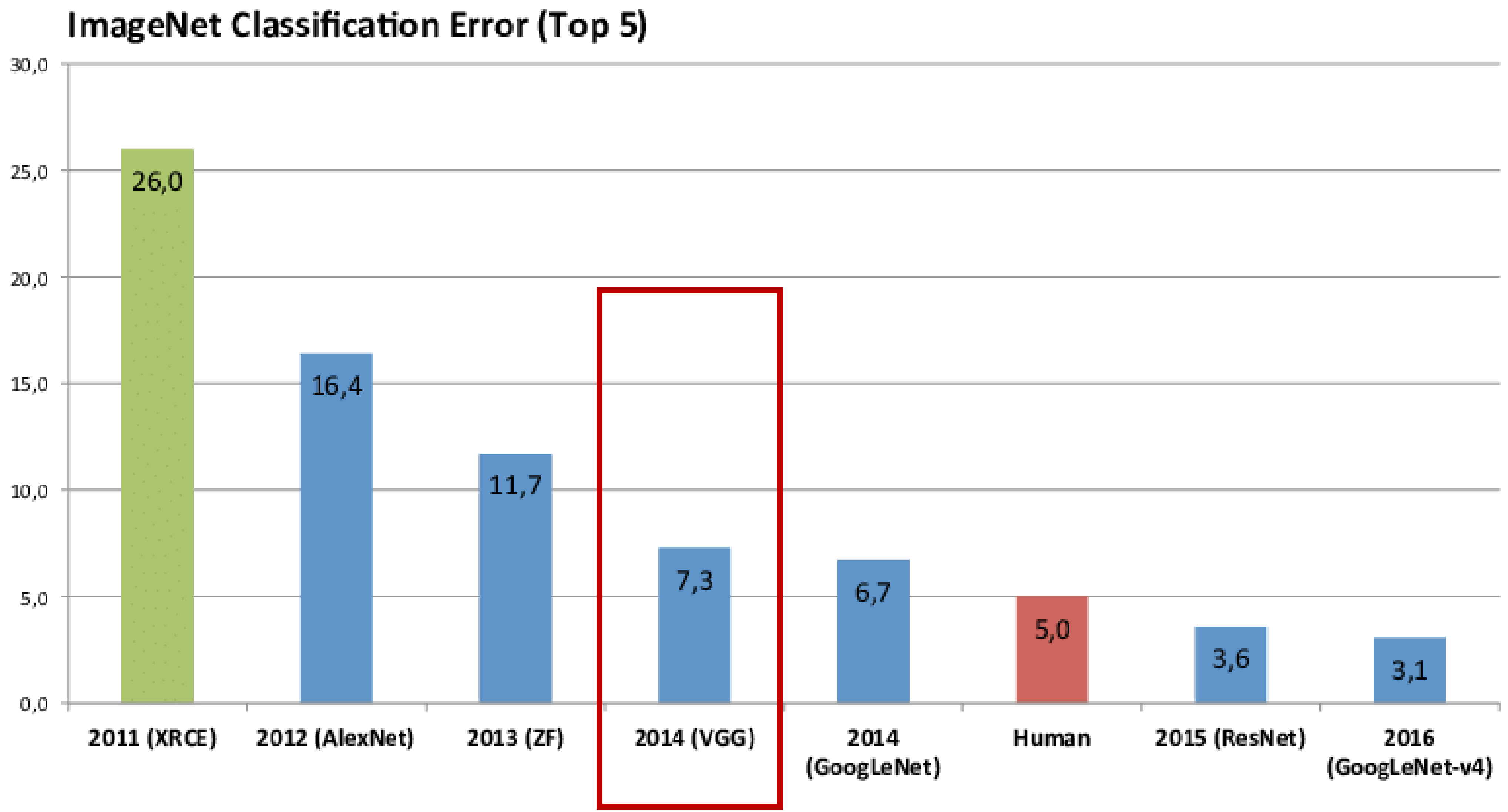
AlexNet



Conv1 -> Maxpool -> Conv2 -> Maxpool -> Conv3 -> Conv4 -> Conv5 -> Maxpool -> FC6 -> FC7 -> FC8

- Input for FC6: 6 x 6 x 256 feature map
- Output for FC6: 4096. Since the layer is fully-connected, the number of parameter is: 6 x 6 x 256 x 4096 = 38 million

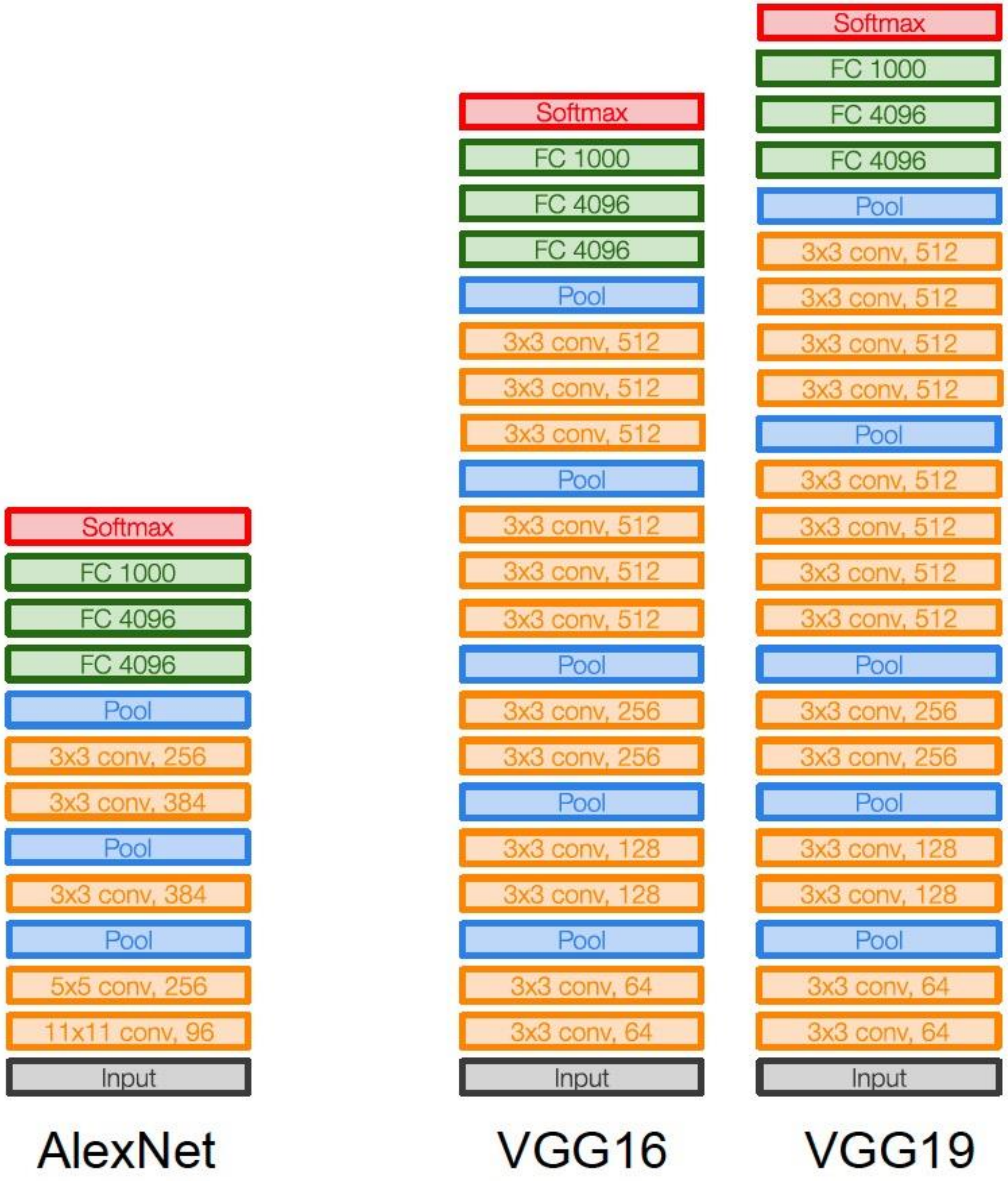
ImageNet Performance



VGGNet

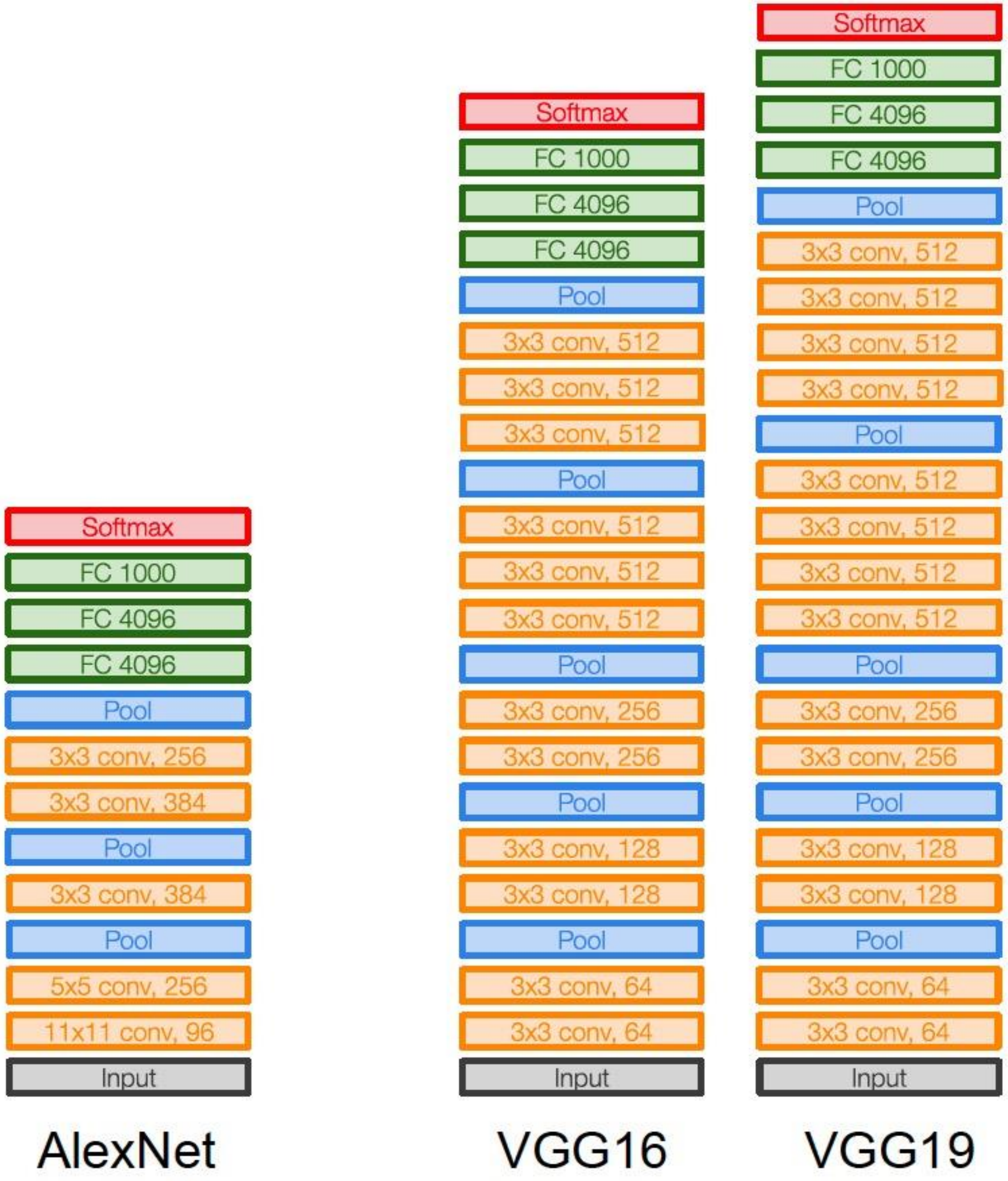
- AlexNet: Larger filters, less layers (8 layers).

- VGG: smaller filters, more layers (16 or 19 layers).

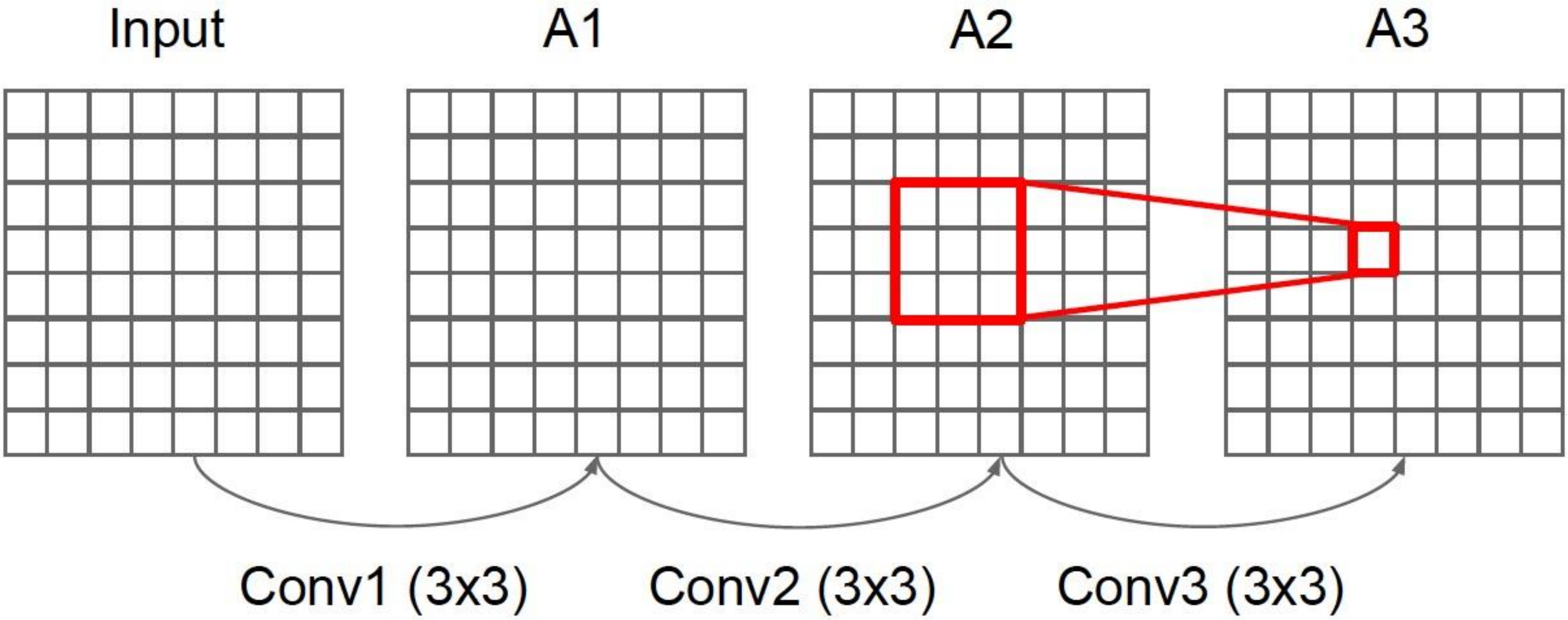


VGGNet

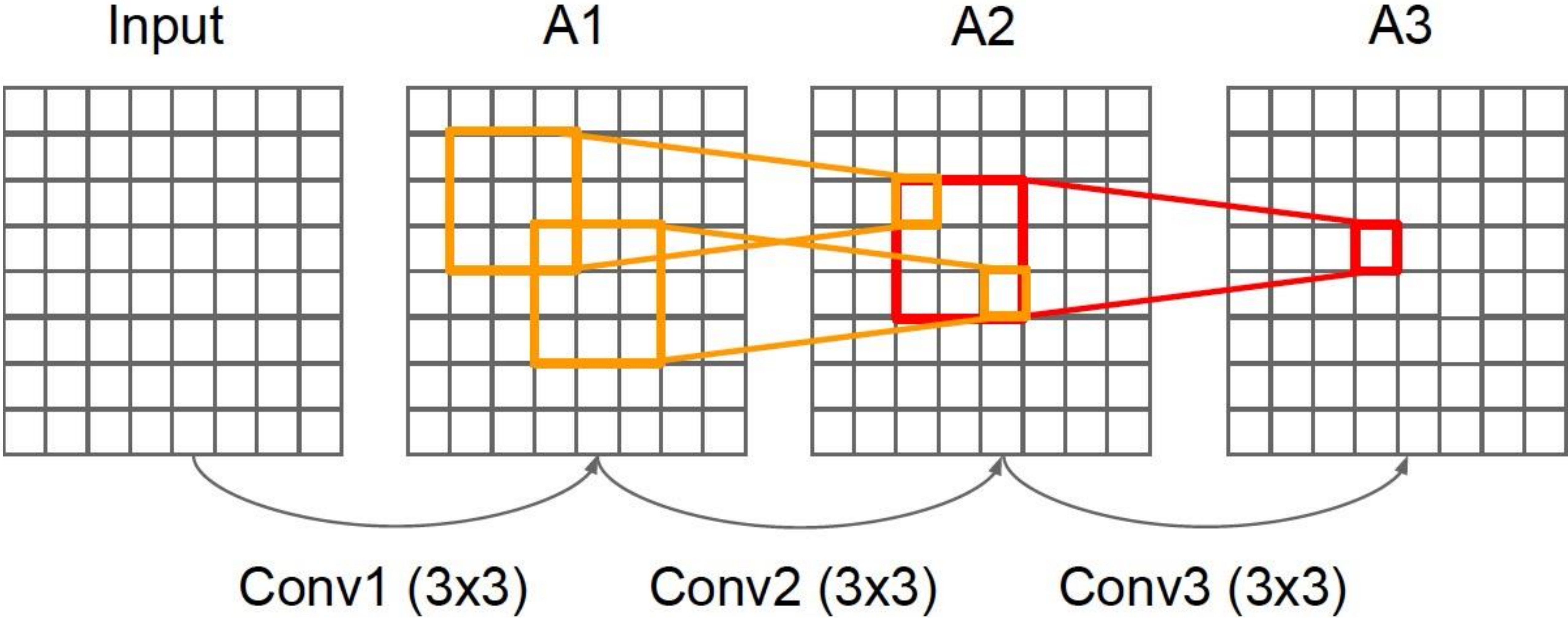
- A stack of three 3x3 conv filters has the same receptive field as a 7x7 conv filter
- Three 3x3 conv filters have more non-linear transformation



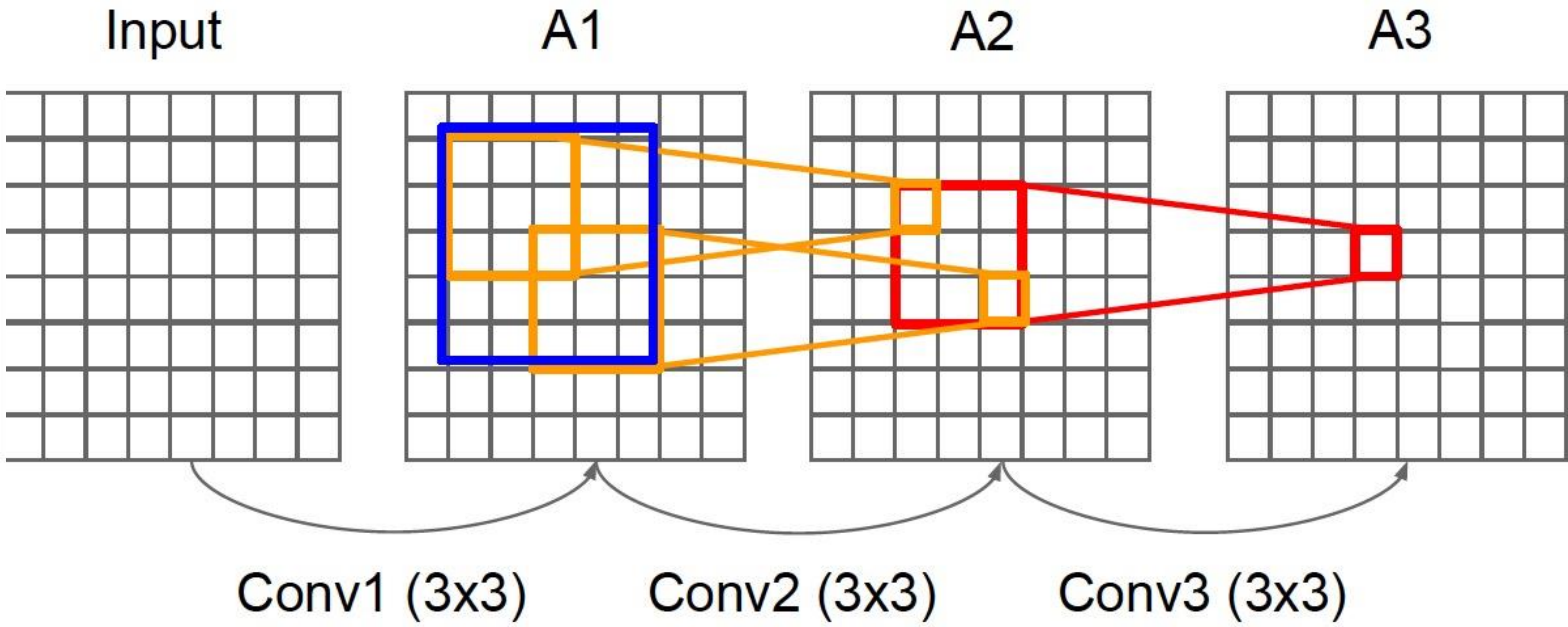
VGGNet-Receptive Fields



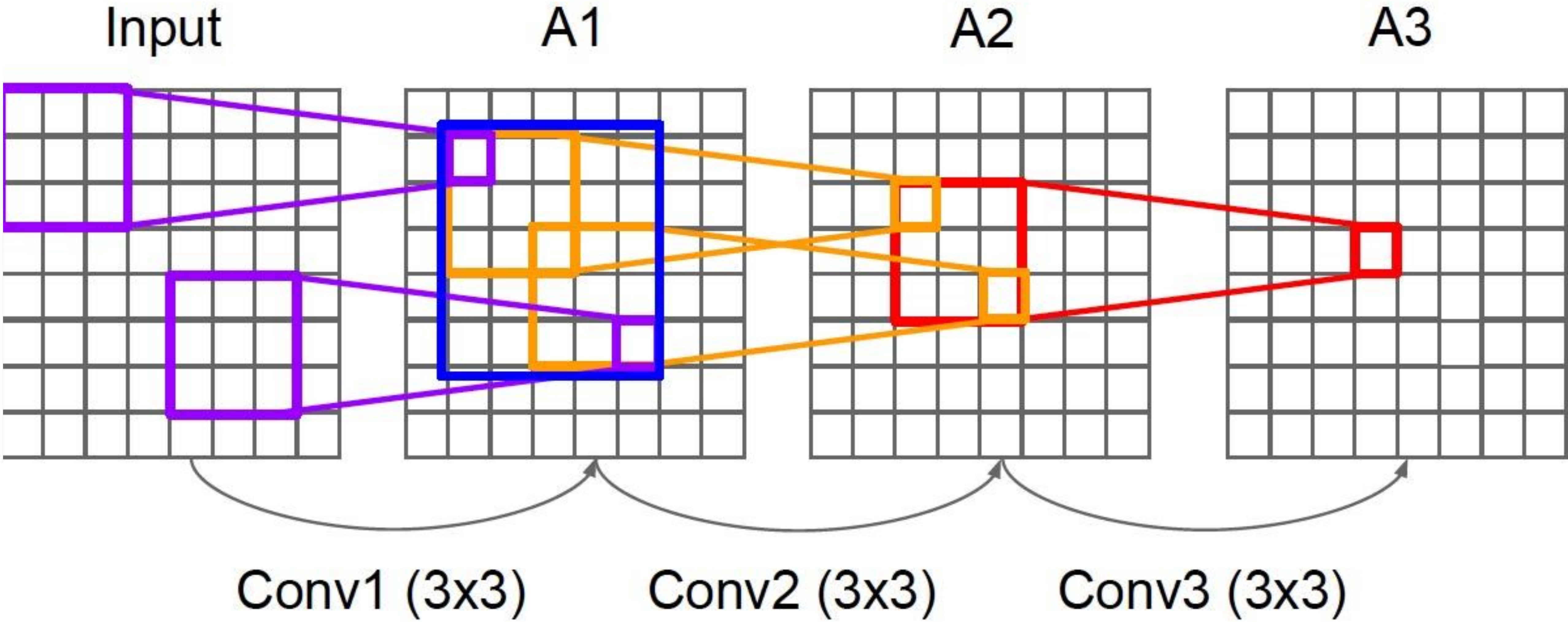
VGGNet-Receptive Fields



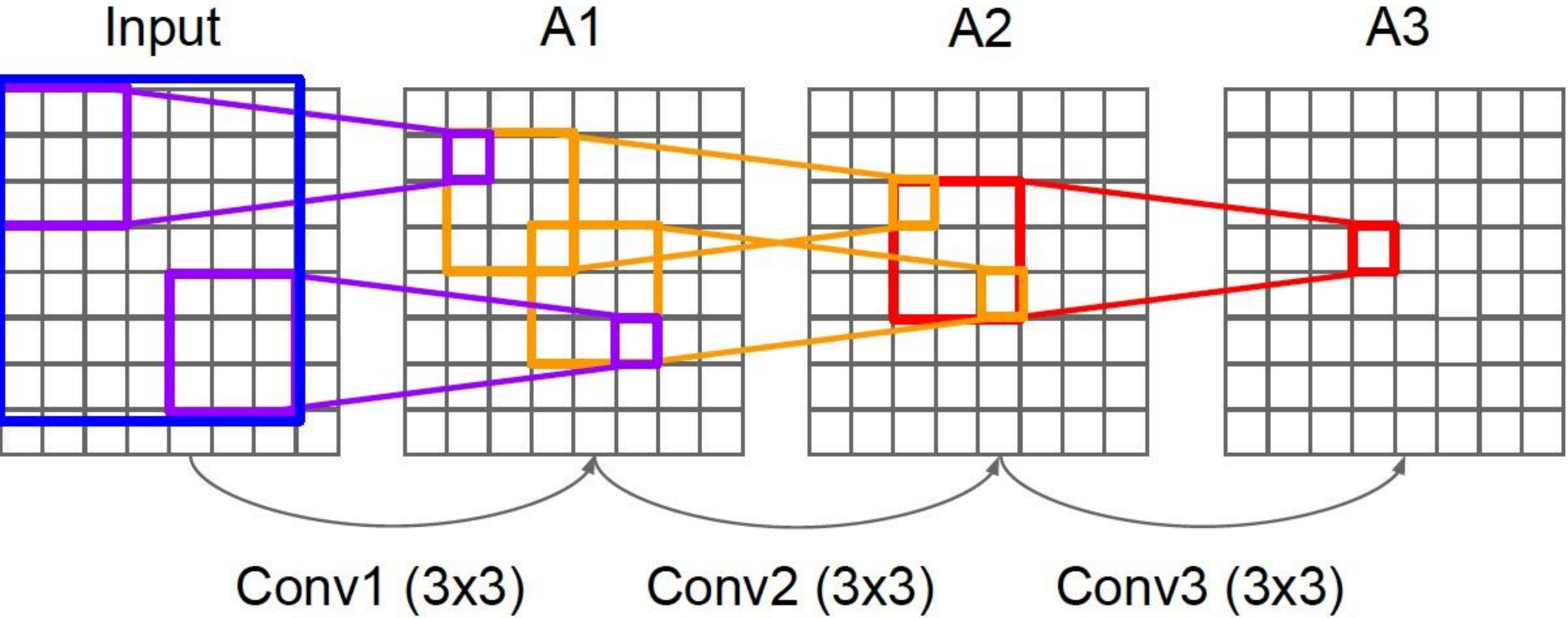
VGGNet-Receptive Fields



VGGNet-Receptive Fields

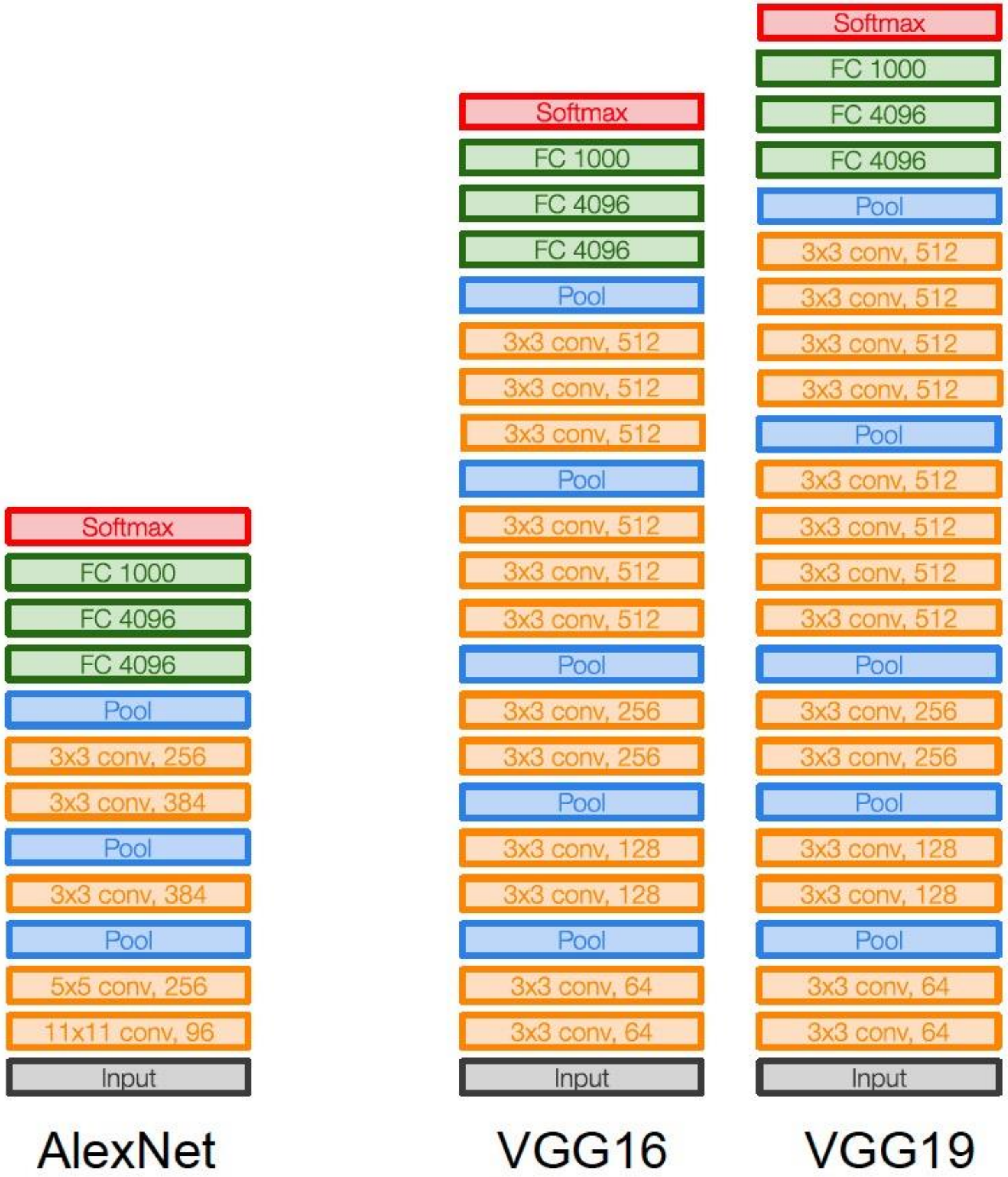


VGGNet-Receptive Fields

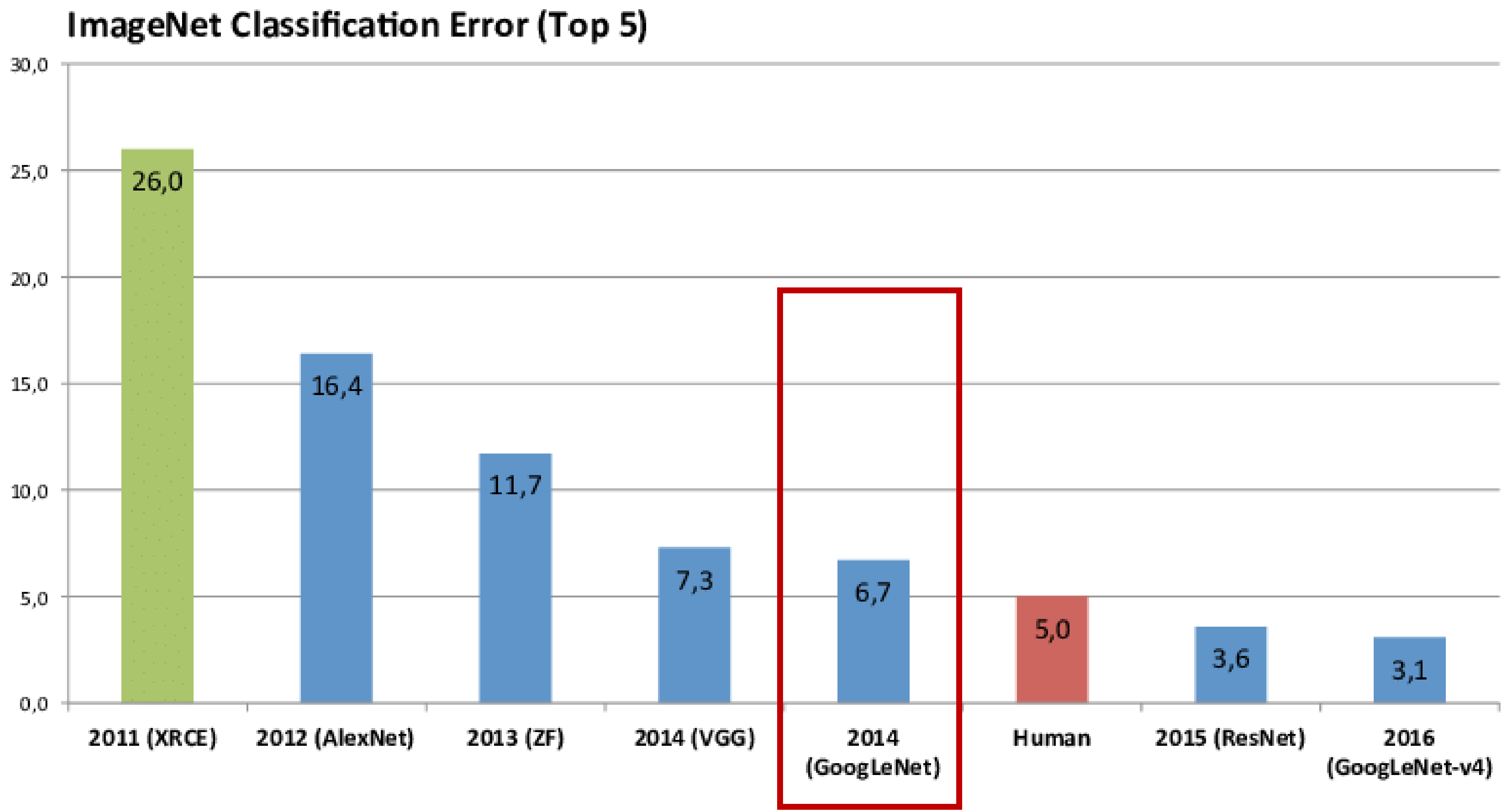


VGGNet

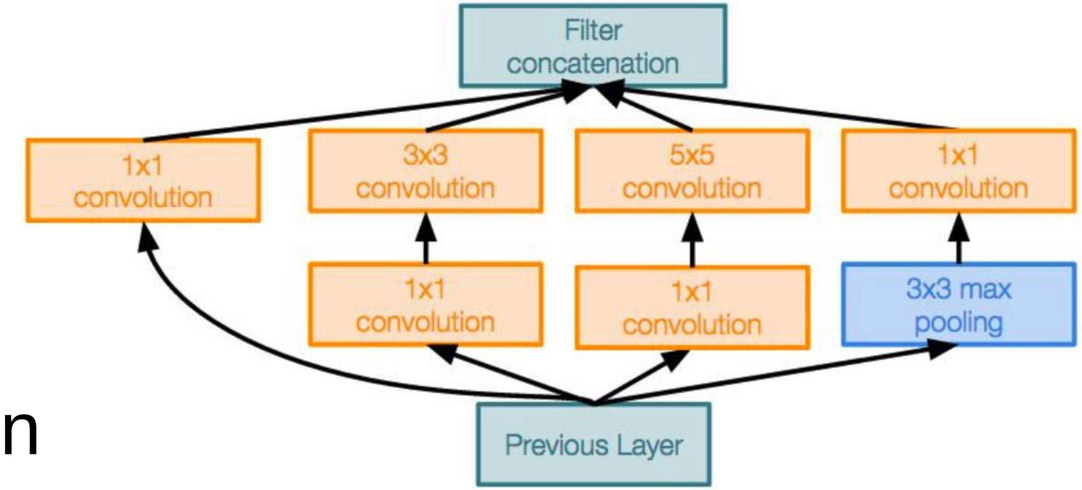
- A general direction: Going deeper with 3x3 convolution

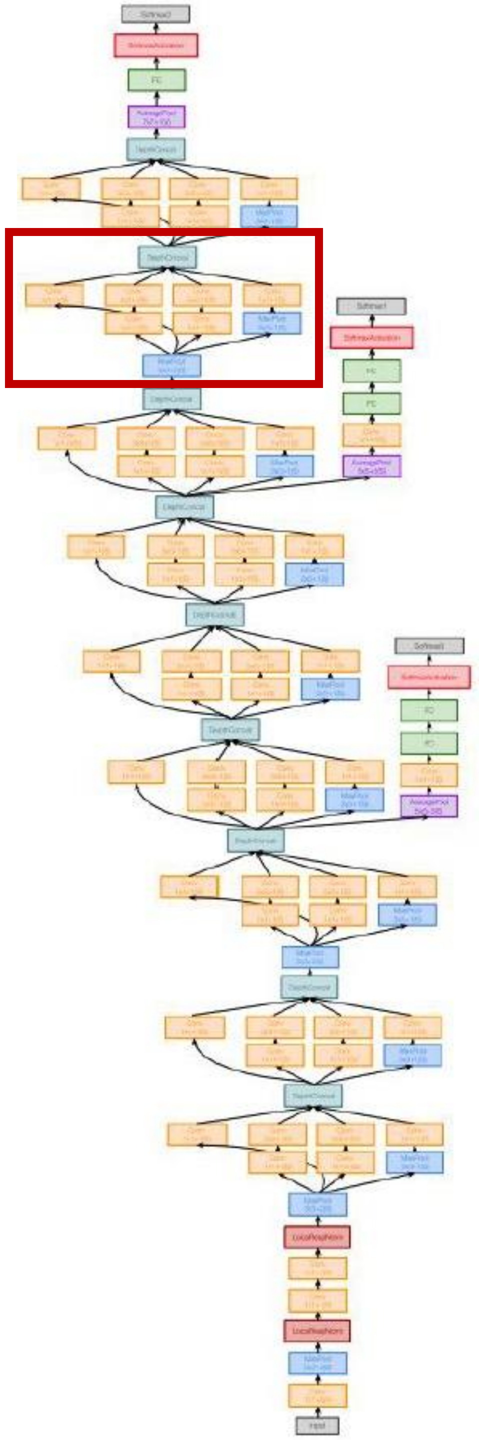


ImageNet Performance



GoogleNet

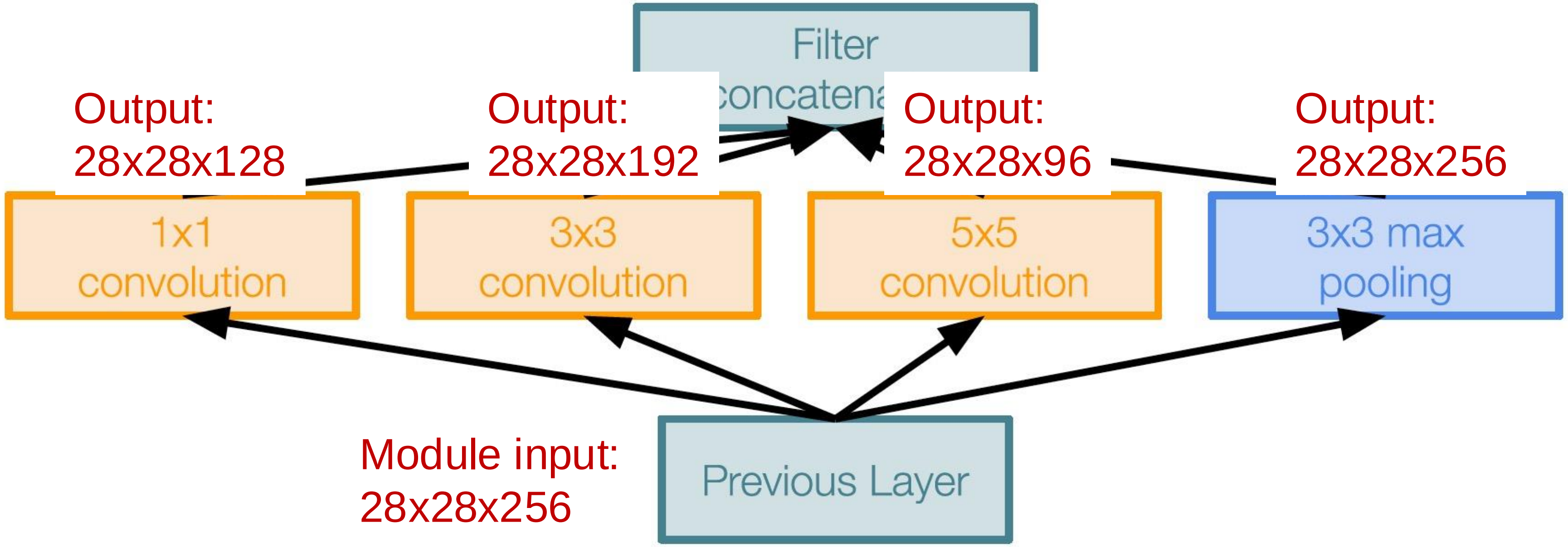




- Apply multiple filters in parallel

- Concat the results of multiple filters for the next layer

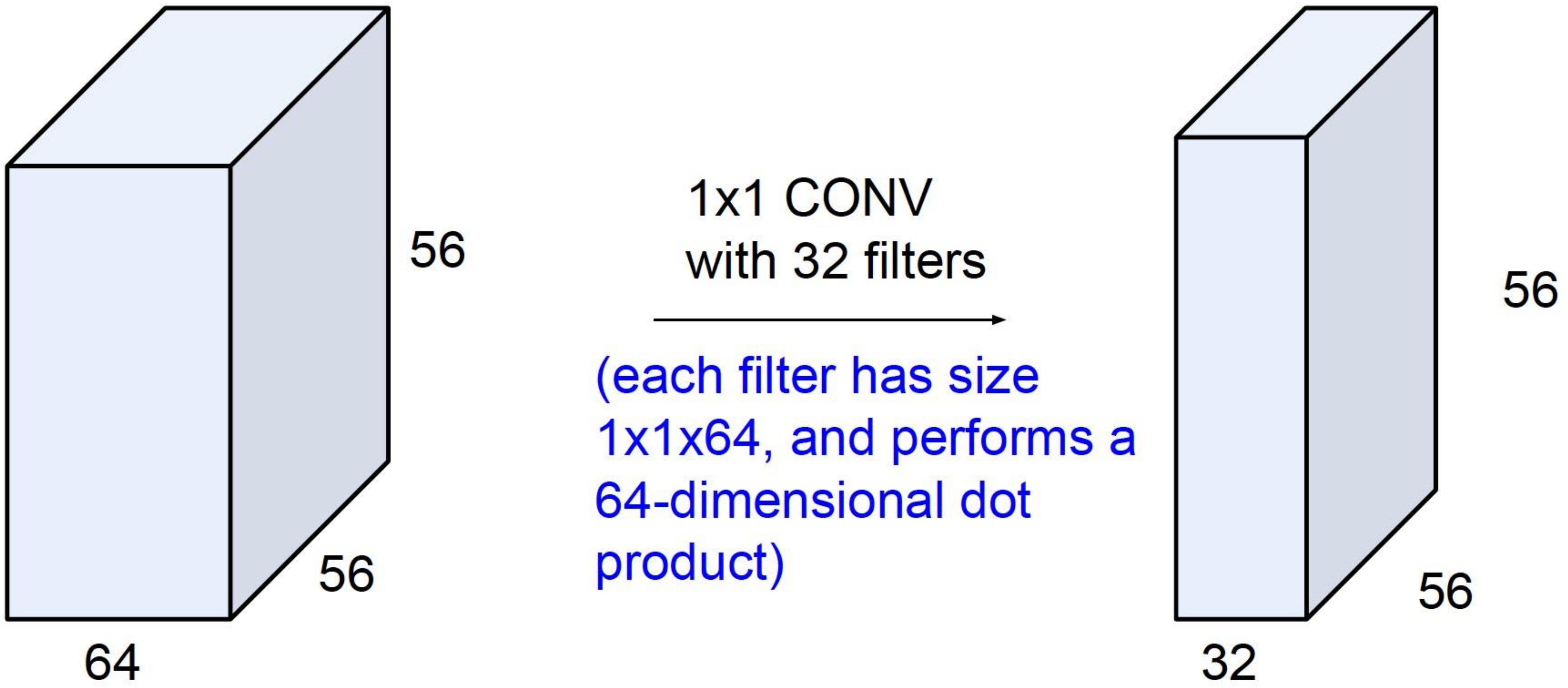
GoogleNet -- A naive inception module



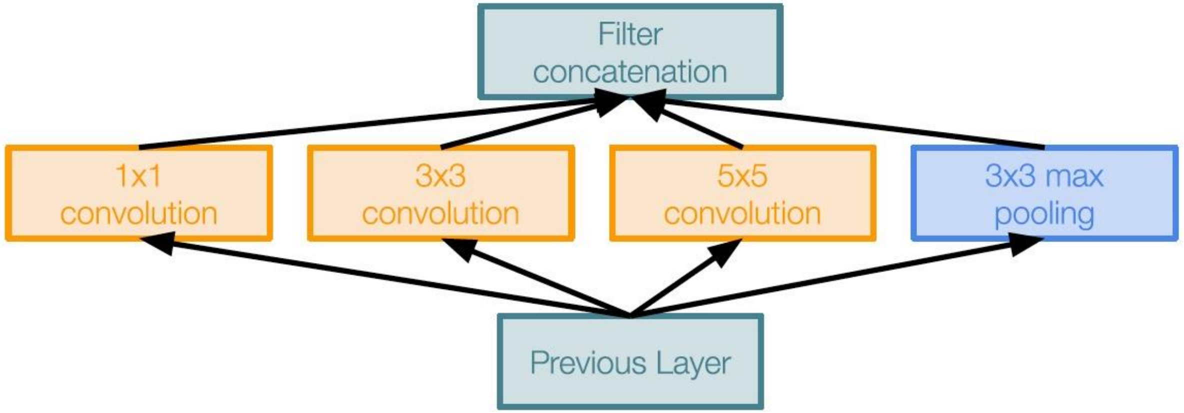
- Take 3x3 convolution as an example:
- Filter size: 3x3x192x256
- Conv Ops: 28x28x**3x3x192x256**

Can we reduce the computation?

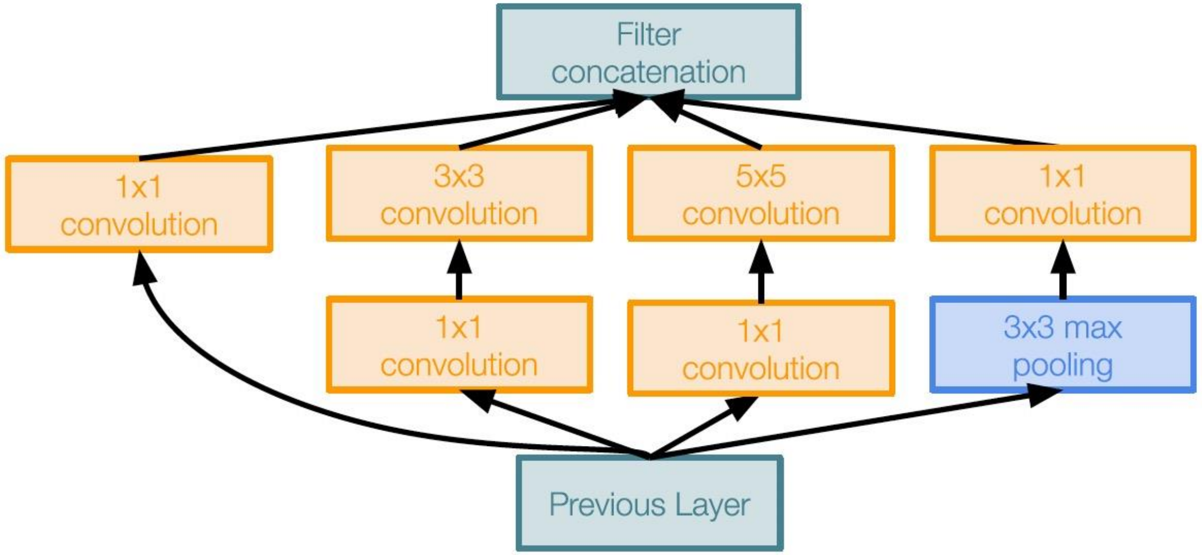
1 x 1 convolutions: dimension reduction



GoogleNet

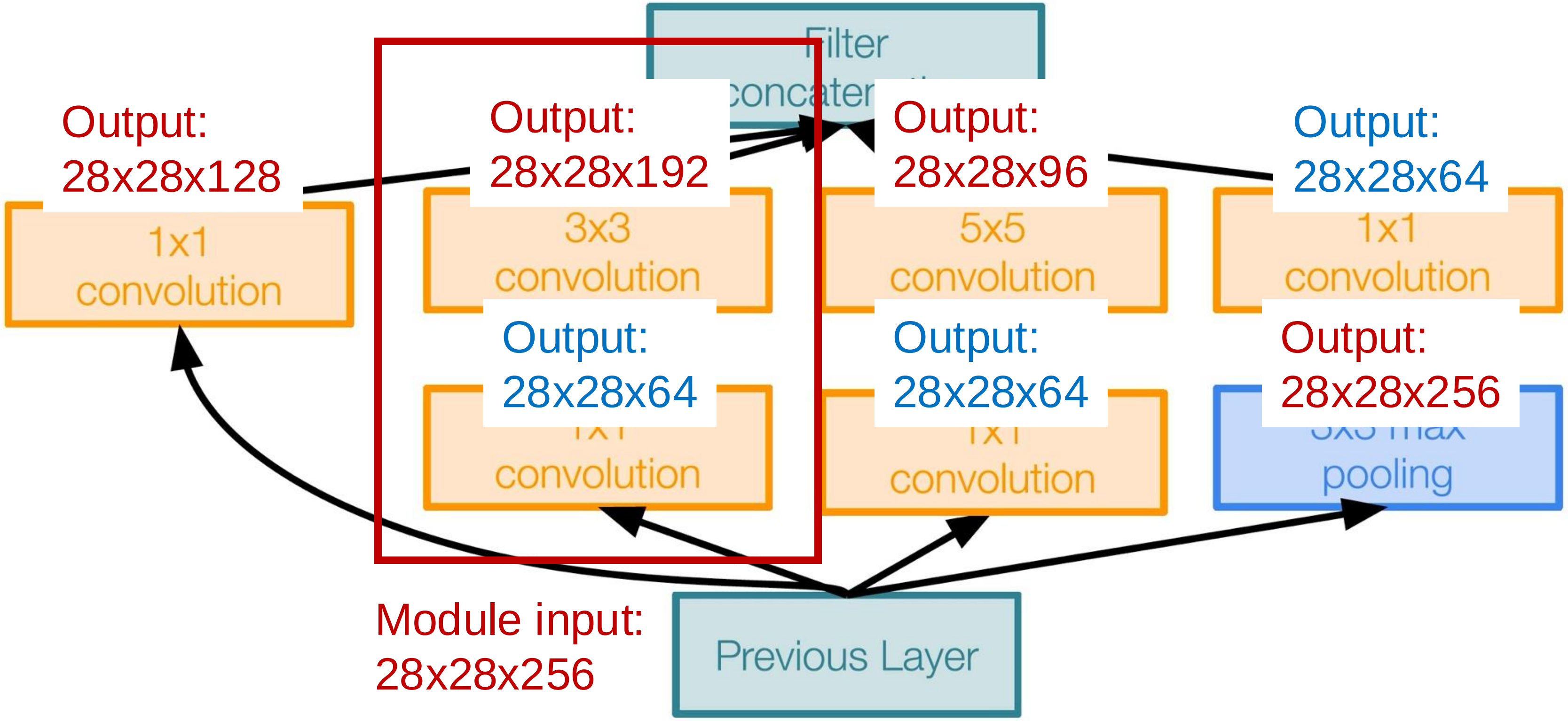


Naive Inception module



Inception module with dimension reduction

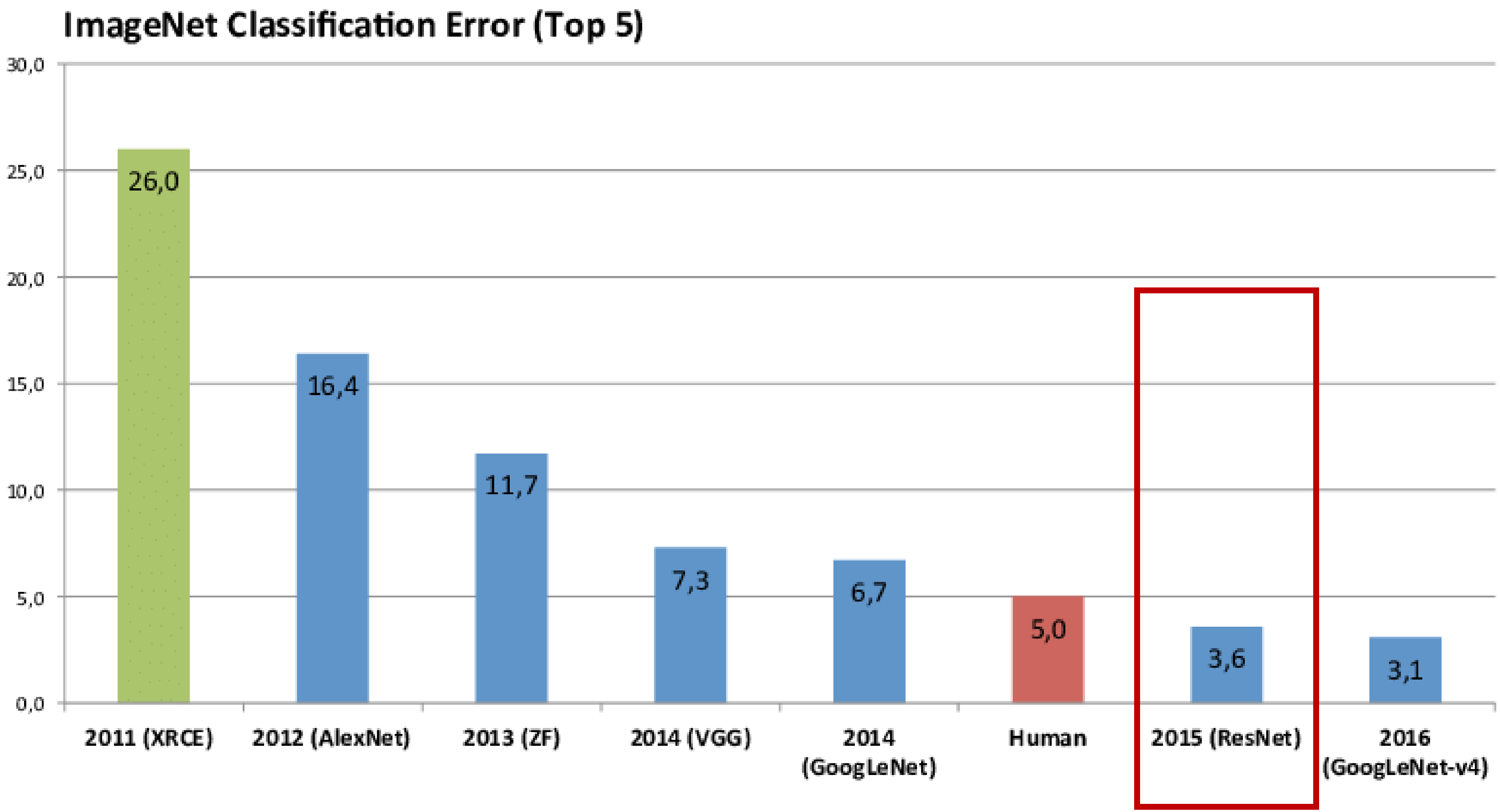
GoogleNet



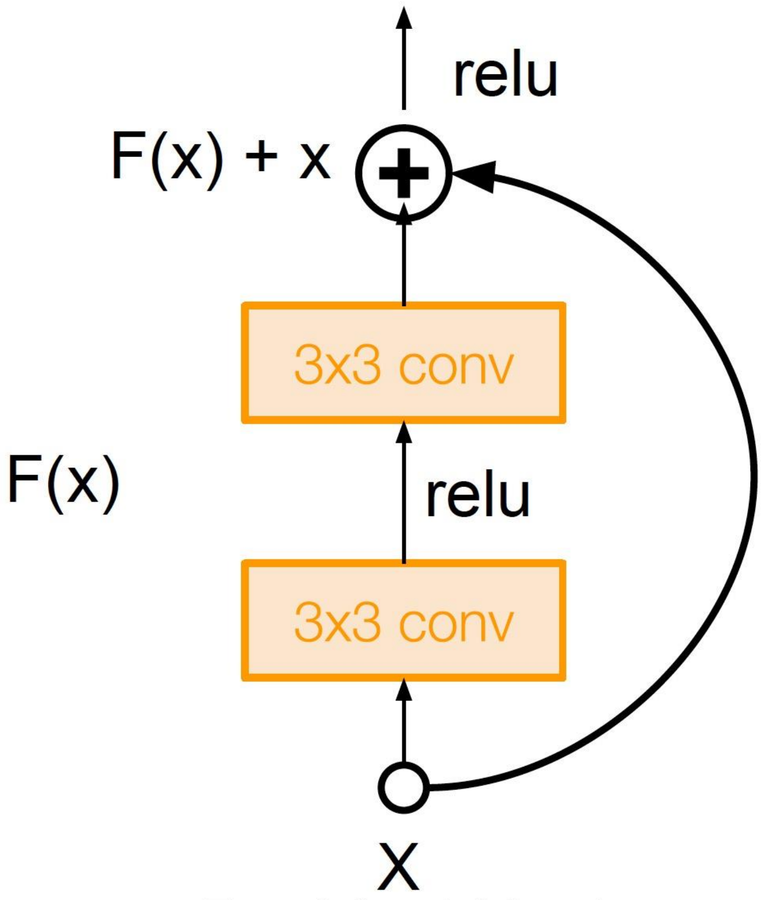
- Take 3x3 + 1x1 convolutions as an example:
- Filter size:
  3x3x192x64
  1x1x64x256
- Conv Ops:
  28x28x**3x3x192x64**
  28x28x**1x1x64x256**

Previous: 28x28x**3x3x192x256**

ImageNet Performance

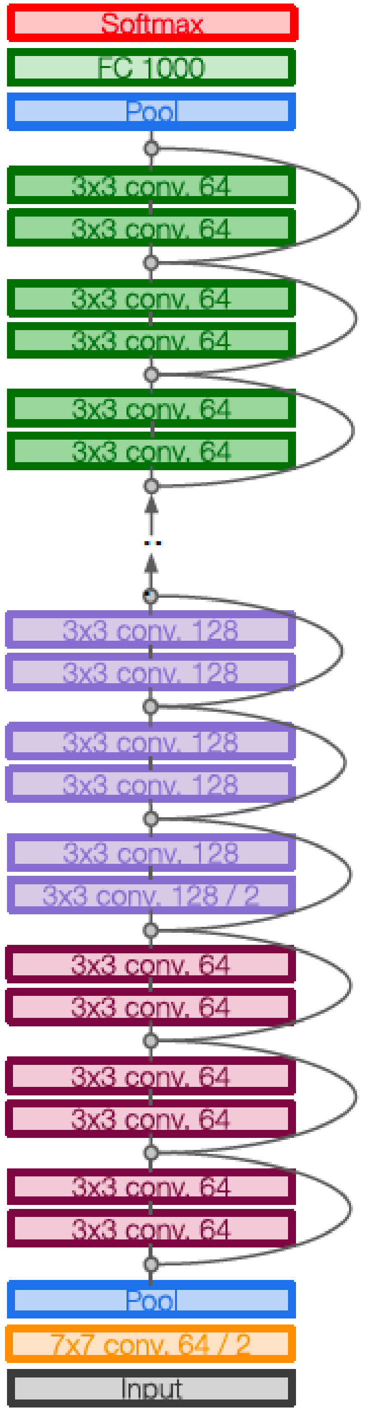


ResNet



Residual block

$$y = F(x) + x$$



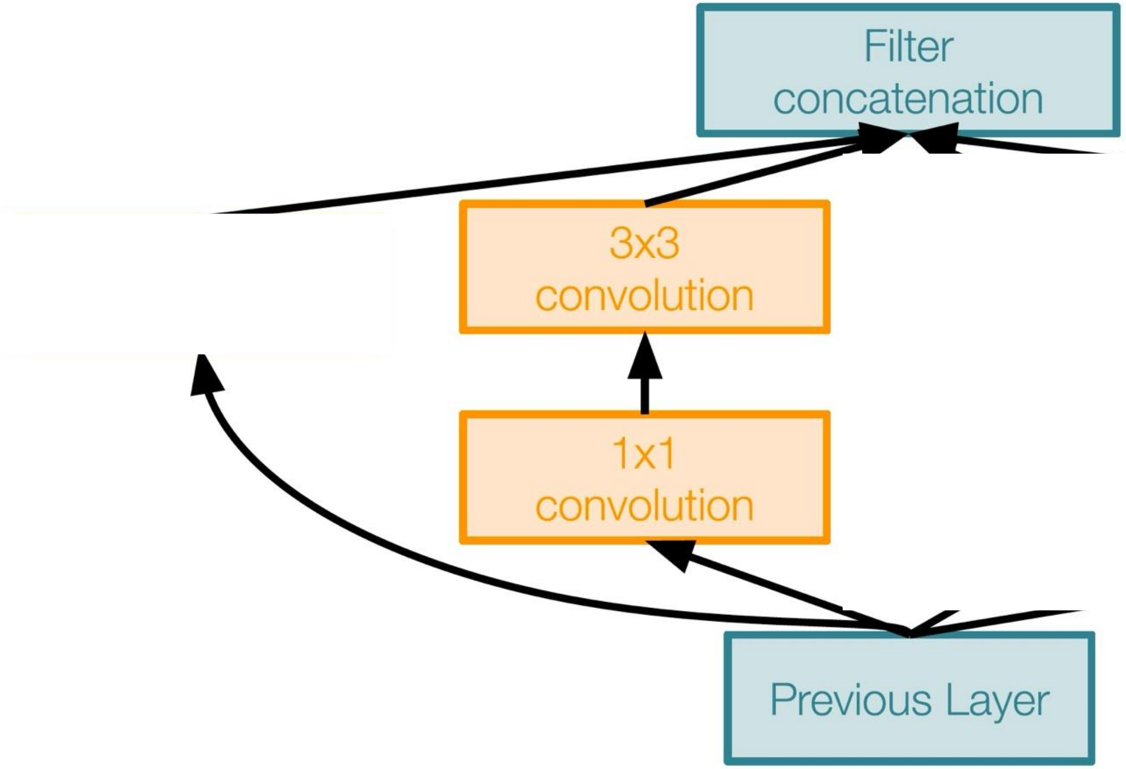
How is ResNet developed?

- Simplifying GoogleNet Inception module!

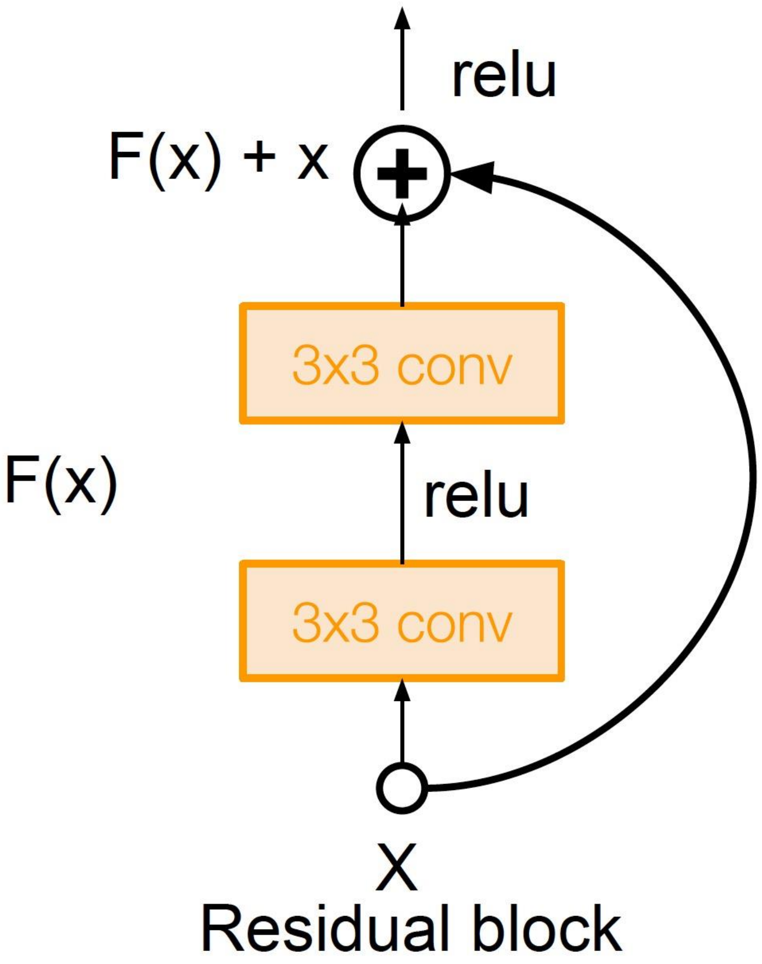
GoogleNet ResNet VGG16

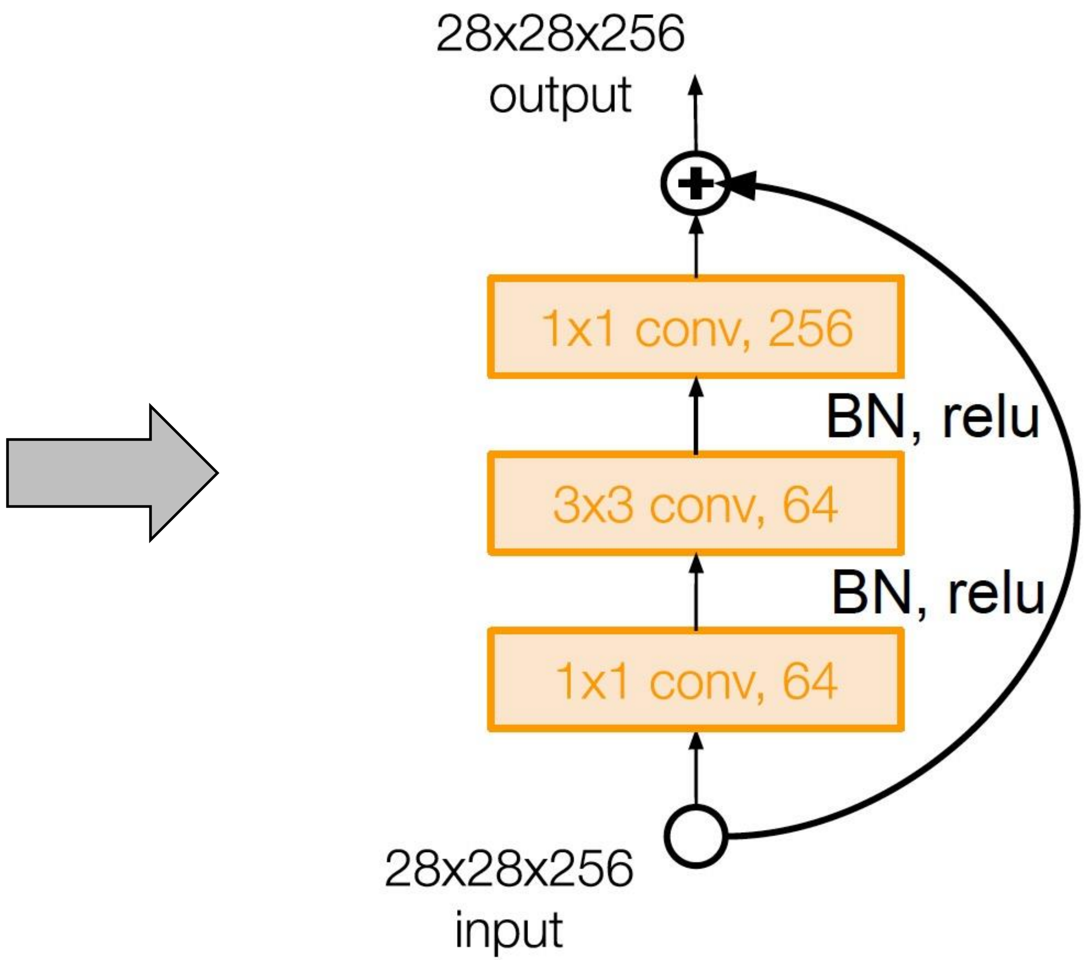
How is ResNet developed?

- Simplifying Inception module!



BottleNeck with 1x1 convolution

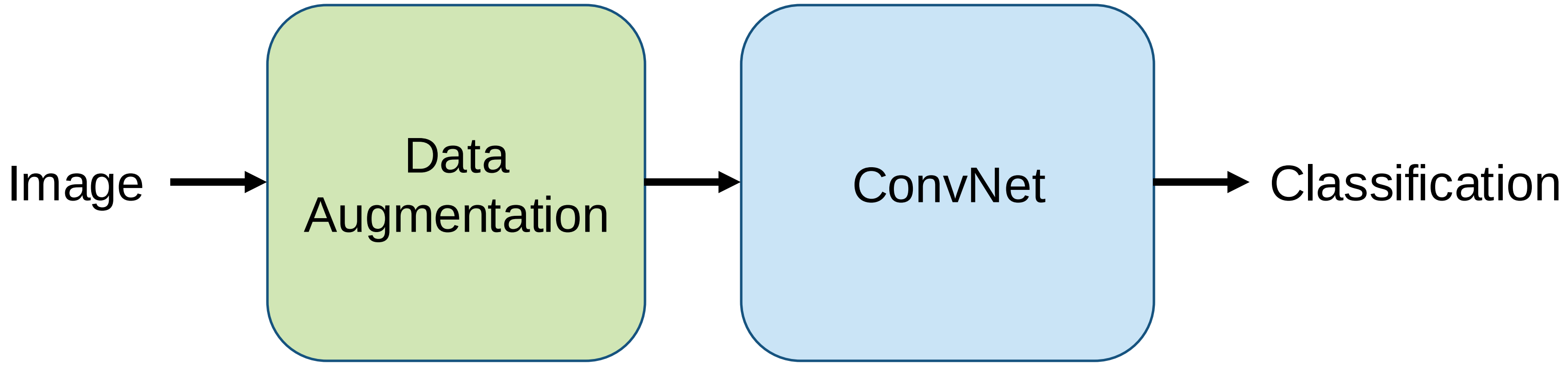




Data Augmentation

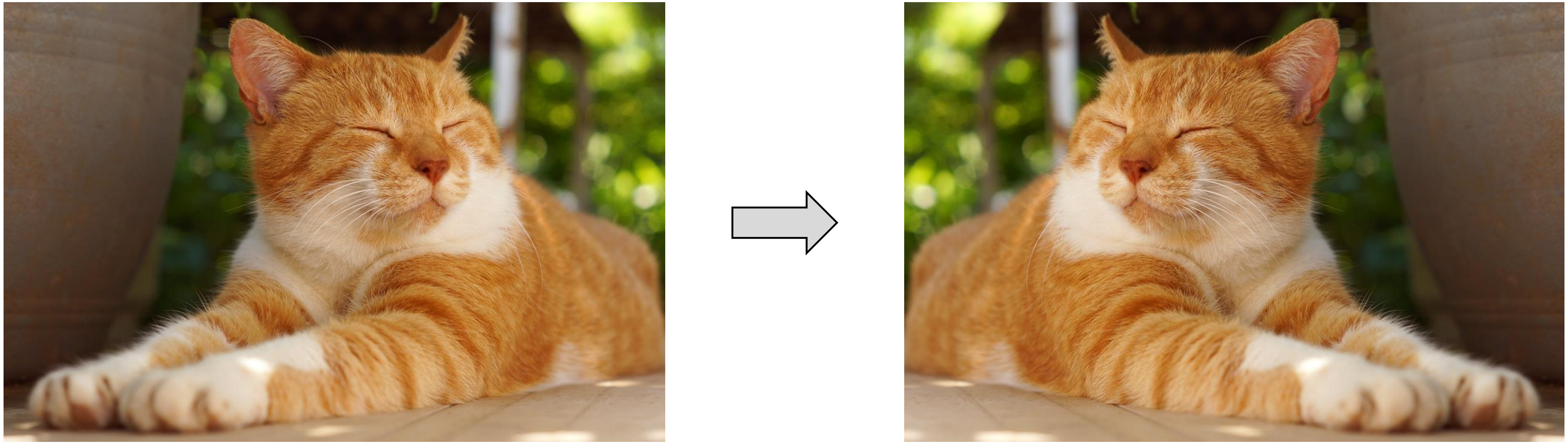
Data Augmentation

- Data augmentation is a free way to increase training data
- Prevent overfitting
- Improve performance



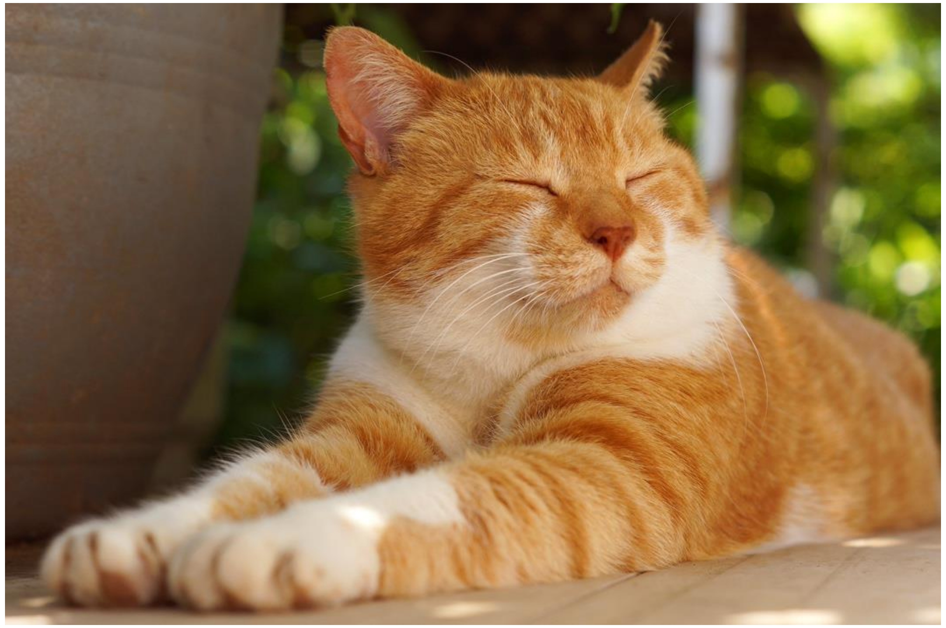
Data Augmentation for Classification

- Horizontal Flip (useful)



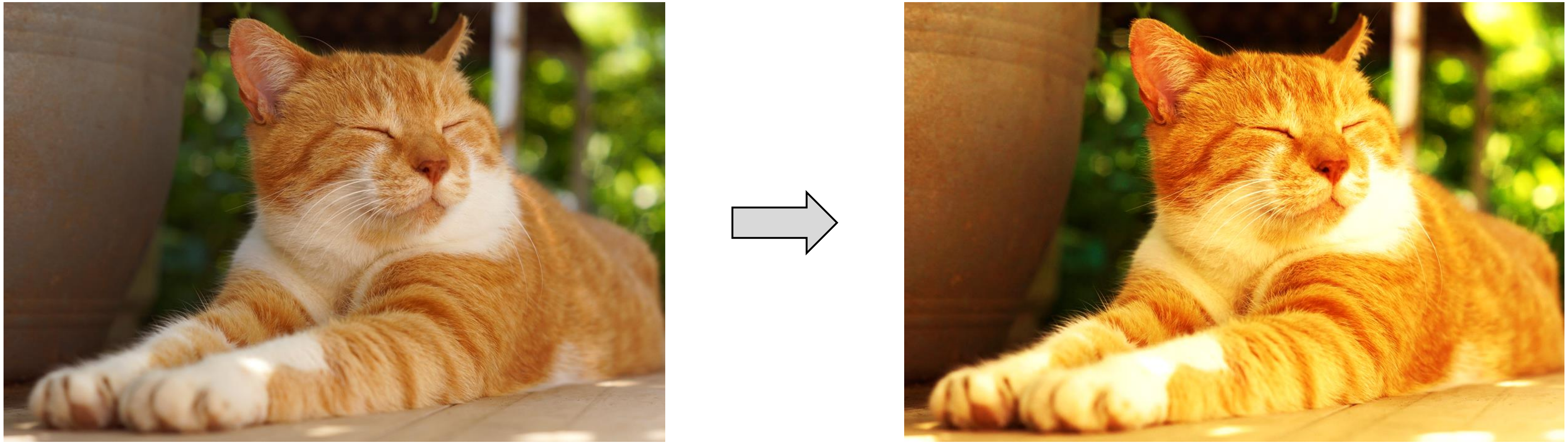
Data Augmentation for Classification

- Random Crop (critical)



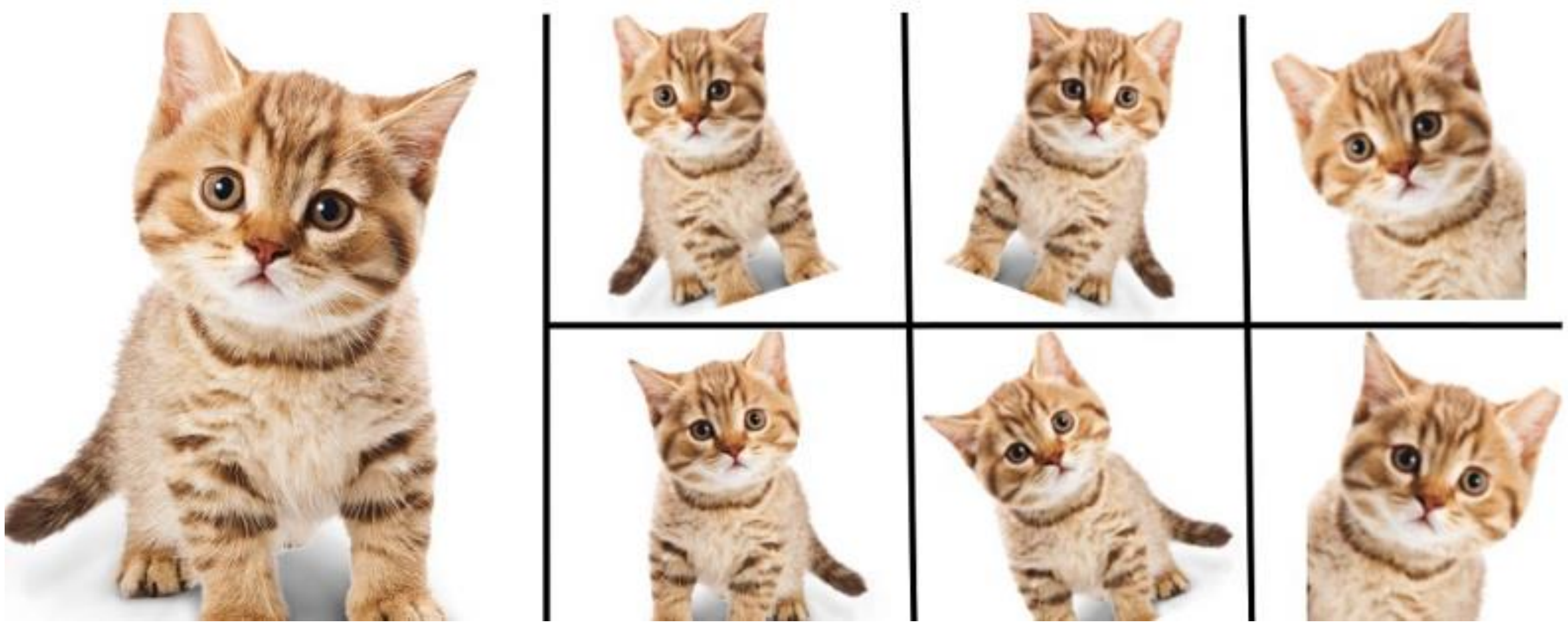
Data Augmentation for Classification

- Color augmentation, brightness, contrast (can ignore)



Data Augmentation for Classification

- Rotation (sometimes useful, especially for pose estimation)



Data Augmentation for Classification

- Training:
  - Pick a random L in range [256, 480]
  - Resize the image, the short side is resized to length L, maintaining the original aspect ratio
  - Randomly crop an [224, 224] patch out of the image

- Testing:
  - Resize the image, the short side is resized to length 256
  - Crop an [224, 224] patch from the center of the image

Next Class

PyTorch Tutorial