Lecture 02: Convolutional Neural Networks and Variants

Course Information

- Course website: https://www.saiqianzhang.com/COURSE/
- I use Brightspace to post announcements and grades
- I provide an online zoom meeting option for people interested in auditing the class. However, enrolled students are required to attend in person unless special condition.
- A suggested reading list which contains interesting papers can be found here.
- Discussion groups has been created in the Brightspace
- Course email: efficientaiaccelerator@gmail.com

Recap

- DNN basics
  - Multilayer perceptron
    - Linear layer, activation function, softmax layer
  - Loss functions
  - Weights decay
  - Dropout
  - Optimizer
  - Learning rate scheduler
  - Weight Initialization

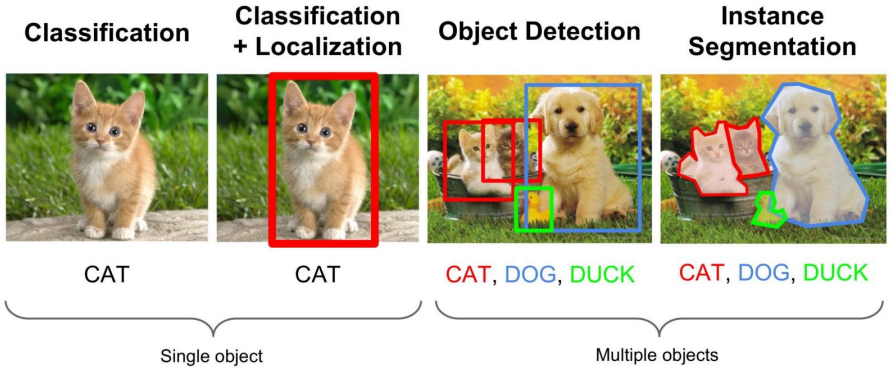
Topics

- Convolutional Neural Network
  - Basic building blocks
  - Popular CNN architectures
    - VGG
    - ResNet
    - MobileNet
    - ShuffleNet
    - SqueezeNet
    - DenseNet
    - EfficientNet
    - ConvNext
    - ShiftNet
  - CNN architectures for other vision tasks
    - Image Segmentation, Object Detection

Convolutional Neural Networks

- Convolutional Neural Networks (CNNs) are a type of artificial neural network designed for processing structured grid data, such as images. They're particularly effective in tasks like image recognition, object detection and segmentation.

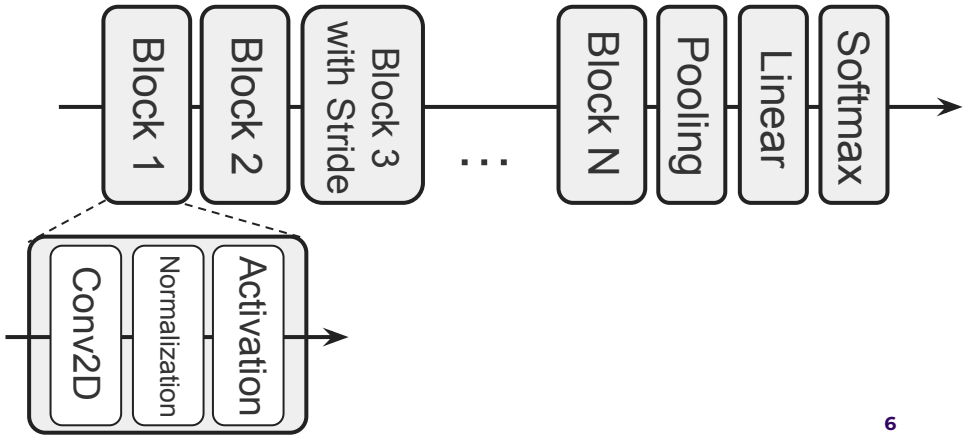
- The building blocks of a CNN includes:
  - Convolutional layer
  - Activation layer
  - Normalization layer
  - Pooling layer
  - Softmax layer



Convolutional Neural Networks

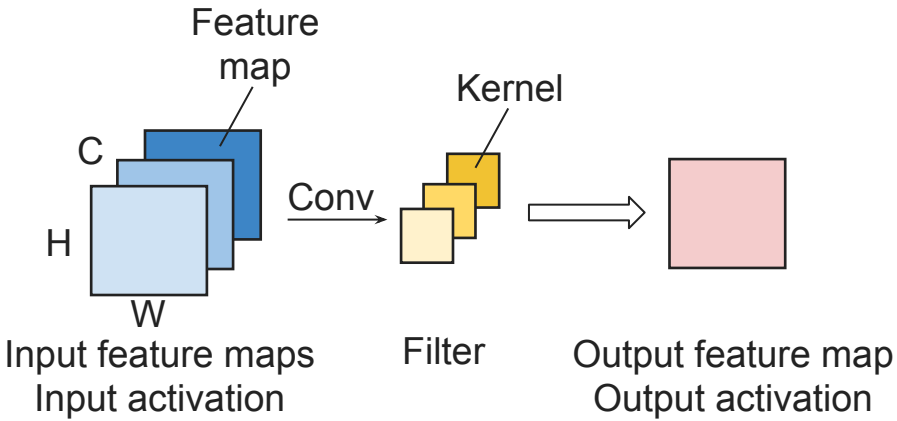
- Convolutional Neural Networks (CNNs) are a type of artificial neural network designed for processing structured grid data, such as images. They're particularly effective in tasks like image recognition, object detection and segmentation.

- The building blocks of a CNN includes:
  - Convolutional layer
  - Activation layer
  - Normalization layer
  - Pooling layer
  - Softmax layer



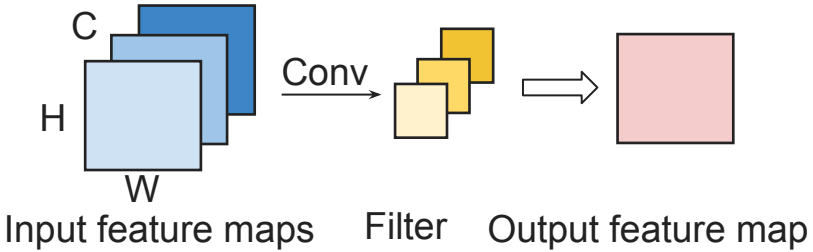
NYU SAI LAB

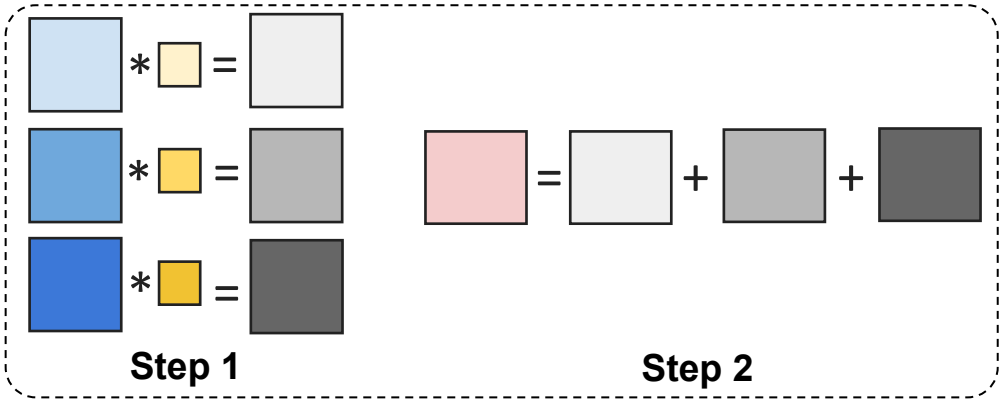
Convolutional Layers: Terminology



- Core building block of a CNN, it is also the most computational intensive layer.

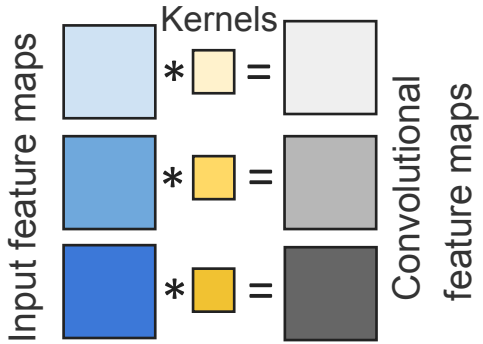
Convolutional Layers





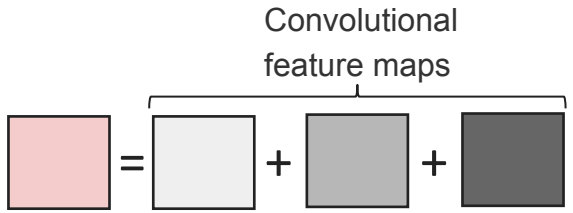
- Core building block of a CNN, it is also the most computational intensive layer.

Convolutional Layers



Step 1

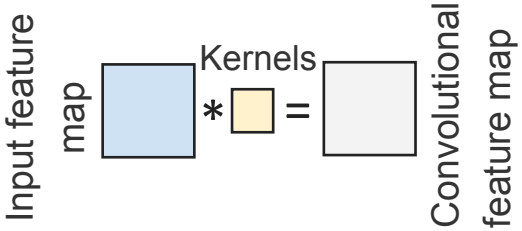
- Each kernel moves across the spatial dimensions of feature maps in the input activations, analyzing the information within those spatial dimensions.



Step 2

- The information from each feature maps are then aggregated by summing the Convolutional feature maps together.
- A bias may be introduced.

2D Convolution: An Example



Input feature map

1	0	5
3	-1	-2
4	0	-3

* Kernels

1	0
0	2

= Convolutional feature map

-1	-4
3	-7

1	0	5	Inner product with filter →	-1	
3	-1	-2			
4	0	-3			

1	0	5	Inner product with filter →	-4	
3	-1	-2			
4	0	-3			

1	0	5	Inner product with filter →	3	
3	-1	-2			
4	0	-3			

1	0	5	Inner product with filter →	-7	
3	-1	-2			
4	0	-3			

NYU SAI LAB

Paddings

Input feature map

1	0	5
3	-1	-2
4	0	-3

*

Kernel

1	0	3
0	2	-2
1	3	-1

= 25

- Padding is used to preserved the spatial size of the output features.

Padding of 1

Input

0	0	0	0	0
0	1	0		0
0	3	-1	-2	0
0		0	-3	0
0	0	0	0	0

*

Filter

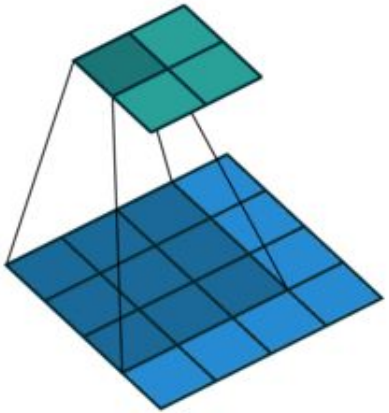
1	0	3
0	2	-2
1	3	-1

=

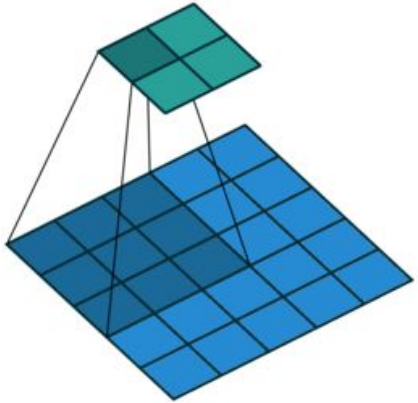
12	20	3
20	25	-10
5	3	-7

NYU SAI LAB

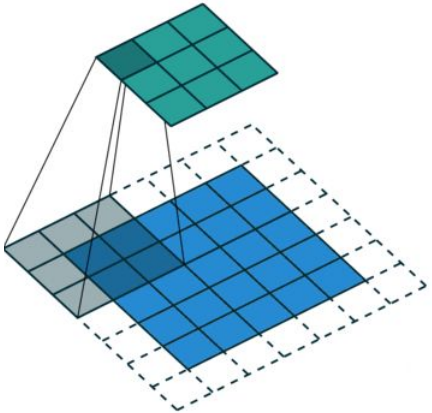
Stride



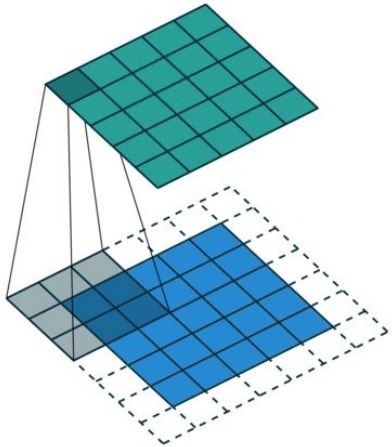
padding = 0, stride = 1



padding = 0, stride = 2



padding = 1, stride = 2

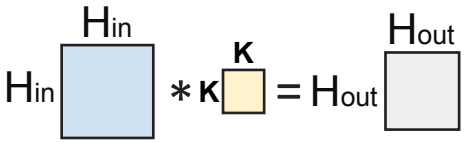


padding = 1, stride = 1

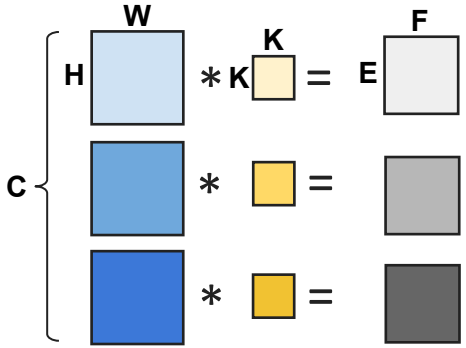
https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

Summary

- $H_{out} = (H_{in} - K + 2P)/S + 1$
- H_{in} and H_{out} are the spatial sizes of the input and convolutional feature maps.
- K is the weight kernel size
- P is the padding size
- S is the stride
- For example:
  - For input size of 224x224x3, weight kernel size is 3x3, padding size is 1 and stride size is 1, then the output size is (224-3+2)/1 + 1 = 224.

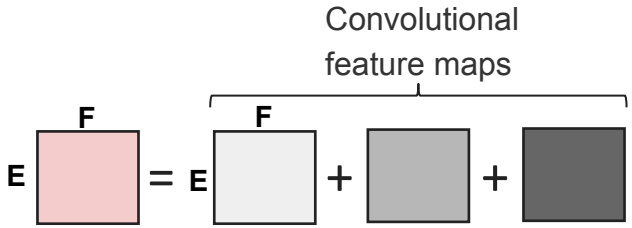


Computational Cost



Step 1

- Each kernel moves across the spatial dimensions of feature maps in the input activations, analyzing the information within those spatial dimensions.



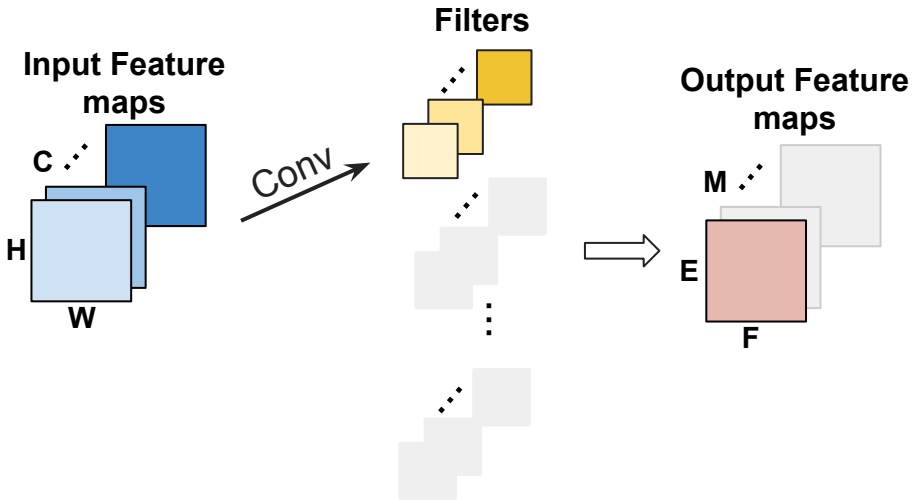
Step 2

- The information from each feature maps are then aggregated by summing the Convolutional feature maps together.
- A bias may be introduced.

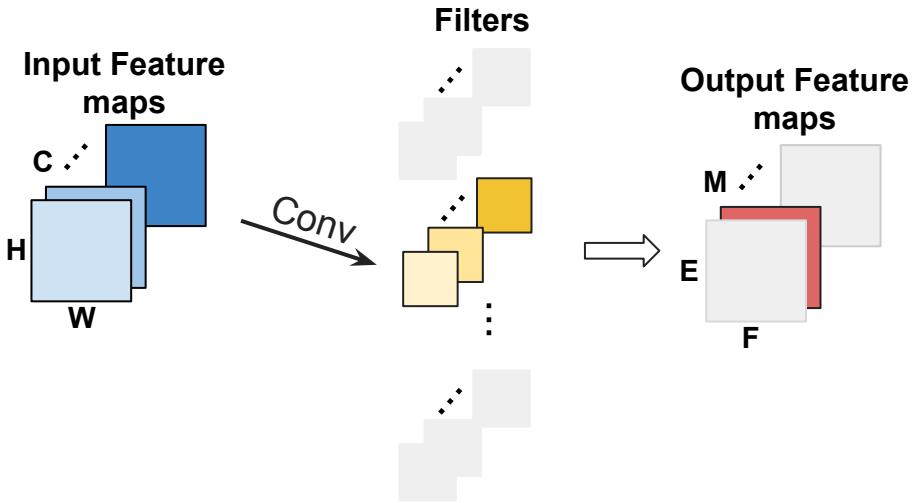
Computational cost in Multiply–accumulate operations (MAC): E✖F✖K✖K✖C

NYU SAI LAB

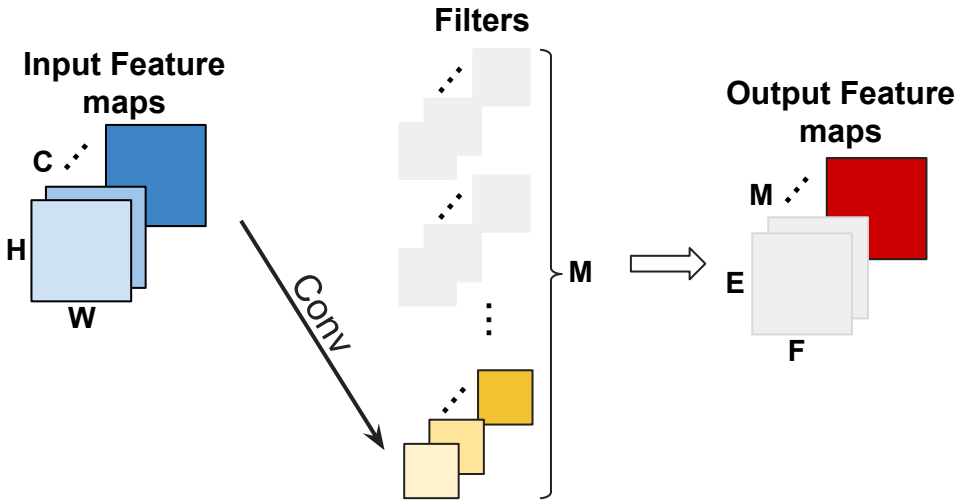
Convolution



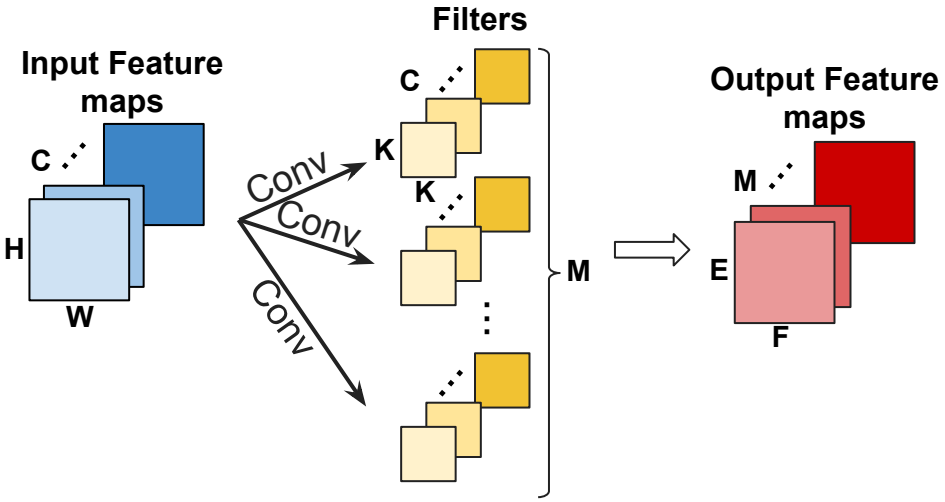
Convolution



Convolution



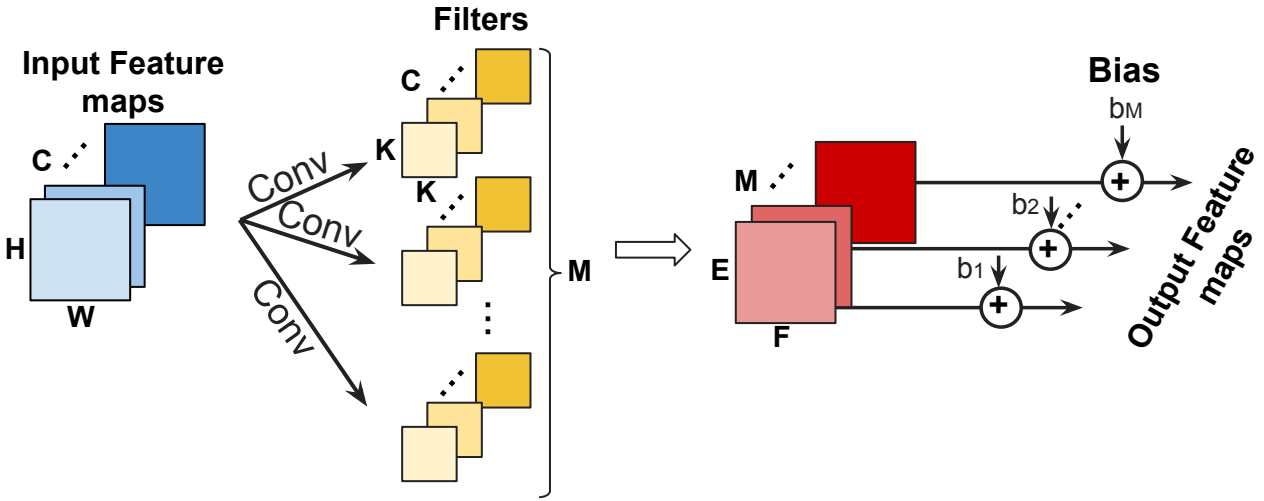
Convolution



- Number of MACs: M×K×K×C×E×F
- Storage cost: 32×(M×C×K×K+C×H×W+M×E×F)

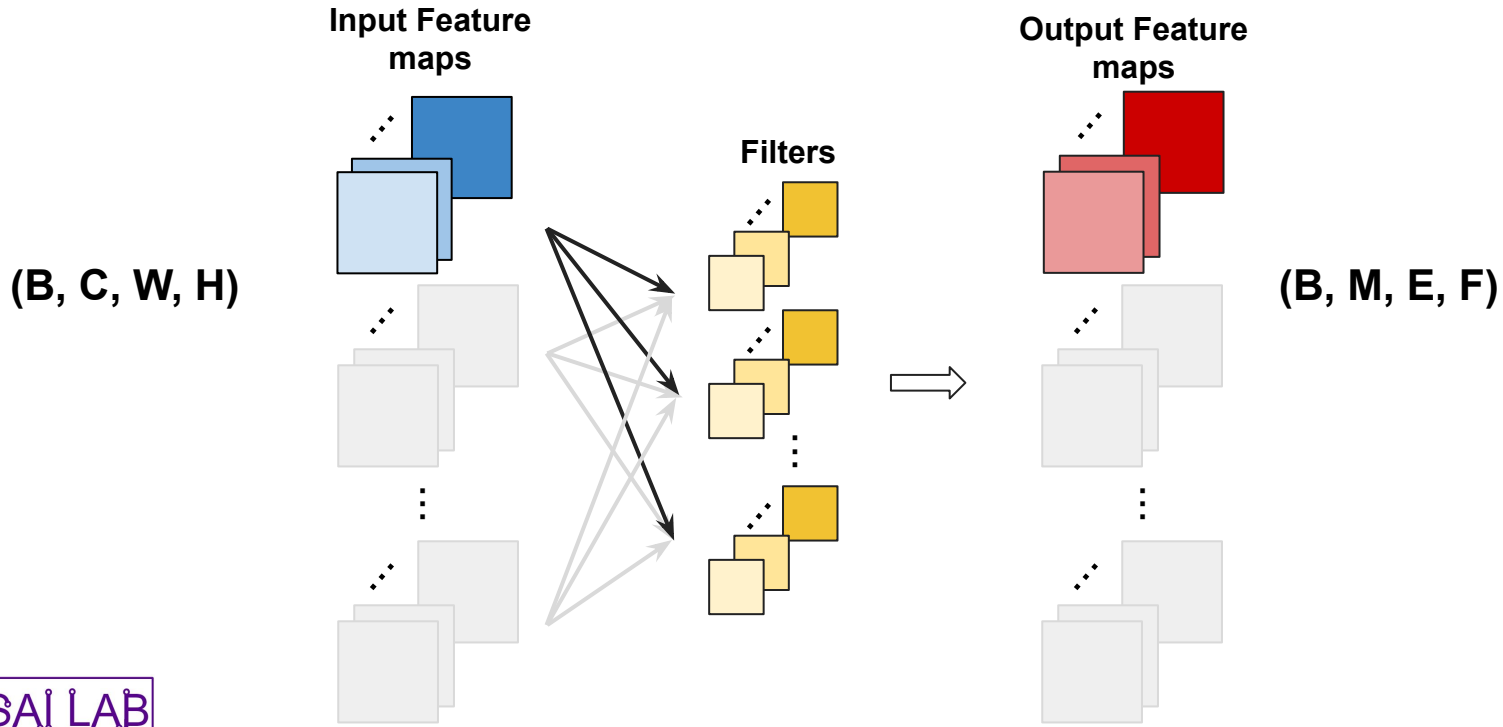
C: number of input channels
H,W: size of the input feature maps
M: number of weight filters
K: weight kernel size
E,F: size of the output feature maps

Convolution

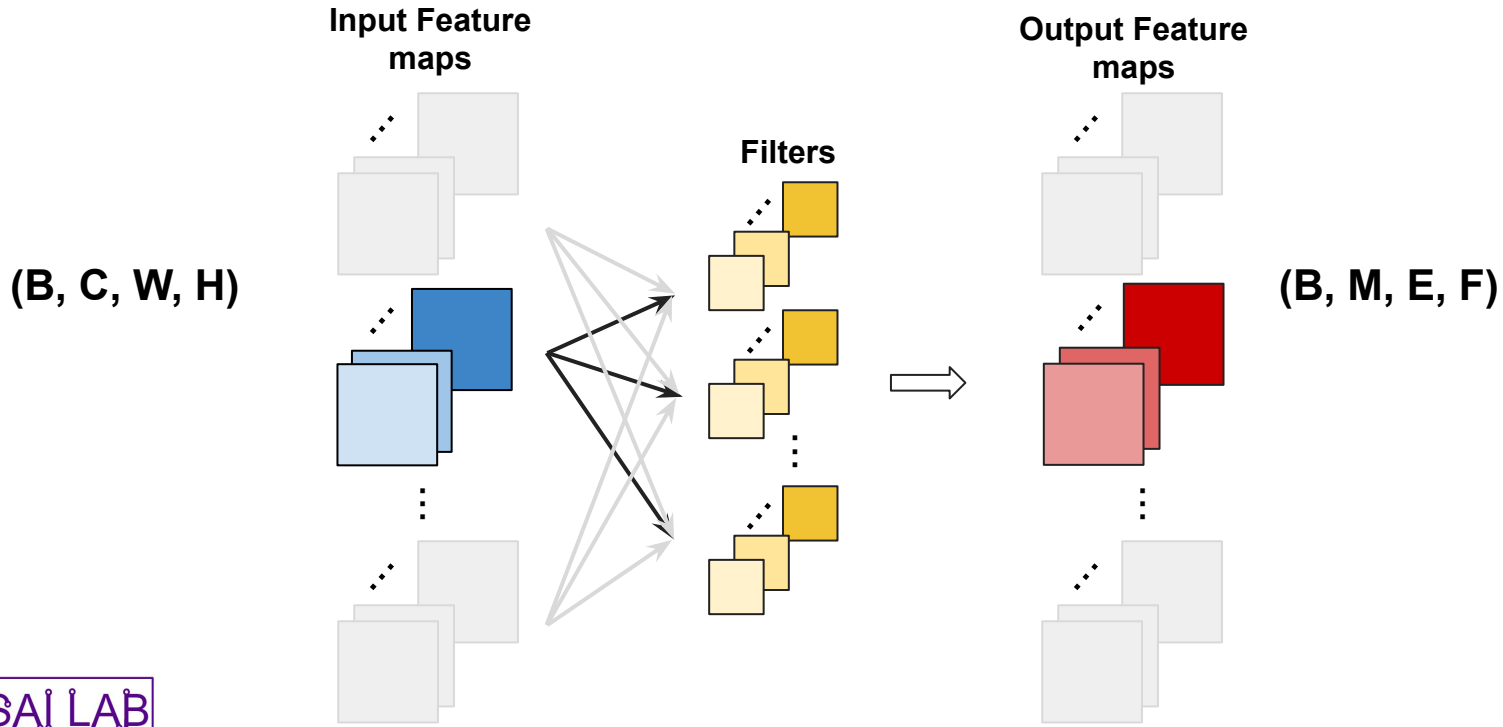


- A bias term may be added to the convolutional output, applied across each output channel.

Convolution

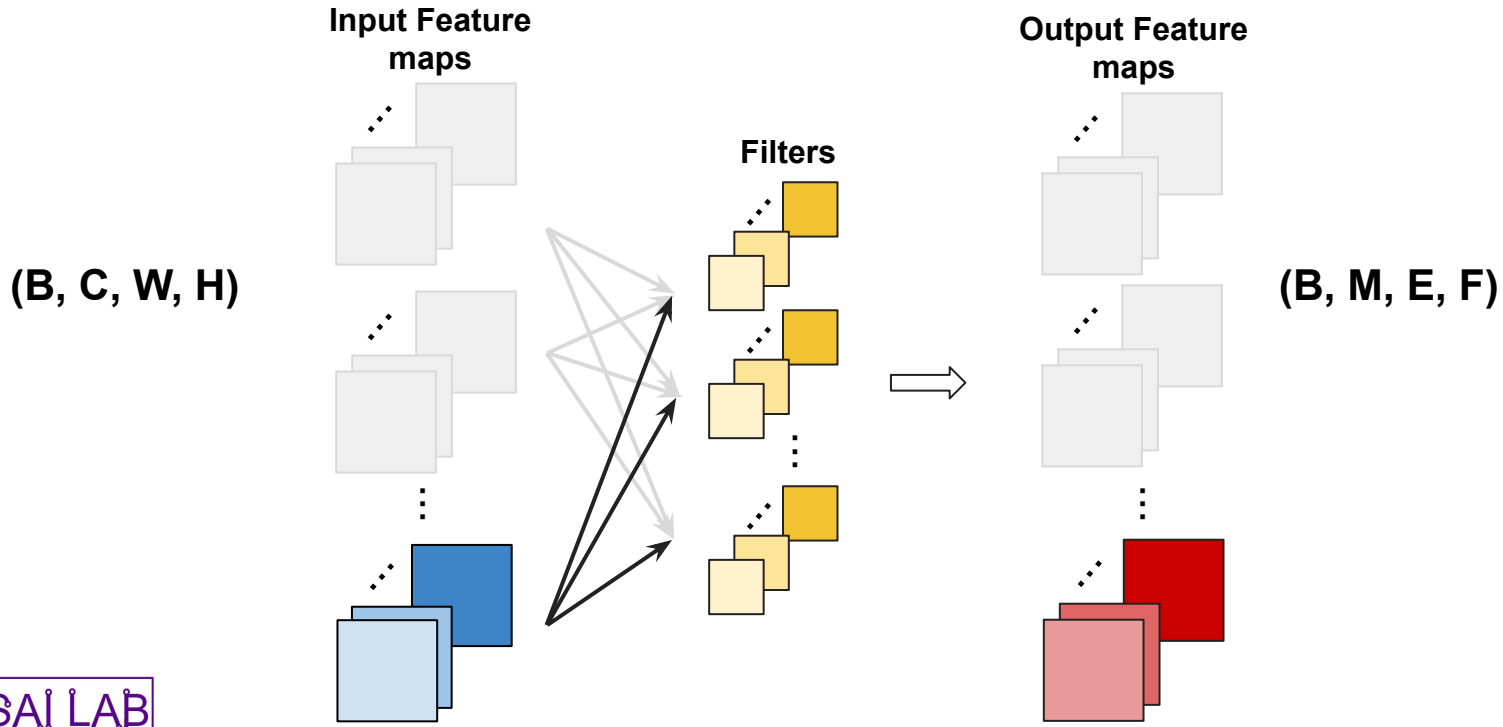


Convolution

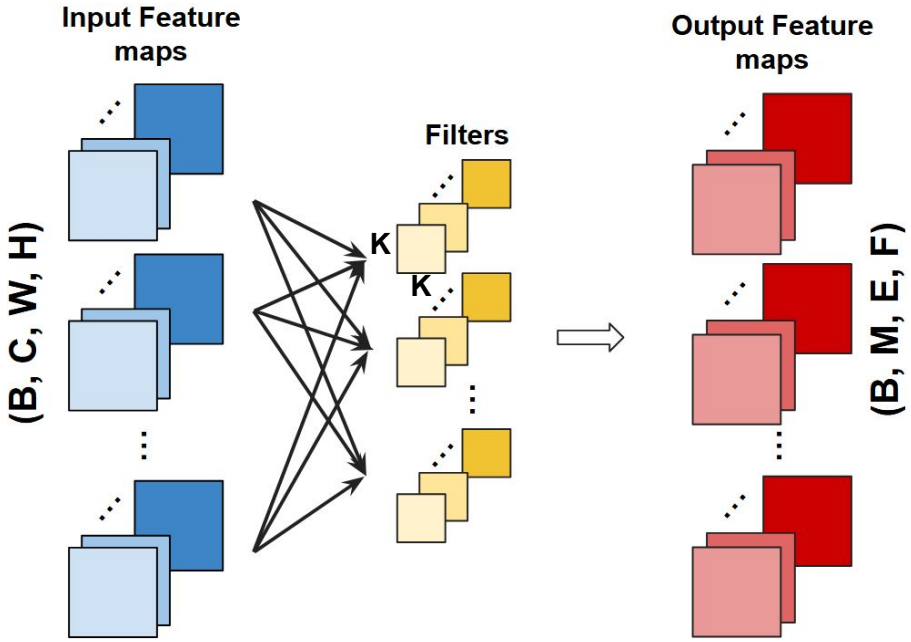


NYU SAI LAB

Convolution



Computational Cost: Standard Convolution

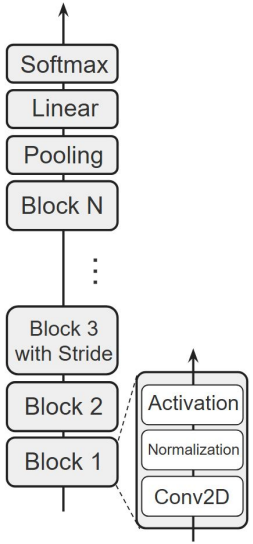


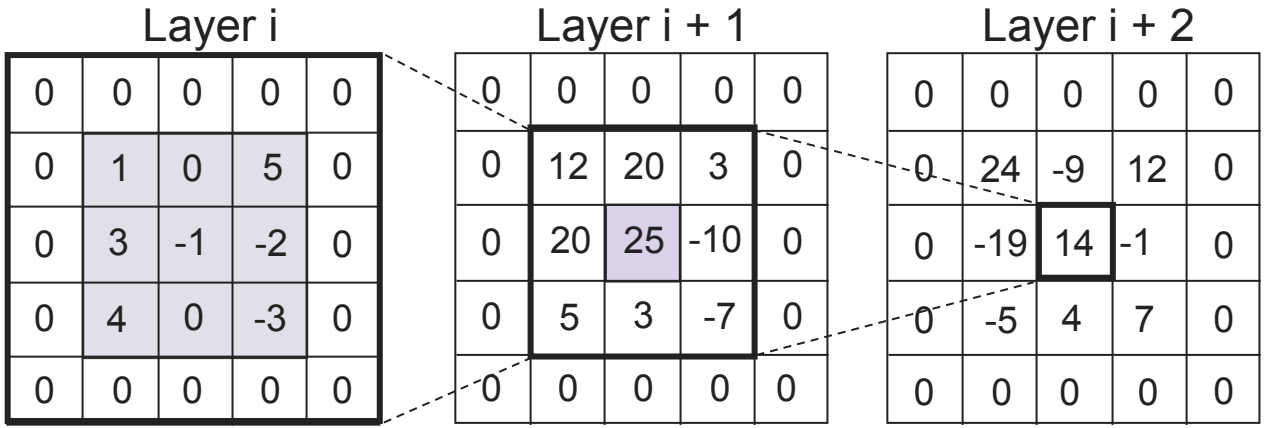
- Number of MACs: B×M×K×K×C×E×F
- Storage cost: 32×(M×C×K×K+B×C×H×W+B×M×E×F)

B: batch size
C: number of input channels
H,W: size of the input feature maps
M: number of weight filters
K: weight kernel size
E,F: size of the output feature maps

NYU SAI LAB

Receptive Field of CNN across Layers



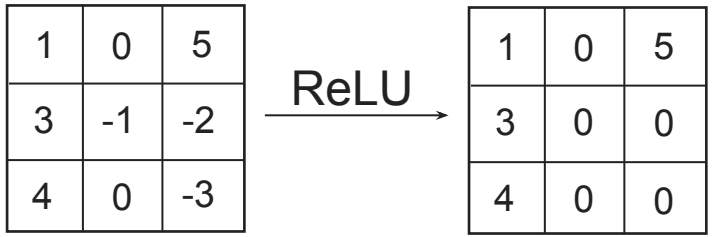


- Assume a kernel size of 3 by 3.
- Every elements at layer i is a function of the entire receptive fields of the previous layers.

NYU SAI LAB

Activation Functions: ReLU

- Rectifier linear operation (ReLU) applies an elementwise activation function to the output feature maps.
- This leaves the size of the output feature maps unchanged.
- f(x) = x if x > 0, f(x) = 0 otherwise.



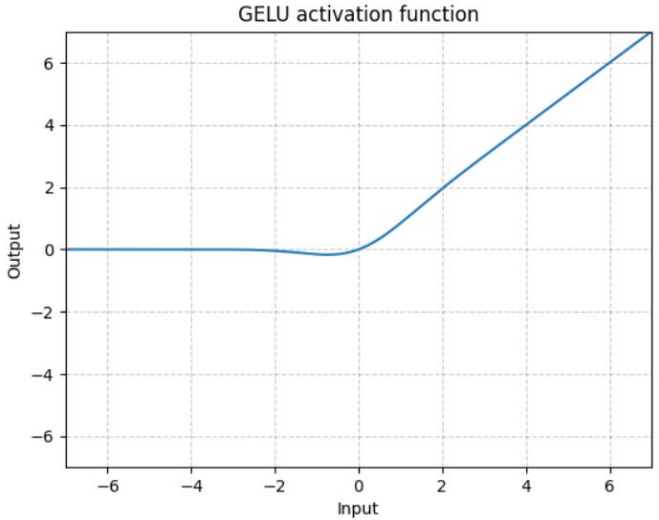
NYU SAI LAB

Activation Functions: GeLU

- Gaussian error linear unit (GeLU):

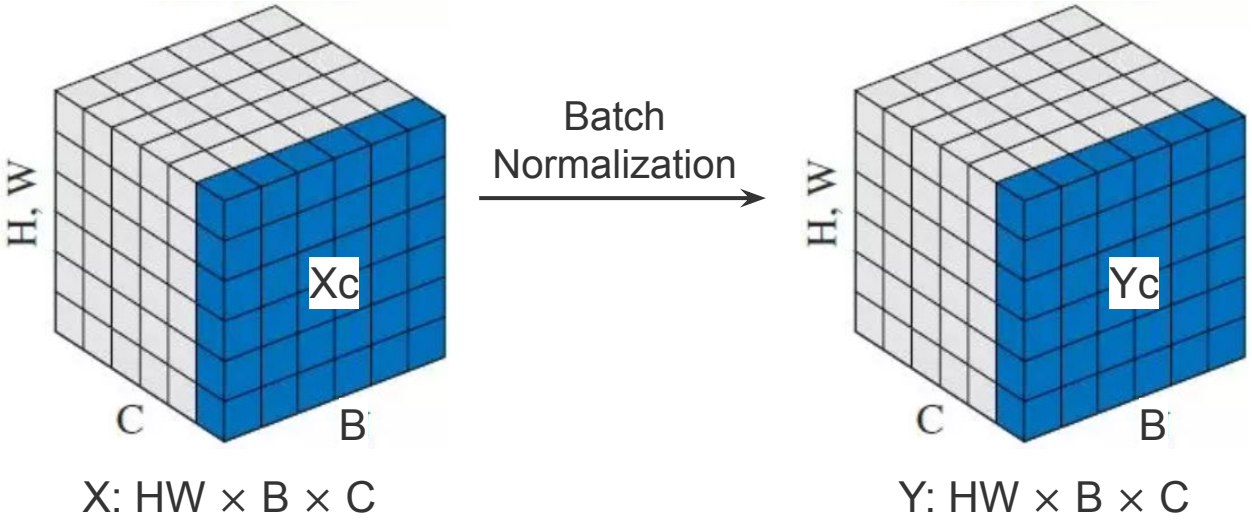
$$GeLU(x) = x\Phi(x)$$
$$\Phi(x) = P(y \leq x), \quad \text{where } \mathrm{Y} \sim N(0,1)$$
$$0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$$

- GeLU is increasingly being adopted in transformers and CNNs today.



Hendrycks, Dan, and Kevin Gimpel. "Gaussian error linear units (gelus)." *arXiv preprint arXiv:1606.08415* (2016).

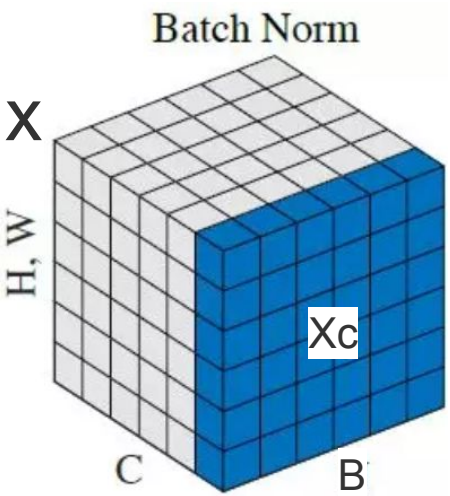
Batch Normalization



- **Batch Normalization (BatchNorm)** is a technique used in deep learning to improve the training stability and performance of neural networks.

Ioffe, Sergey. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).

Batch Normalization



X: HW × B × C

$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c \qquad \text{For each } c \in C$$

$$\alpha = \{\alpha_c\}, \beta = \{\beta_c\}, \mu = \{\mu_c\}, \sigma = \{\sigma_c\}$$

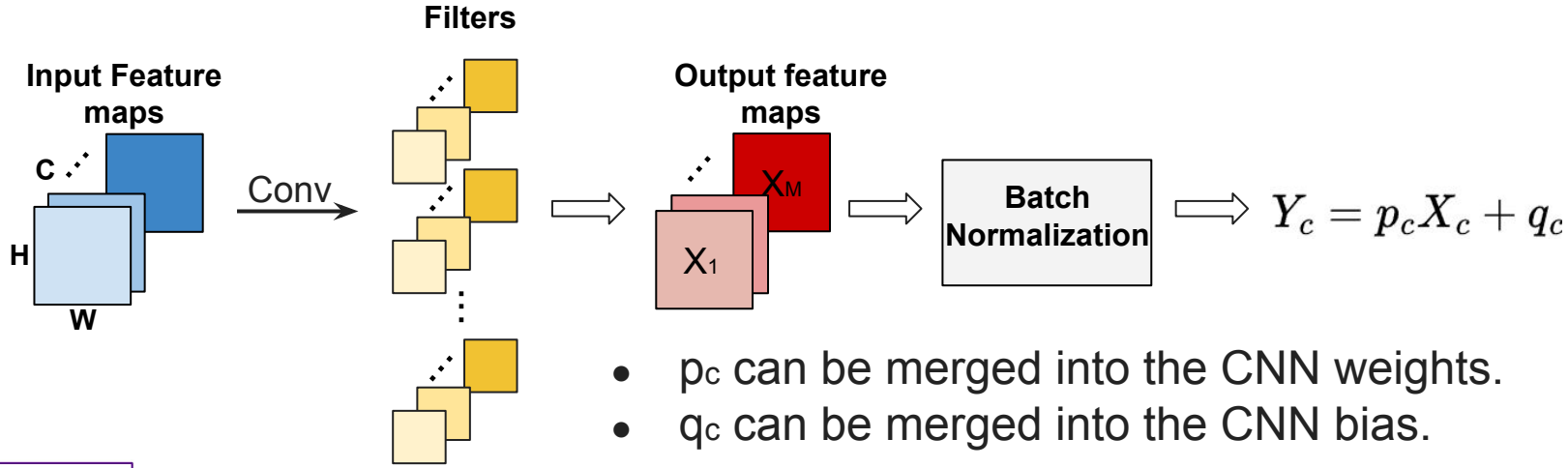
- For each channel c, we have:
  - Xc: (HW x B)
  - μ_c and δ_c are the mean and standard deviation of Xc.
  - α_c and β_c are learnable parameters
  - α_c, β_c, μ_c, δ_c are scalers
- Overall, we have:
  - μ, δ, α and β all have a length of of C
  - μ, δ, α and β are all fixed during the inference
  - μ, δ are statistics based on the training dataset

NYU SAI LAB

Batch Normalization: During Inference

- Given all the parameters are fixed, for each channel c, we have:

$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c = \frac{\alpha_c}{\sigma_c} X_c + (\beta_c - \frac{\alpha_c \mu_c}{\sigma_c}) \Longrightarrow Y_c = p_c X_c + q_c$$

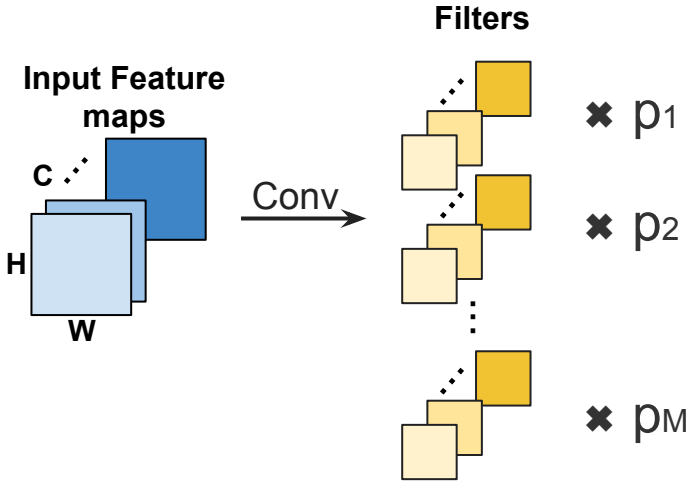


- p_c can be merged into the CNN weights.
- q_c can be merged into the CNN bias.

Batch Normalization

- For each channel c, we have:

$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c = \frac{\alpha_c}{\sigma_c} X_c + (\beta_c - \frac{\alpha_c \mu_c}{\sigma_c}) \Rightarrow Y_c = p_c X_c + q_c$$

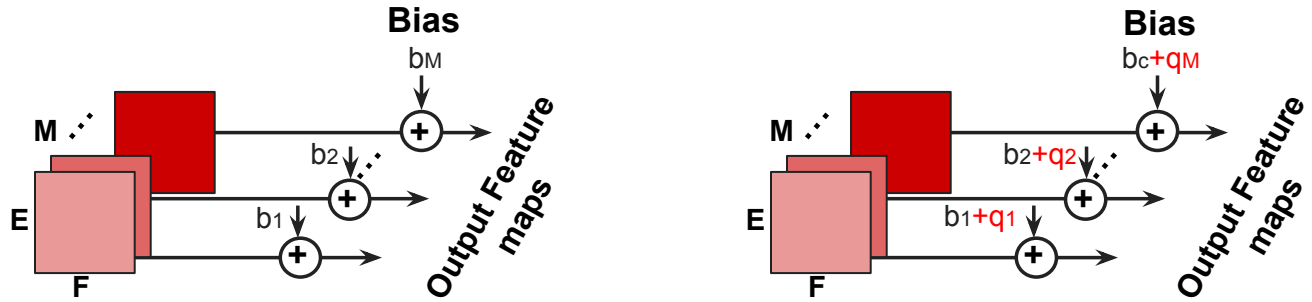


- We can fold in the p and q to the weights and bias of convolutional layer during inference and reduce the online computational cost.

Batch Normalization

- For each channel c, we have:

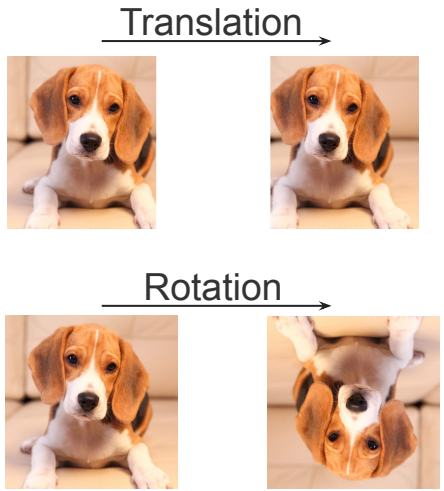
$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c = \frac{\alpha_c}{\sigma_c} X_c + (\beta_c - \frac{\alpha_c \mu_c}{\sigma_c}) \Rightarrow Y_c = p_c X_c + q_c$$



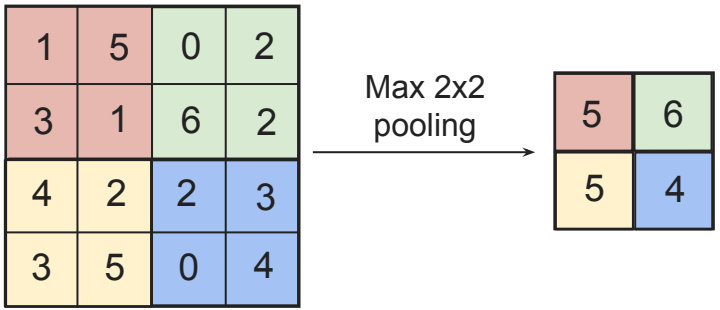
- We can fold in the p and q to the weights and bias of convolutional layer during inference and reduce the online computational cost.

Pooling

- Enhance the model invariance to spatial transformations such as translation and rotation, thereby reducing the risk of overfitting.
- Reduce the spatial size of the representation and reduce the amount of parameters and computation in the CNN.



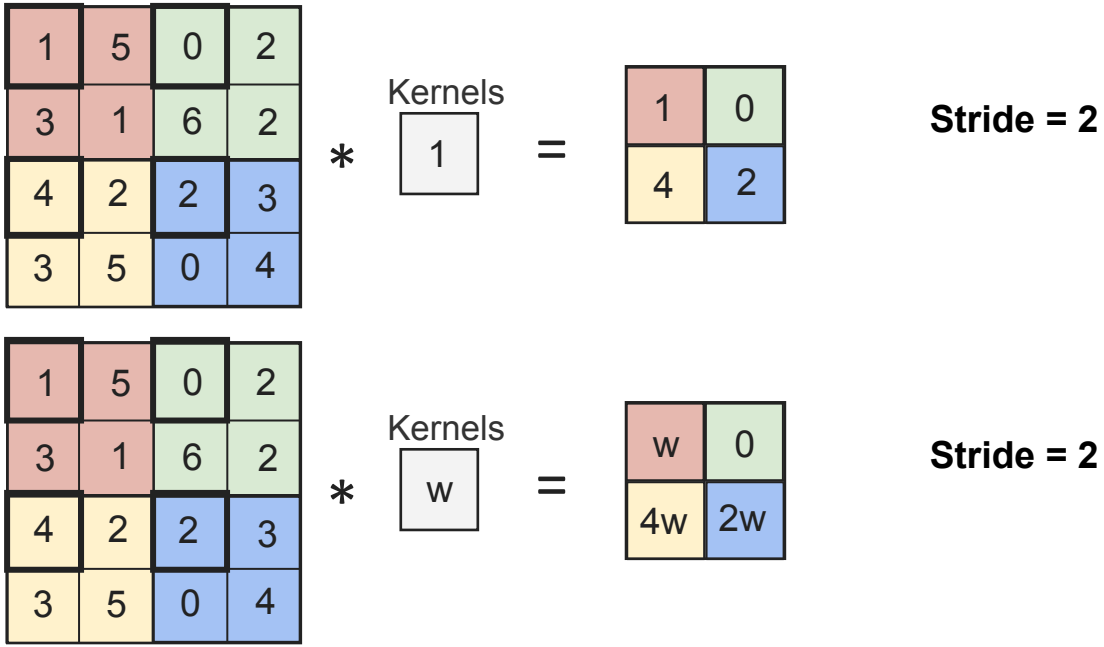
Should produce the same prediction result



NYU SAI LAB

Strided Convolution As Pooling Layer

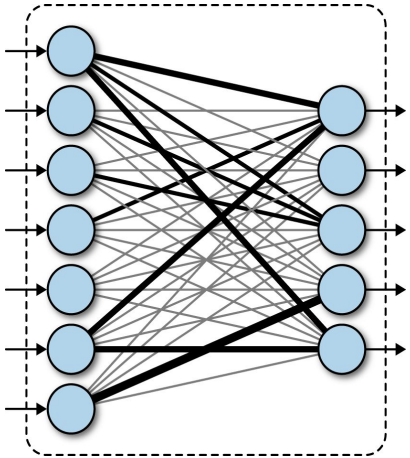
- Recent Neural Networks replace pooling layers with Strided Convolution



NYU SAI LAB

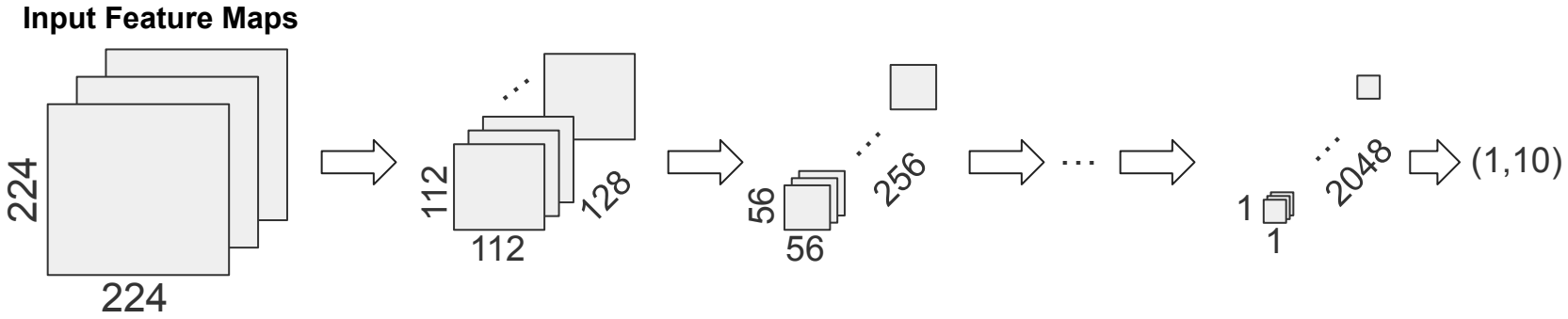
Fully Connected Layers

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular neural networks.



- Normally used in the last several layers to generate the classification results.

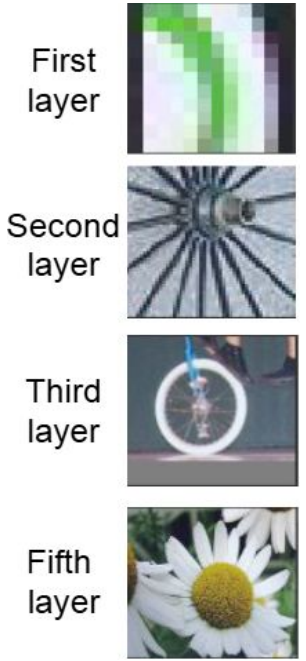
CNN Architecture for Image Classification Task



- For image classification task, during the forward propagation of CNN, the spatial size reduces while the number of channels increases.

Interpretation of Convolutional Features

- Each layer progressively extracts higher level features of the input image, until the last layer which aggregates all the high-level abstraction and makes a final decision.
- Early CNN layers tend to focus on detecting the local features (e.g., edge or corner in the image), whereas later layers usually look for the high-level abstractions (e.g., shapes of the object in the image)

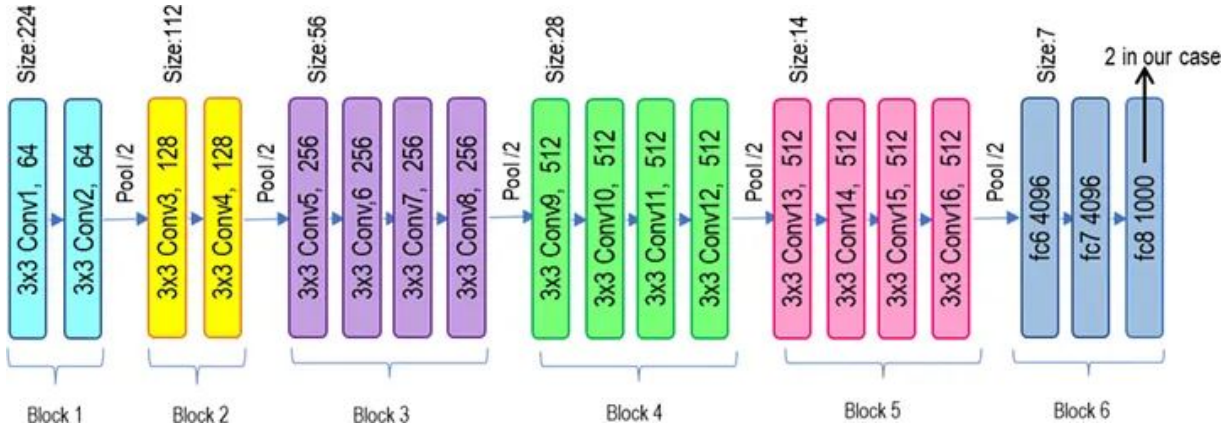


Yosinski, Jason, et al. "Understanding neural networks through deep visualization." *arXiv preprint arXiv:1506.06579* (2015).

Topics

- Convolutional Neural Network
  - Basic building blocks
  - Popular CNN architectures
    - VGG
    - ResNet
    - MobileNet
    - ShuffleNet
    - SqueezeNet
    - DenseNet
    - EfficientNet
    - ConvNext
    - ShiftNet
  - CNN architectures for other vision tasks
    - Image Segmentation, Object Detection

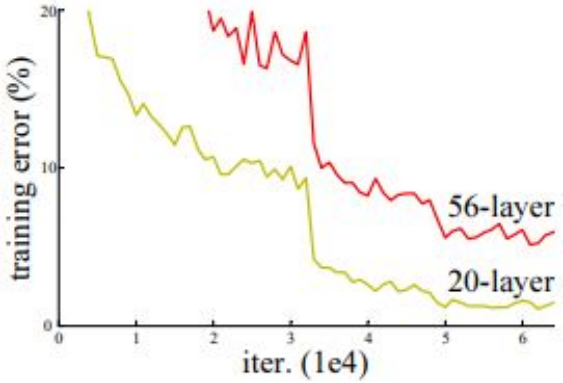
VGG

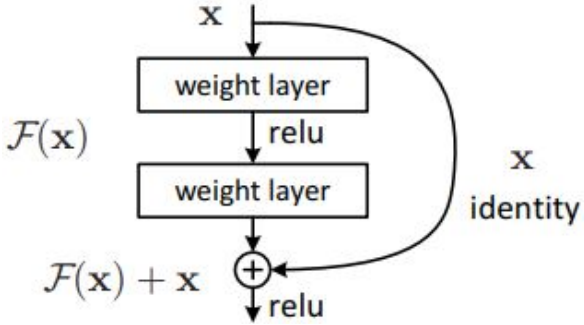


- The **key contribution** was demonstrating that **depth and the use of very small convolutional kernels (3×3)** were crucial for dramatically improving image recognition performance, establishing a simple and scalable architecture that became foundational in deep learning research.
- Achieves 75%-76% accuracy on ImageNet, which is much higher than other networks at that time (AlexNet: 62.5%).

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

ResNet

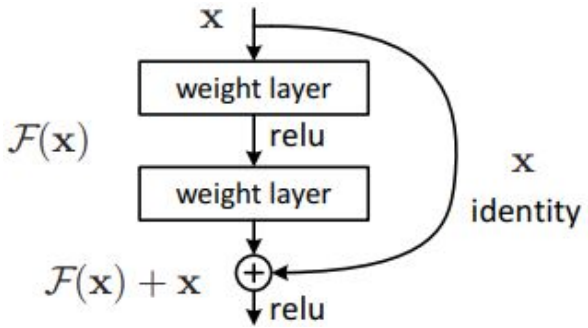


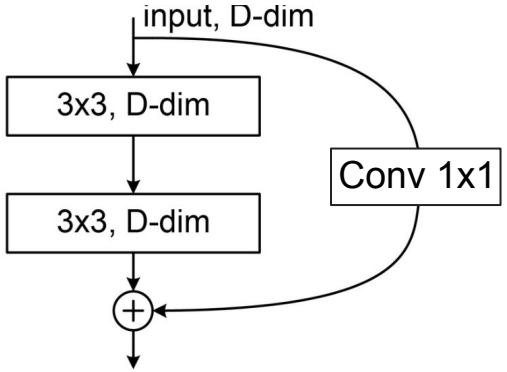


- When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly.
- By introducing the residual link, we reduce the complexity of the learning process by ensuring that the performance is at least as good as the shallower DNN.

He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

ResNet



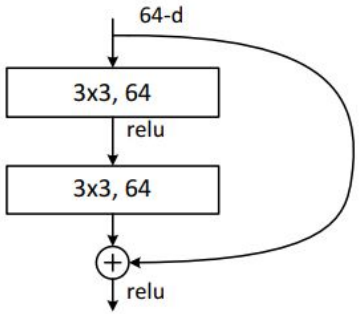


$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x} \qquad \mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$$

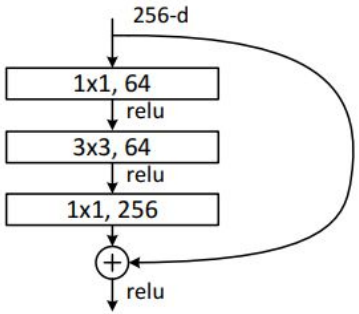
- A straightforward strided convolutional layer may also be added to both branches when subsampling the output.

ResNet Bottleneck Design

Total params:
3×3×64×64 +
3×3×64×64 =73728



Basic block



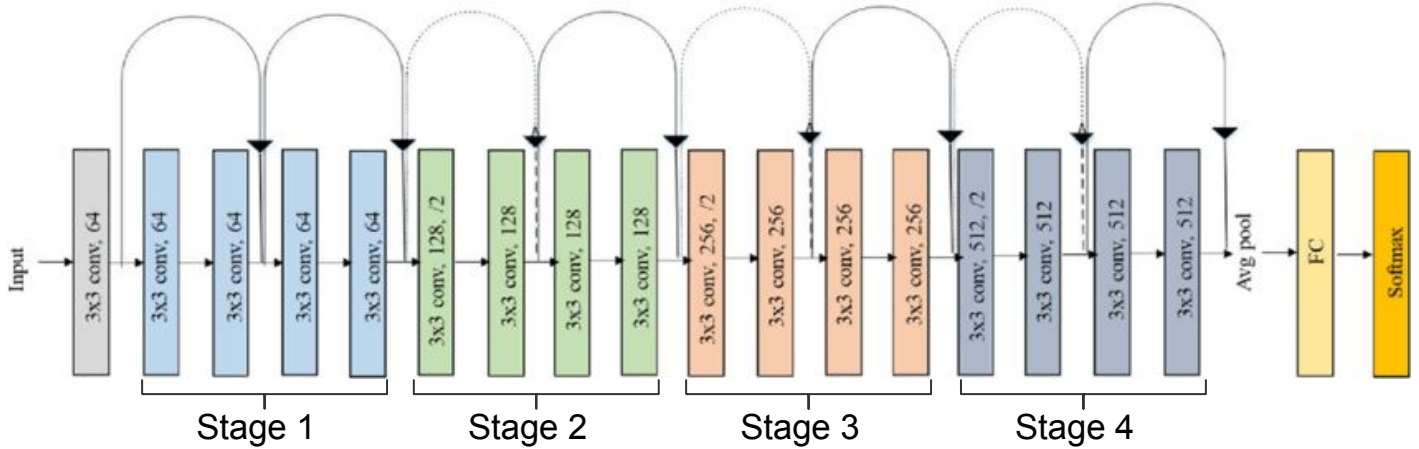
Bottleneck block

Total params:
1×1×256×64 + 3×3×64×64
+ 1×1×64×256 = 69632

- For deeper ResNet, the bottleneck block is used.
- The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers reduces the output channel dimension.

NYU SAI LAB

ResNet 18



- ResNet-18 is partitioned into several stages, across two consecutive stages, the output channels doubles, and the spatial size is 2x2 subsampled.

ResNet

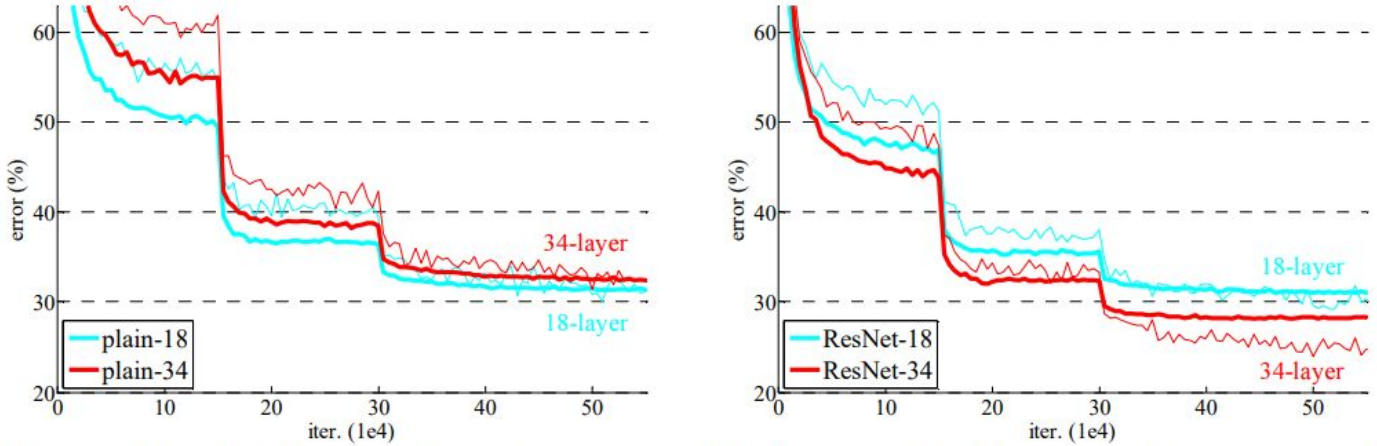


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

ResNet Performance

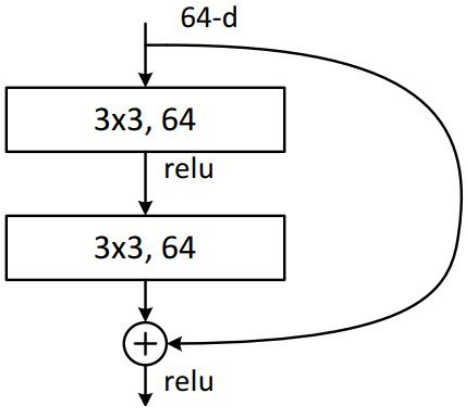
Performance on ImageNet

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Performance on CIFAR-10

method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

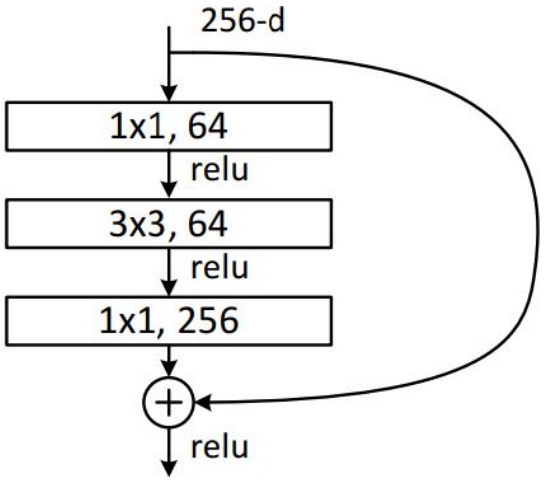
ResNet Implementation




```python
class BasicBlock(nn.Module):
  expansion = 1
  def __init__(self, in_planes, planes, stride=1):
    super(BasicBlock, self).__init__()
    self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
    self.bn1 = nn.BatchNorm2d(planes)
    self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
    self.bn2 = nn.BatchNorm2d(planes)

    self.shortcut = nn.Sequential()
    if stride != 1 or in_planes != self.expansion*planes:
      self.shortcut = nn.Sequential(
          nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=stride, bias=False),
          nn.BatchNorm2d(self.expansion*planes)
      )

  def forward(self, x):
    out = F.relu(self.bn1(self.conv1(x)))
    out = self.bn2(self.conv2(out))
    out += self.shortcut(x)
    out = F.relu(out)
    return out
```

NYU SAI LAB

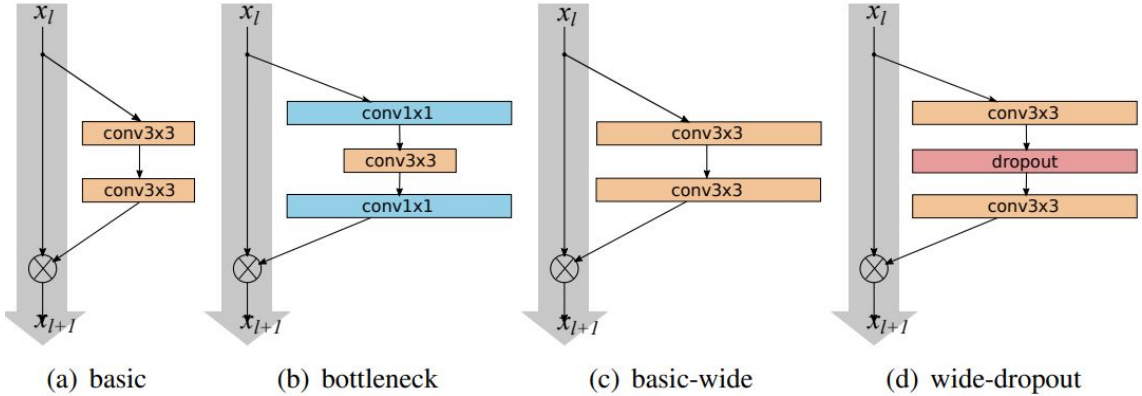
ResNet Implementation




```python
class BottleNeck(nn.Module):
  expansion = 4

  def __init__(self, in_planes, planes, stride=1):
    super(BottleNeck, self).__init__()
    self.conv1 = nn.Conv2d(in_planes , planes, kernel_size=1, bias=False)
    self.bn1 = nn.BatchNorm2d(planes)
    self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
    self.bn2 = nn.BatchNorm2d(planes)
    self.conv3 = nn.Conv2d(planes, self.expansion*planes, kernel_size=1, bias=False)
    self.bn3 = nn.BatchNorm2d(self.expansion*planes)

    self.shortcut = nn.Sequential()
    if stride != 1 or in_planes != self.expansion*planes :
      self.shortcut = nn.Sequential(
          nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=stride, bias=False),
          nn.BatchNorm2d(self.expansion*planes)
      )

  def forward(self, x):
    out = F.relu(self.bn1(self.conv1(x)))
    out = F.relu(self.bn2(self.conv2(out)))
    out = self.bn3(self.conv3(out))
    out += self.shortcut(x)
    out = F.relu(out)
    return out
```

NYU SAI LAB

Wide ResNet



- How ResNet should scale? Increase depth or increase width?
- Wide networks with only 16 layers can significantly outperform 1000-layer deep networks on CIFAR, as well as that 50-layer outperform 152-layer on ImageNet.
- The main power of residual networks is in residual blocks, and not in extreme depth as claimed earlier.

Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." *arXiv preprint arXiv:1605.07146* (2016).

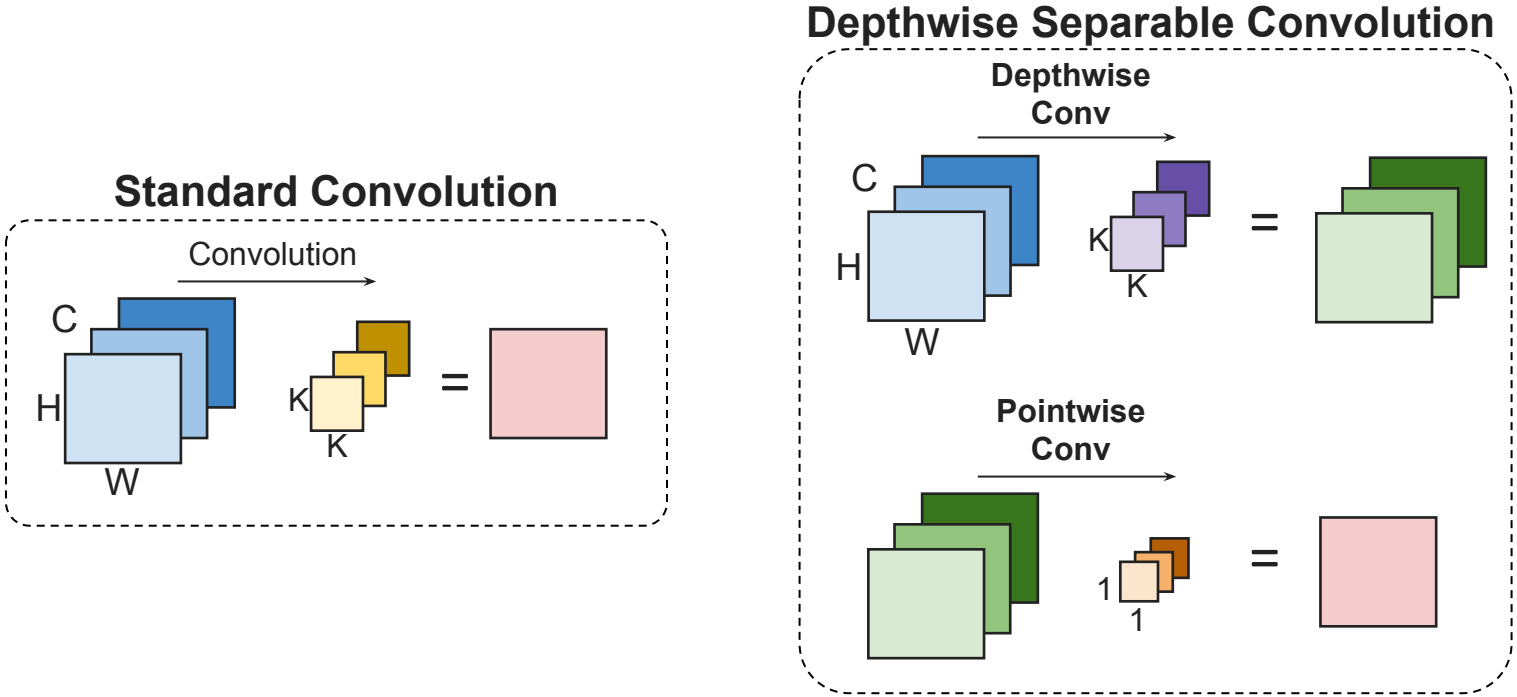
Wide ResNet

depth	k	# params	CIFAR-10	CIFAR-100
40	1	0.6M	6.85	30.89
40	2	2.2M	5.33	26.04
40	4	8.9M	4.97	22.89
40	8	35.7M	4.66	-
28	10	36.5M	**4.17**	20.50
28	12	52.5M	4.33	**20.43**
22	8	17.2M	4.38	21.22
22	10	26.8M	4.44	20.75
16	8	11.0M	4.81	22.07
16	10	17.1M	4.56	21.59

- ResNet with wider architecture achieves a better performance than deeper architectures.

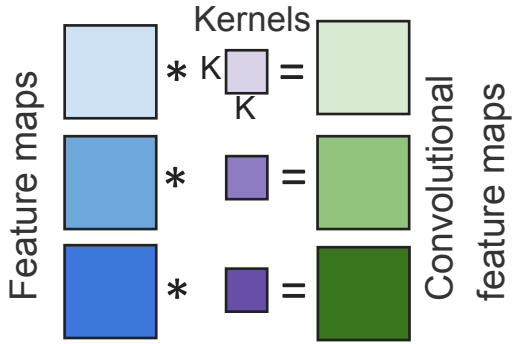
Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." *arXiv preprint arXiv:1605.07146* (2016).

MobileNet



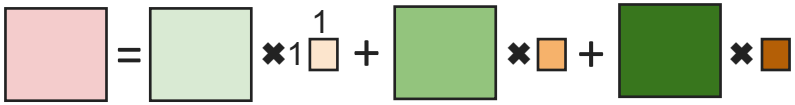
Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." *arXiv preprint arXiv:1704.04861* (2017).

MobileNet



Step 1 Depthwise Convolution

- Each kernel moves across the spatial dimensions of feature maps in the input activations, analyzing the information within those spatial dimensions.

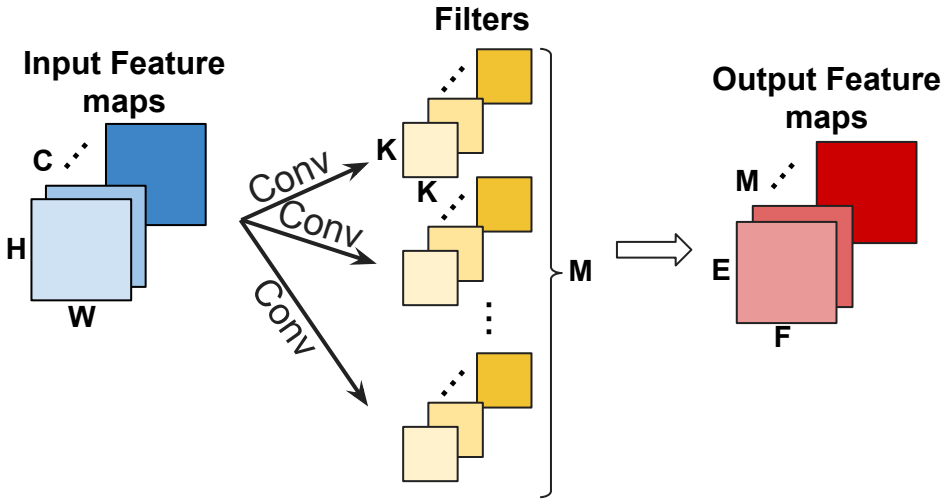


Step 2 Pointwise Convolution

- The information from each feature maps are then aggregated by multiplying with the weight in the pointwise conv kernel and summing the Convolutional feature maps together.
- A bias may be introduced.

NYU SAI LAB

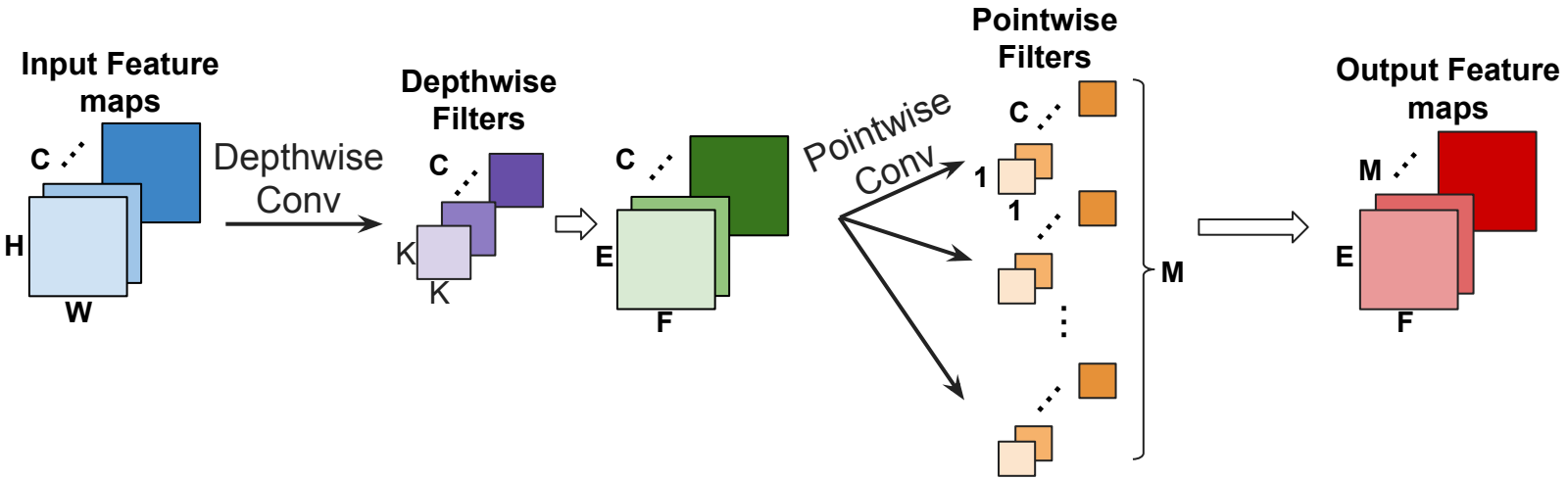
Standard Convolution



- Number of MACs: M×K×K×C×E×F
- Storage cost: 32×(M×C×K×K+C×H×W+M×E×F)

C: number of input channels
H,W: size of the input feature maps
M: number of weight filters
K: weight kernel size
E,F: size of the output feature maps

Depthwise Separable Convolution



- Number of MACs: K×K×C×E×F + M×C×E×F
- Storage cost: 32×(C×H×W+C×K×K+C×E×F+M×C+M×E×F)

NYU SAI LAB

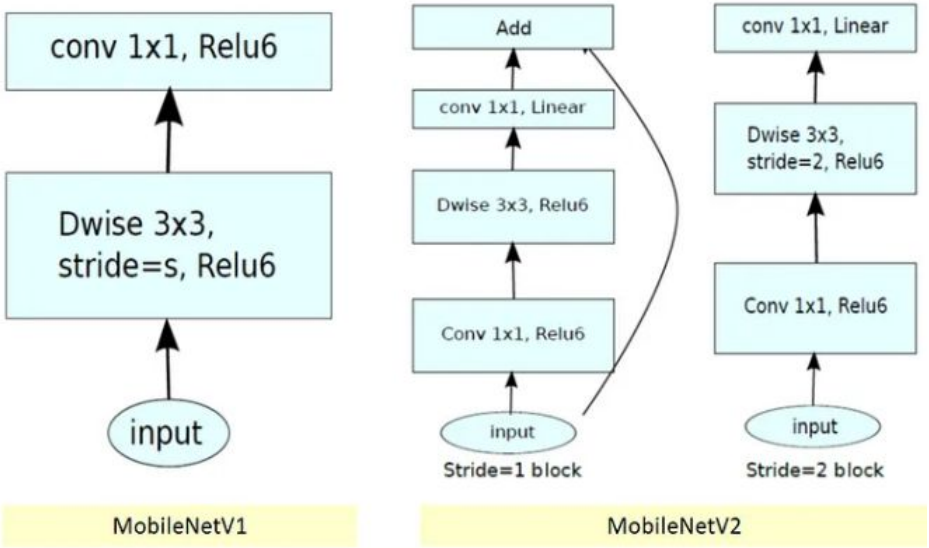
Why Depthwise Conv is Cheaper?

- Number of MACs for depthwise separable Conv: **K×K×C×E×F + M×C×E×F**
- Number of MACs for standard Conv: **M×K×K×C×E×F**
- When M is large the computational saving is about K×K (9) times.

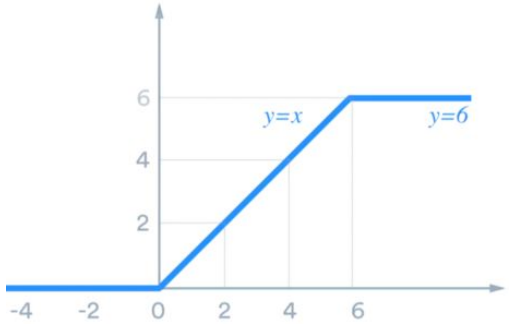
- With a batch size of B, number of MACs are:
- Number of MACs: B×K×K×C×E×F + B×M×C×E×F
- Storage cost: 32×(B×C×H×W+C×K×K+B×C×E×F+M×C+B×M×E×F)

NYU SAI LAB

MobileNet-V2

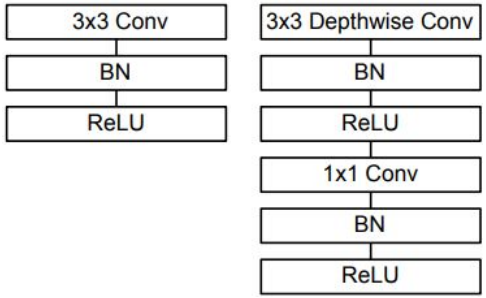


- Add residual link between the blocks.
- Adopt ReLU6 replace ReLU.



Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

MobileNet Implementation

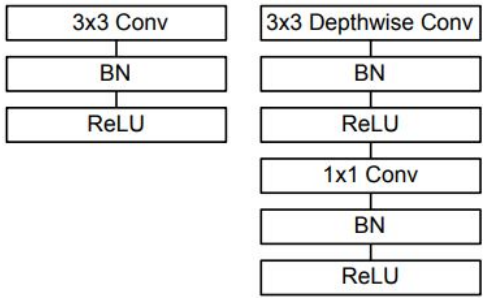



```
self.depthwise_conv = nn.Conv2d(
    in_channels,
    in_channels,
    (3, 3),
    stride=stride,
    padding=1,
    groups=in_channels,
)
self.bn1 = nn.BatchNorm2d(in_channels)

self.relu1 = nn.ReLU6() if use_relu6 else nn.ReLU()

# Pointwise conv
self.pointwise_conv = nn.Conv2d(in_channels, out_channels, (1, 1))
self.bn2 = nn.BatchNorm2d(out_channels)

self.relu2 = nn.ReLU6() if use relu6 else nn.ReLU()
```

```
def forward(self, x):
    """Perform forward pass."""

    x = self.depthwise_conv(x)
    x = self.bn1(x)
    x = self.relu1(x)
    x = self.pointwise_conv(x)
    x = self.bn2(x)
    x = self.relu2(x)

    return x
```

NYU SAI LAB

MobileNet V2 Implementation

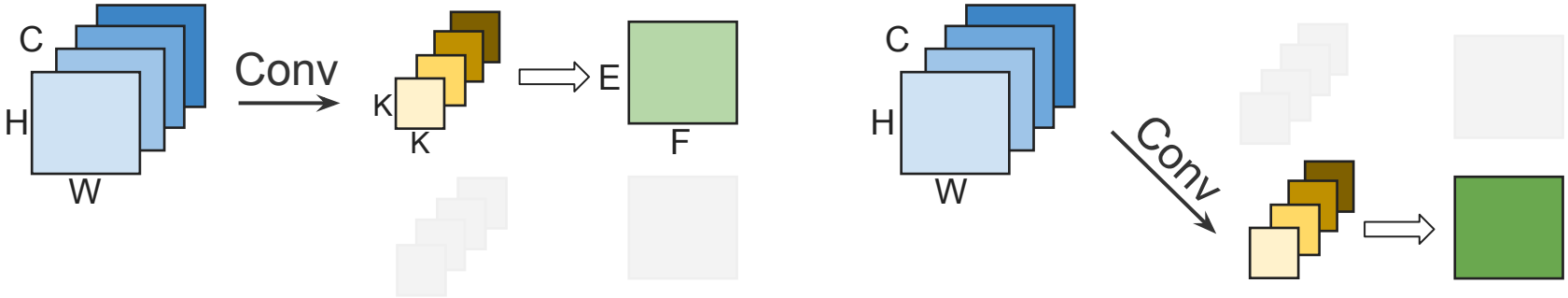



```
self.conv1 = (
    ConvNormReLUBlock(in_channels, hidden_channels, (1, 1))
    if in_channels != hidden_channels
    else nn.Identity()
)
self.depthwise_conv = ConvNormReLUBlock(
    hidden_channels,
    hidden_channels,
    (3, 3),
    stride=stride,
    padding=1,
    groups=hidden_channels,
)
self.conv2 = ConvNormReLUBlock(
    hidden_channels, out_channels, (1, 1), activation=nn.Identity
)
```

```
def forward(self, x):
    """Perform forward pass."""

    identity = x

    x = self.conv1(x)
    x = self.depthwise_conv(x)
    x = self.conv2(x)

    if self.residual:
        x = torch.add(x, identity)

    return x
```

NYU SAI LAB

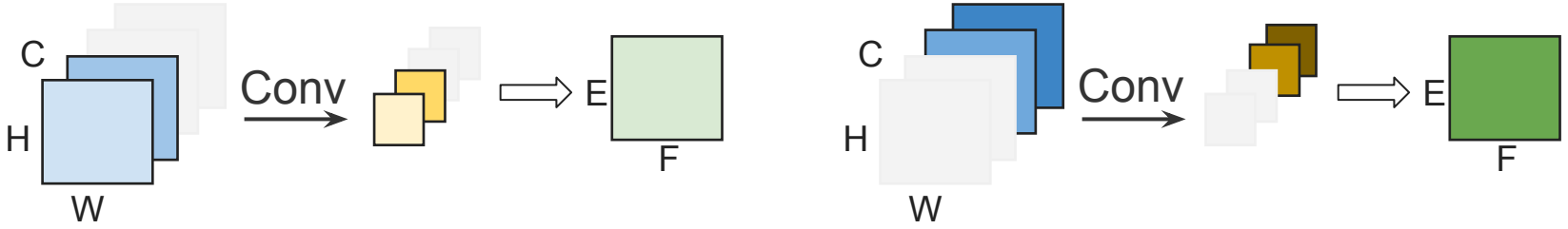
Group Convolution



- The original MAC: E×F×K×K×C×M

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." *Communications of the ACM* 60.6 (2017): 84-90.

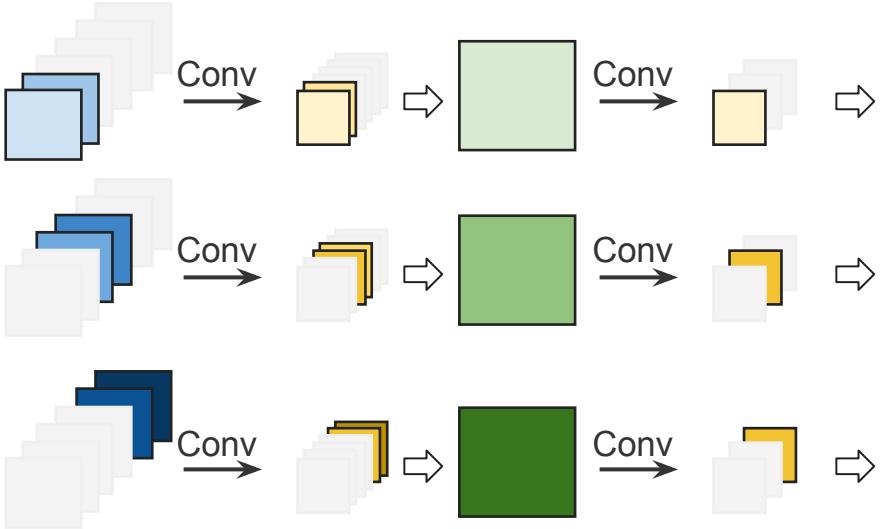
Group Convolution



- Group size = 2
- Each group of feature maps within the input only convolved with partial weight kernels.
- This will lead to a large saving on memory consumption and computational cost.
- The number of MAC: $E \times F \times K \times K \times C \times M/G$

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." *Communications of the ACM* 60.6 (2017): 84-90.

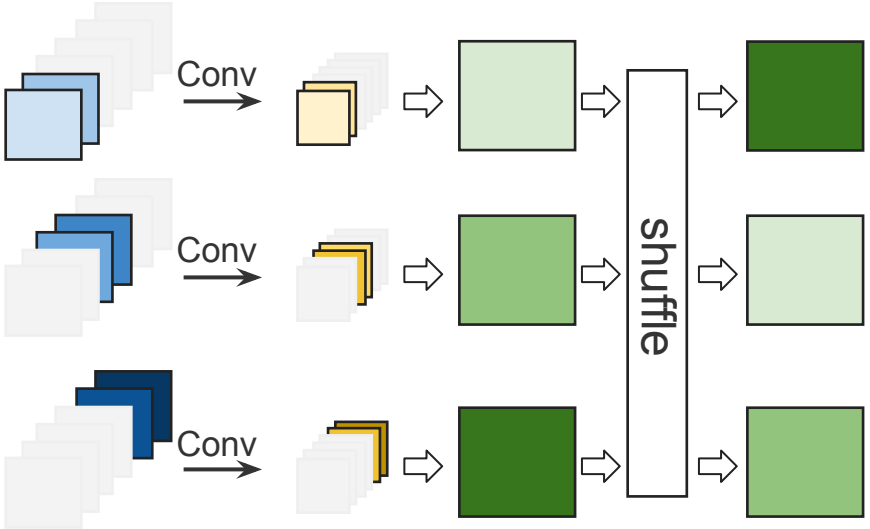
ShuffleNet



- Group convolution prevents feature maps from different groups from exchanging information.

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

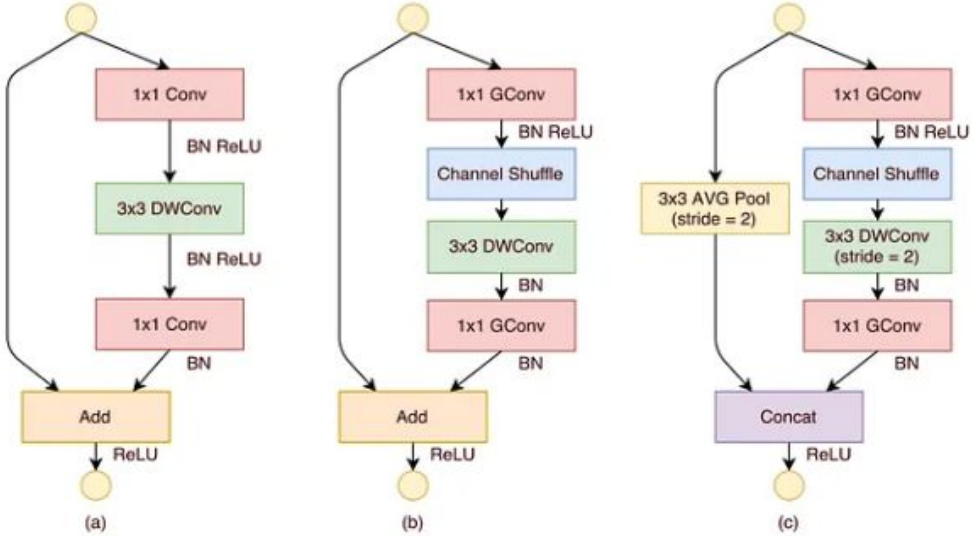
ShuffleNet



- The shuffle operation is used to exchange the information across the groups.
- The shuffle operation with group convolution can replace the conventional full-channel convolution without noticeable accuracy degradation.
- A predetermined pattern is applied for the shuffling operations.

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

ShuffleNet



NYU SAI LAB

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

ShuffleNet

Model	Cls err. (%, no shuffle)	Cls err. (%, shuffle)	Δ err. (%)
ShuffleNet 1x ($g = 3$)	34.5	**32.6**	1.9
ShuffleNet 1x ($g = 8$)	37.6	**32.4**	5.2
ShuffleNet 0.5x ($g = 3$)	45.7	**43.2**	2.5
ShuffleNet 0.5x ($g = 8$)	48.1	**42.3**	5.8
ShuffleNet 0.25x ($g = 3$)	56.3	**55.0**	1.3
ShuffleNet 0.25x ($g = 8$)	56.5	**52.7**	3.8

Table 3. ShuffleNet with/without channel shuffle (*smaller number represents better performance*)

- G is the group size, a× is the scaling factor on number of channels.
- Shuffling operation can greatly improve the accuracy.

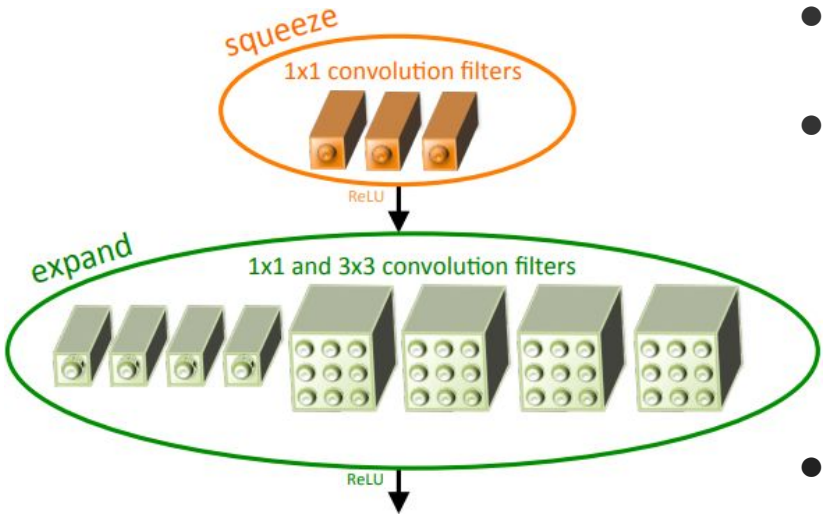
ShuffleNet

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet 2× ($g = 3$)	524	**26.3**	3.1
ShuffleNet 2× (with *SE*[13], $g = 3$)	527	**24.7**	4.7
0.75 MobileNet-224	325	31.6	-
ShuffleNet 1.5× ($g = 3$)	292	**28.5**	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet 1× ($g = 8$)	140	**32.4**	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet 0.5× ($g = 4$)	38	**41.6**	7.8
ShuffleNet 0.5× (shallow, $g = 3$)	40	42.8	6.6

Table 5. ShuffleNet vs. MobileNet [12] on ImageNet Classification

- Under the same level of computational complexity, shufflenet is better than MobileNet.

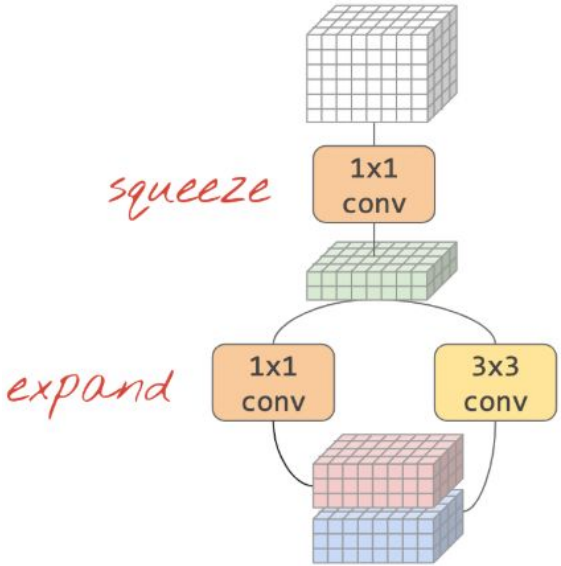
SqueezeNet

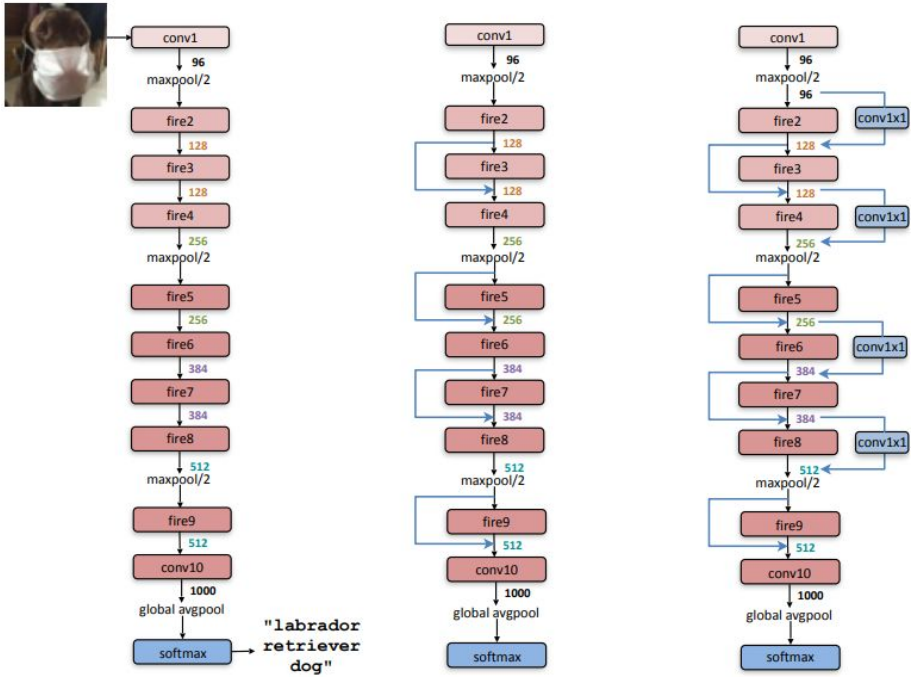


- Achieves great accuracy with 50x smaller parameters than other baselines (4.8MB).
- Some strategies:
  - Replace 3x3 filters with 1x1 filters.
  - Decrease the number of input channels to 3x3 filters.
  - Downsample late in the network so that convolution layers have large activation maps.
- Aims to reduce the CNN parameter size, not computational cost.

Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).

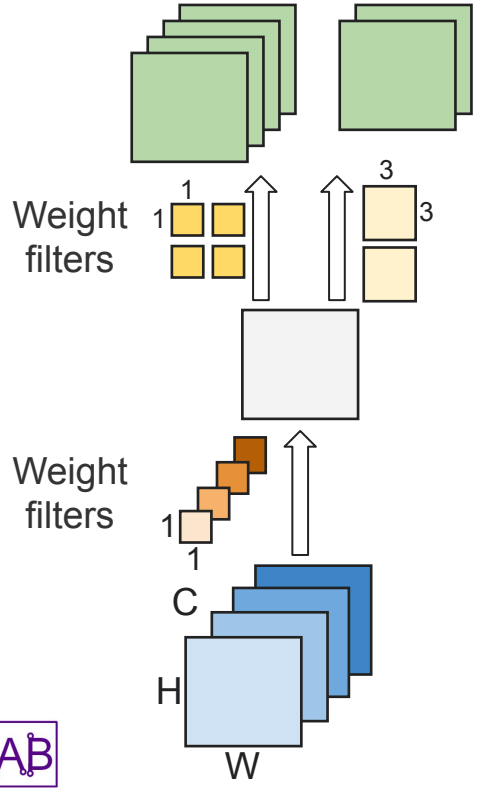
SqueezeNet





Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).

SqueezeNet




```
class fire(nn.Module):
    def __init__(self, inplanes, squeeze_planes, expand_planes):
        super(fire, self).__init__()
        self.conv1 = nn.Conv2d(inplanes, squeeze_planes, kernel_size=1, stride=1)
        self.bn1 = nn.BatchNorm2d(squeeze_planes)
        self.relu1 = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(squeeze_planes, expand_planes, kernel_size=1, stride=1)
        self.bn2 = nn.BatchNorm2d(expand_planes)
        self.conv3 = nn.Conv2d(squeeze_planes, expand_planes, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(expand_planes)
        self.relu2 = nn.ReLU(inplace=True)

        # using MSR initilization
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.in_channels
                m.weight.data.normal_(0, math.sqrt(2./n))

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu1(x)
        out1 = self.conv2(x)
        out1 = self.bn2(out1)
        out2 = self.conv3(x)
        out2 = self.bn3(out2)
        out = torch.cat([out1, out2], 1)
        out = self.relu2(out)
        return out
```

NYU SAI LAB

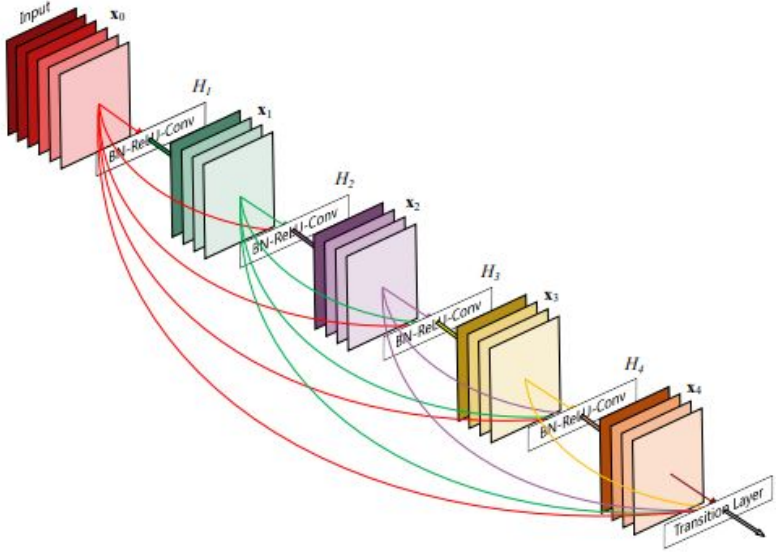
SqueezeNet

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

- Achieve a comparable performance as AlexNet, but still suboptimal compare against other architectures.
- ResNet 50: 100MB, Vision Transformer base> 300MB.

Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).

DenseNet



- ResNet:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}$$

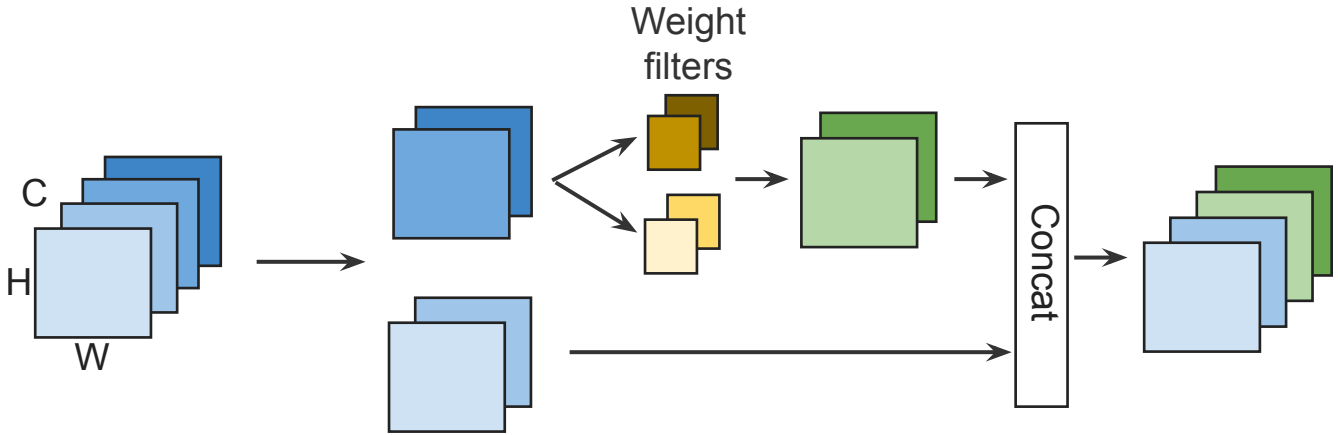
- DeseNet:

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$$

- H(.) is the function of batch normalization, followed by ReLU and 3x3 Convolution.

Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

DenseNet



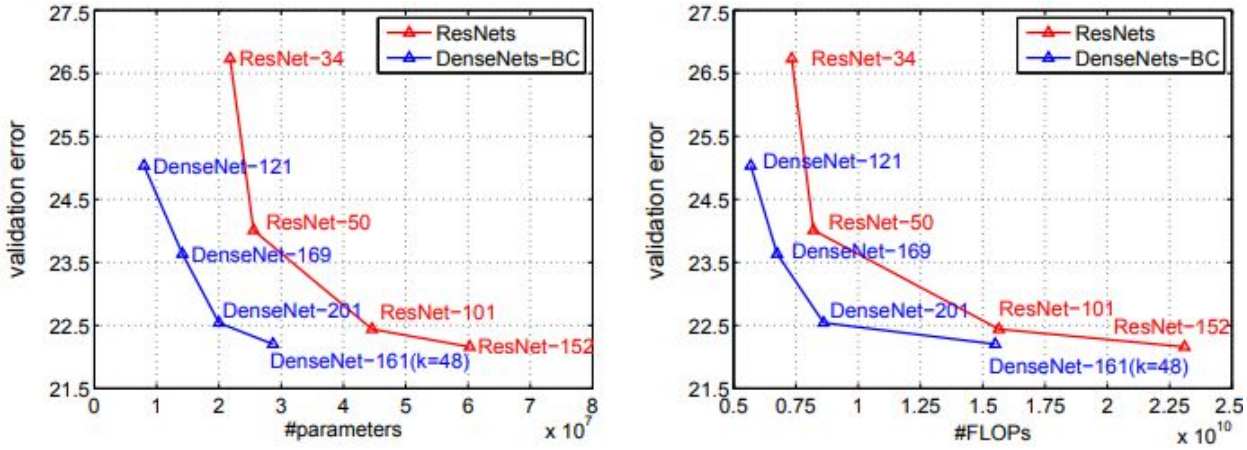
Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

DenseNet Implementation

```
class BasicBlock(nn.Module):
    def __init__(self, in_planes, out_planes, dropRate=0.0):
        super(BasicBlock, self).__init__()
        self.bn1 = nn.BatchNorm2d(in_planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=1,
                               padding=1, bias=False)
        self.droprate = dropRate
    def forward(self, x):
        out = self.conv1(self.relu(self.bn1(x)))
        if self.droprate > 0:
            out = F.dropout(out, p=self.droprate, training=self.training)
        return torch.cat([x, out], 1)
```

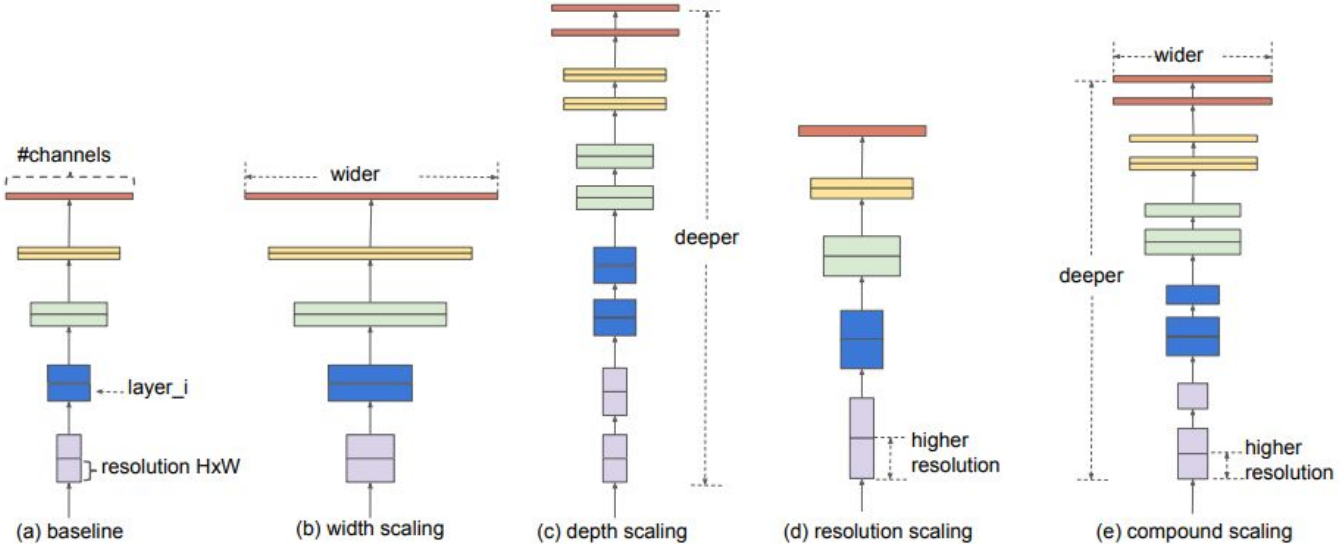
```
class BottleneckBlock(nn.Module):
    def __init__(self, in_planes, out_planes, dropRate=0.0):
        super(BottleneckBlock, self).__init__()
        inter_planes = out_planes * 4
        self.bn1 = nn.BatchNorm2d(in_planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = nn.Conv2d(in_planes, inter_planes, kernel_size=1, stride=1,
                               padding=0, bias=False)
        self.bn2 = nn.BatchNorm2d(inter_planes)
        self.conv2 = nn.Conv2d(inter_planes, out_planes, kernel_size=3, stride=1,
                               padding=1, bias=False)
        self.droprate = dropRate
    def forward(self, x):
        out = self.conv1(self.relu(self.bn1(x)))
        if self.droprate > 0:
            out = F.dropout(out, p=self.droprate, inplace=False, training=self.training)
        out = self.conv2(self.relu(self.bn2(out)))
        if self.droprate > 0:
            out = F.dropout(out, p=self.droprate, inplace=False, training=self.training)
        return torch.cat([x, out], 1)
```

DenseNet



Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

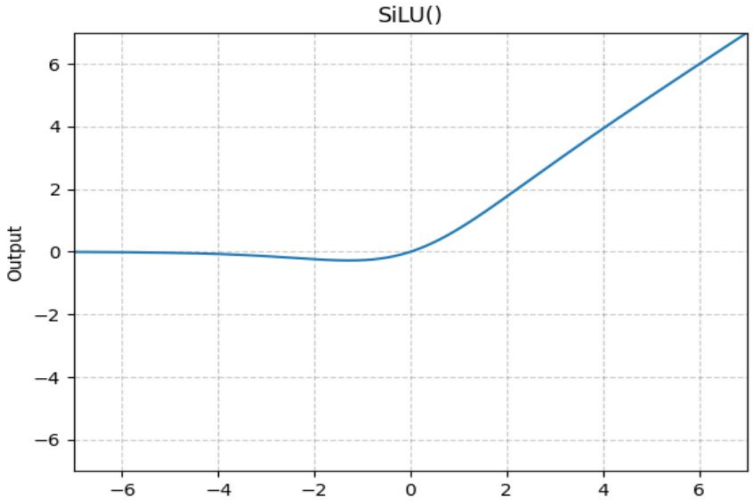
EfficientNet

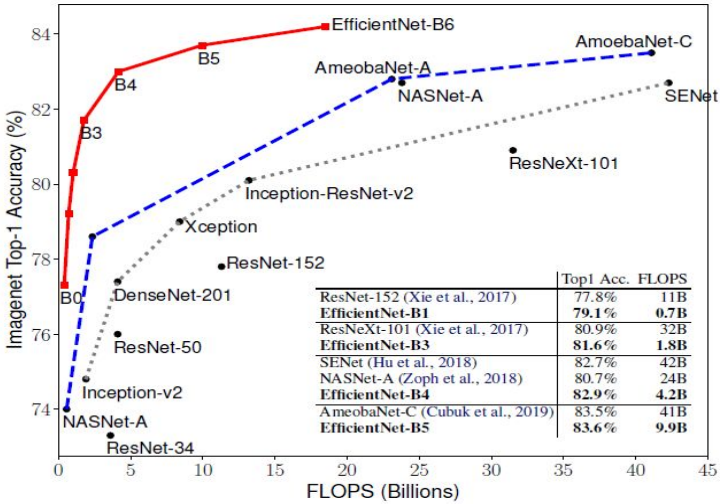


- It is critical to balance all dimensions of network width/depth/resolution, and surprisingly such balance can be achieved by simply scaling each of them with constant ratio.

Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." *International conference on machine learning*. PMLR, 2019.

EfficientNet



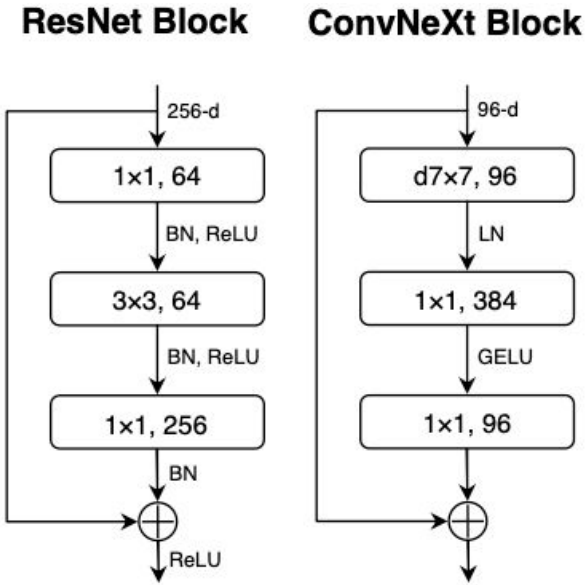


- SiLU is used in the EfficientNet architecture.
- SiLU(x) = x∗σ(x)

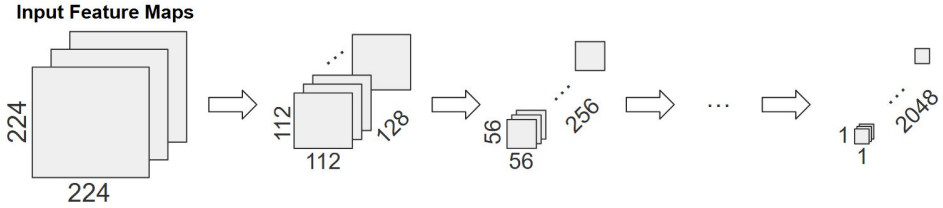
NYU SAI LAB

Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." *International conference on machine learning*. PMLR, 2019.

ConvNext



- Leverage the insight of vision transformer (Swin-T) to enhance the performance of CNN.
- Some major changes to change ResNet 50 to ConvNext 50:
  - Change number of blocks in each stage from (3, 4, 6, 3) in ResNet-50 to (3, 3, 9, 3).
  - Use depthwise separable convolution
  - Large convolutional kernel.
  - Replacing ReLU with GELU
  - Substituting BN with LN.

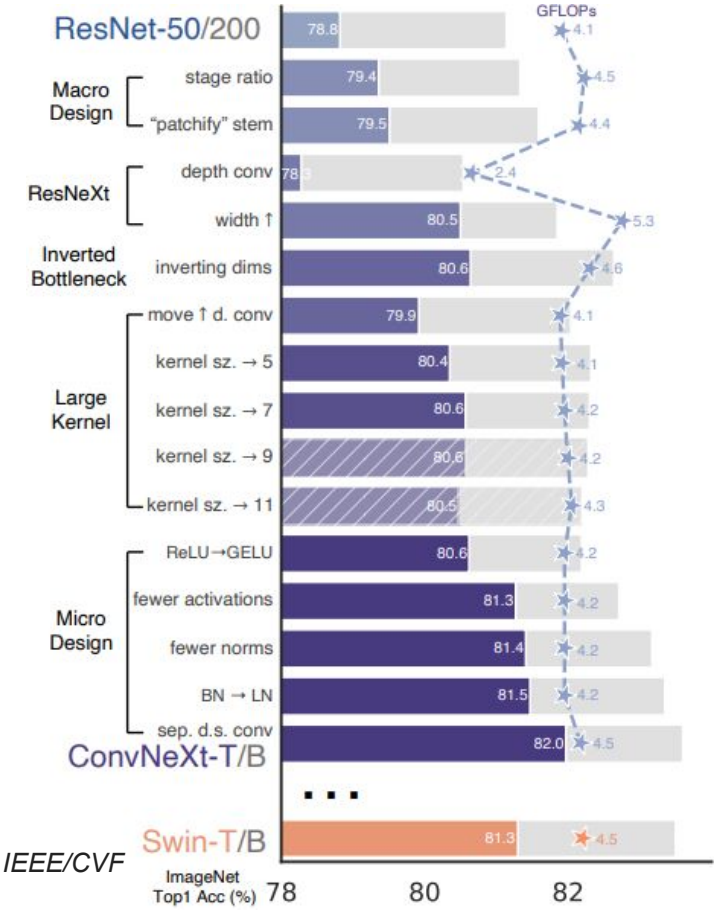


NYU SAI LAB

Liu, Zhuang, et al. "A convnet for the 2020s." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022.

ConvNext

- For larger Swin Transformers, the ratio is 1:1:9:1. Following the design → we adjust the number of blocks in each stage from (3, 4, 6, 3) in ResNet-50 to (3, 3, 9, 3).
- One of the most distinguishing aspects of vision Transformers is their non-local self-attention, which enables each layer to have a global receptive field, so we increase the window size to 7×7.
- Replacing ReLU with GELU: One discrepancy between NLP and vision architectures is the specifics of which activation functions to use.



Liu, Zhuang, et al. "A convnet for the 2020s." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022.

ConvNext

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
	ImageNet-1K trained models				
• RegNetY-16G [54]	224^2	84M	16.0G	334.7	82.9
• EffNet-B7 [71]	600^2	66M	37.0G	55.1	84.3
• EffNetV2-L [72]	480^2	120M	53.0G	83.7	85.7
○ DeiT-S [73]	224^2	22M	4.6G	978.5	79.8
○ DeiT-B [73]	224^2	87M	17.6G	302.1	81.8
○ Swin-T	224^2	28M	4.5G	757.9	81.3
• ConvNeXt-T	224^2	29M	4.5G	774.7	**82.1**
○ Swin-S	224^2	50M	8.7G	436.7	83.0
• ConvNeXt-S	224^2	50M	8.7G	447.1	**83.1**
○ Swin-B	224^2	88M	15.4G	286.6	83.5
• ConvNeXt-B	224^2	89M	15.4G	292.1	**83.8**
○ Swin-B	384^2	88M	47.1G	85.1	84.5
• ConvNeXt-B	384^2	89M	45.0G	95.7	**85.1**
• ConvNeXt-L	224^2	198M	34.4G	146.8	**84.3**
• ConvNeXt-L	384^2	198M	101.0G	50.4	**85.5**

- ConvNext achieves a much better accuracy under the same amount of parameters and computation budgets.

Liu, Zhuang, et al. "A convnet for the 2020s." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022.

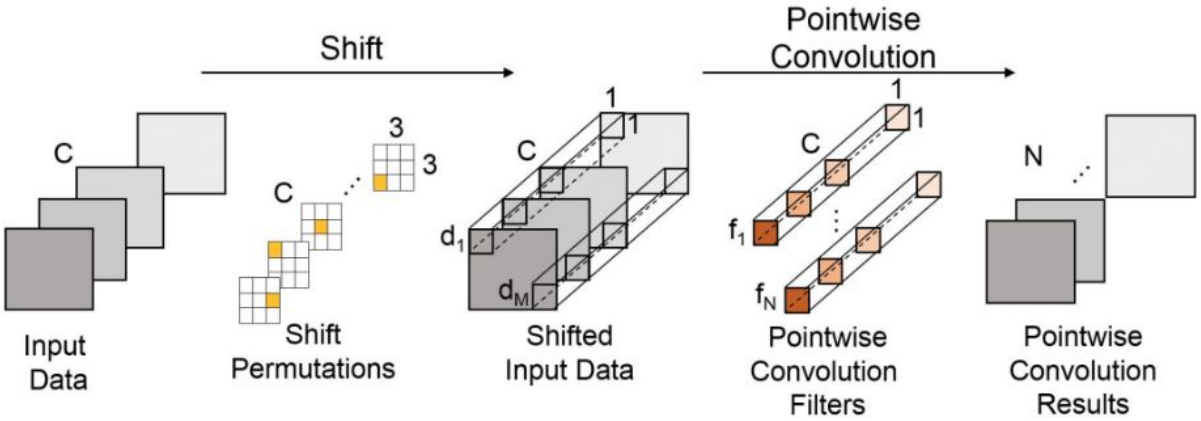
ShiftNet

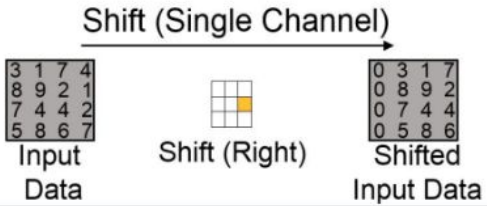
- Number of MACs for depthwise separable Conv: **K×K×C×E×F + M×C×E×F**
- Number of MACs for standard Conv: **M×K×K×C×E×F**
- When M is large the computational saving is about K×K (9) times.

- However, depthwise convolution still needs to read the same amount of input data (B×C×W×H), so on systems limited by memory bandwidth this operation remains slow.

Wu, Bichen, et al. "Shift: A zero flop, zero parameter alternative to spatial convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

ShiftNet

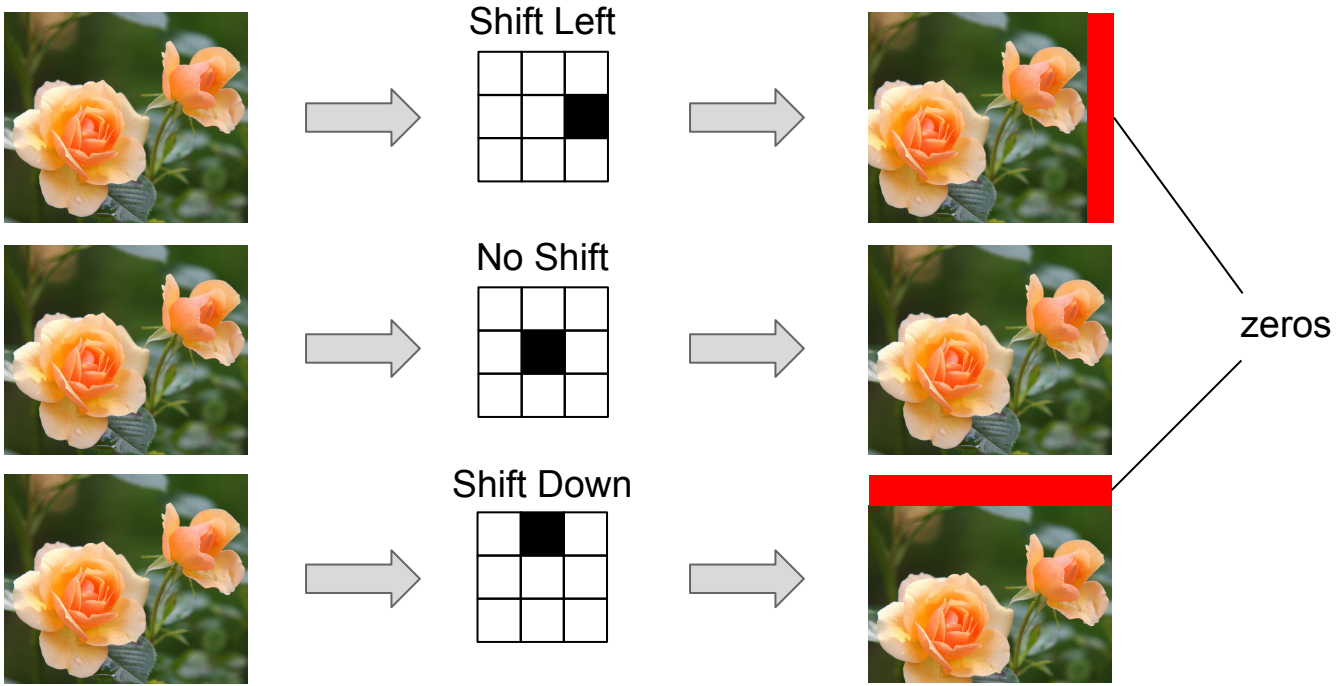




- Completely remove the computation for the depthwise convolution.
- The shift positions are predefined for each channel.

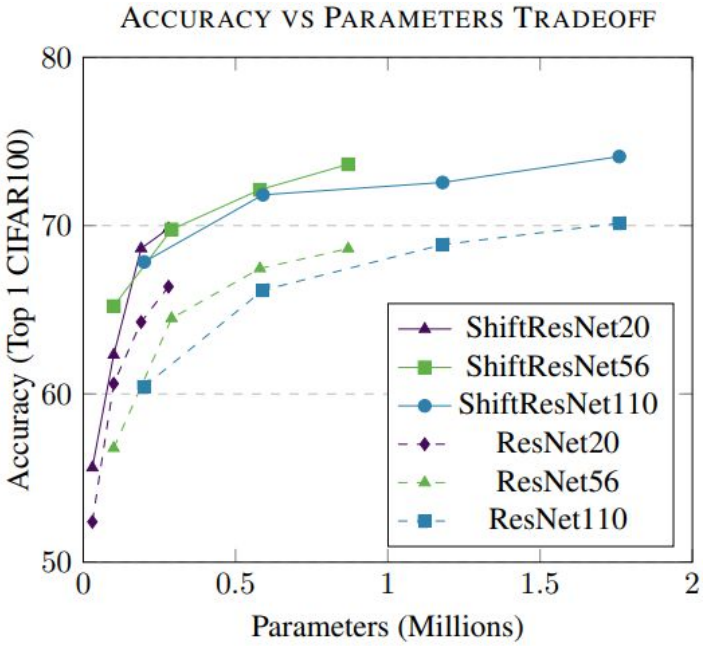
Wu, Bichen, et al. "Shift: A zero flop, zero parameter alternative to spatial convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

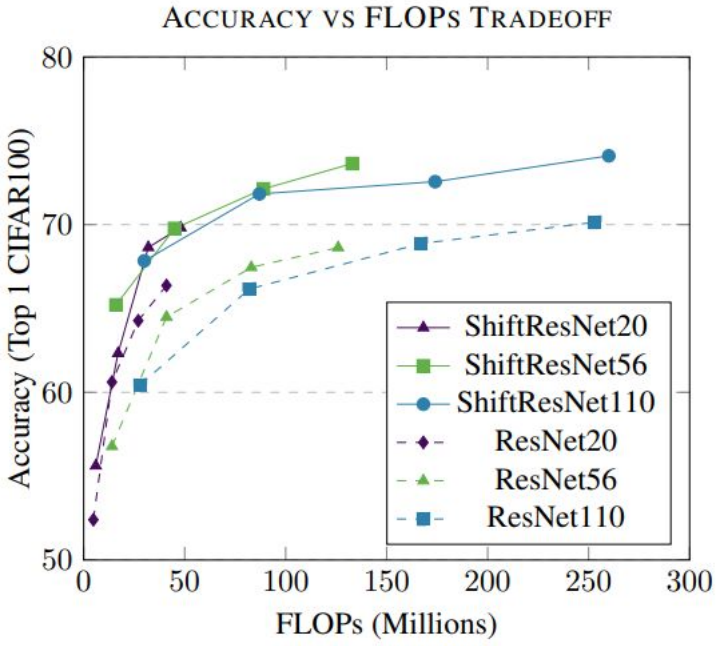
ShiftNet



Wu, Bichen, et al. "Shift: A zero flop, zero parameter alternative to spatial convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

ShiftNet

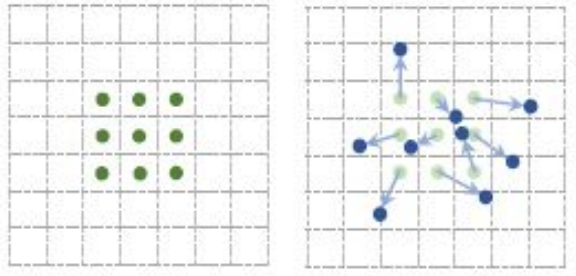




NYU SAI LAB

Deformable Convolutional Networks

- Convolutional neural networks (CNNs) are inherently limited to model geometric transformations due to the fixed geometric structures in their building modules.
- This paper proposed the “learnable weight kernel shape”.



$$\mathcal{R} = \{(-1,-1),(-1,0),\ldots,(0,1),(1,1)\}$$

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n),$$

NYU SAI LAB

Dai, Jifeng, et al. "Deformable convolutional networks." *Proceedings of the IEEE international conference on computer vision*. 2017.

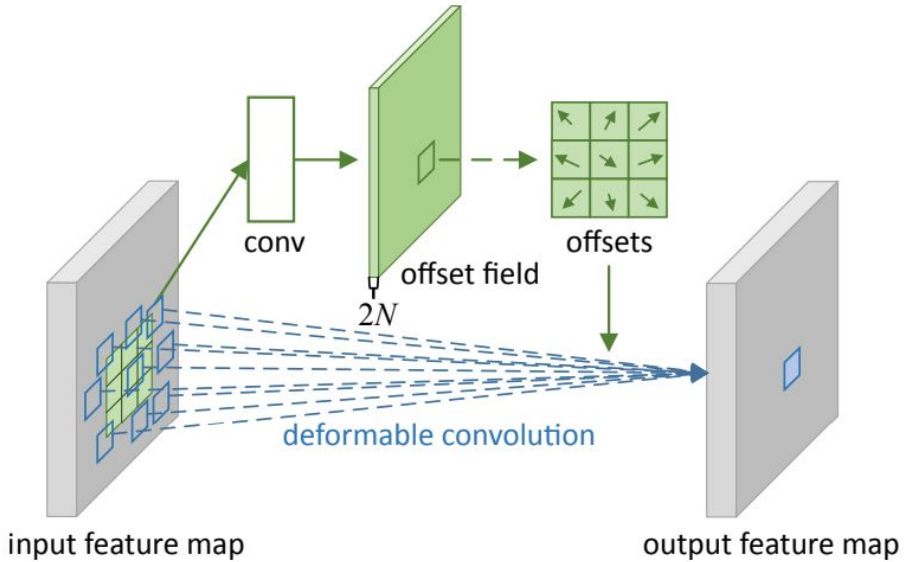
Deformable Convolutional Networks

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n),$$

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)$$

Can be fractional

$$\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q}),$$



Dai, Jifeng, et al. "Deformable convolutional networks." *Proceedings of the IEEE international conference on computer vision*. 2017.

Deformable Convolutional Networks

usage of deformable convolution (# layers)	DeepLab		class-aware RPN		Faster R-CNN		R-FCN	
	mIoU@V (%)	mIoU@C (%)	mAP@0.5 (%)	mAP@0.7 (%)	mAP@0.5 (%)	mAP@0.7 (%)	mAP@0.5 (%)	mAP@0.7 (%)
none (0, baseline)	69.7	70.4	68.0	44.9	78.1	62.1	80.0	61.8
res5c (1)	73.9	73.5	73.5	54.4	78.6	63.8	80.6	63.0
res5b,c (2)	74.8	74.4	74.3	56.3	78.5	63.3	81.0	63.8
res5a,b,c (3, default)	**75.2**	**75.2**	74.5	57.2	78.6	63.3	81.4	64.7
res5 & res4b22,b21,b20 (6)	74.8	75.1	**74.6**	**57.7**	**78.7**	**64.0**	**81.5**	**65.4**

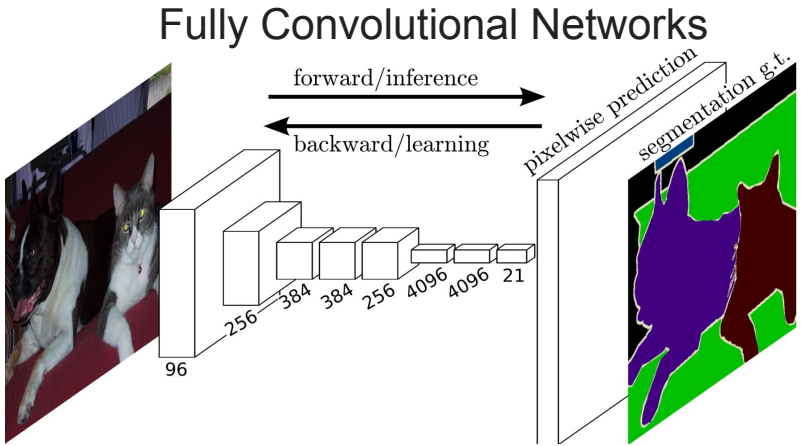
Table 1: Results of using deformable convolution in the last 1, 2, 3, and 6 convolutional layers (of 3×3 filter) in ResNet-101 feature extraction network. For *class-aware RPN*, *Faster R-CNN*, and *R-FCN*, we report result on VOC 2007 test.

NYU SAI LAB

Topics

- Convolutional Neural Network
  - Basic building blocks
  - Popular CNN architectures
    - VGG
    - ResNet
    - MobileNet
    - ShuffleNet
    - SqueezeNet
    - DenseNet
    - EfficientNet
    - ConvNext
    - ShiftNet
  - CNN architectures for other vision tasks
    - Image Segmentation, Object Detection

CNNs for Other Tasks: Image Segmentation



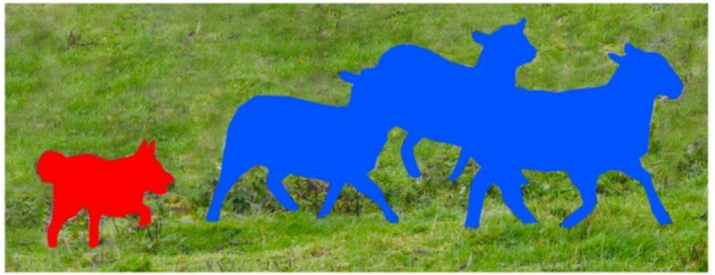
- A fully convolutional based DNN for image segmentation.
- Input: H✖W✖3 → Output: H✖W✖C

- Image segmentation is a computer vision technique used to divide an image into multiple segments or regions, each representing a different object, part of an object, or background.
- The goal of image segmentation is to simplify or change the representation of an image into something more meaningful and easier to analyze.

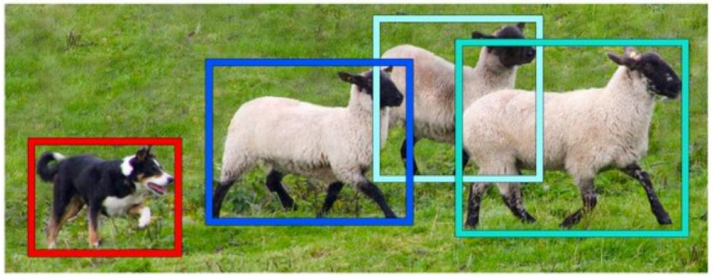
Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

Segmentation

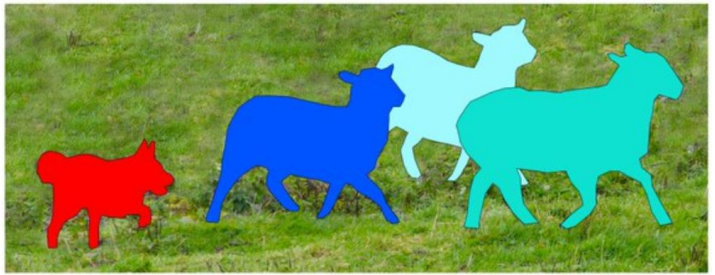
Image Recognition



Semantic Segmentation

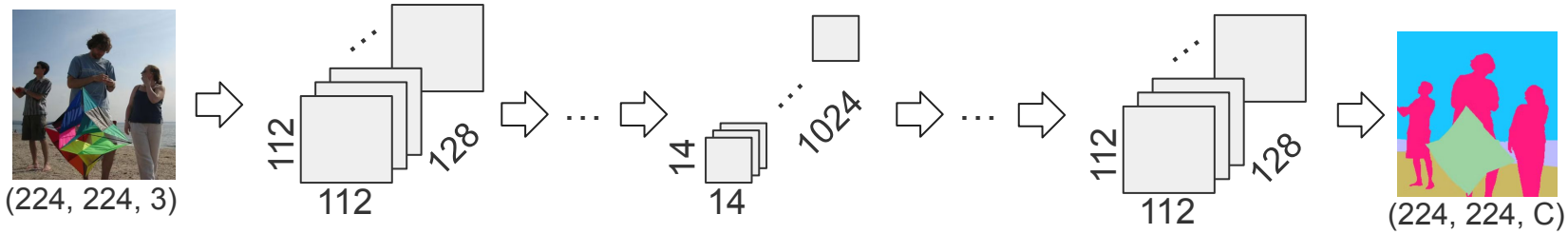


Object Detection



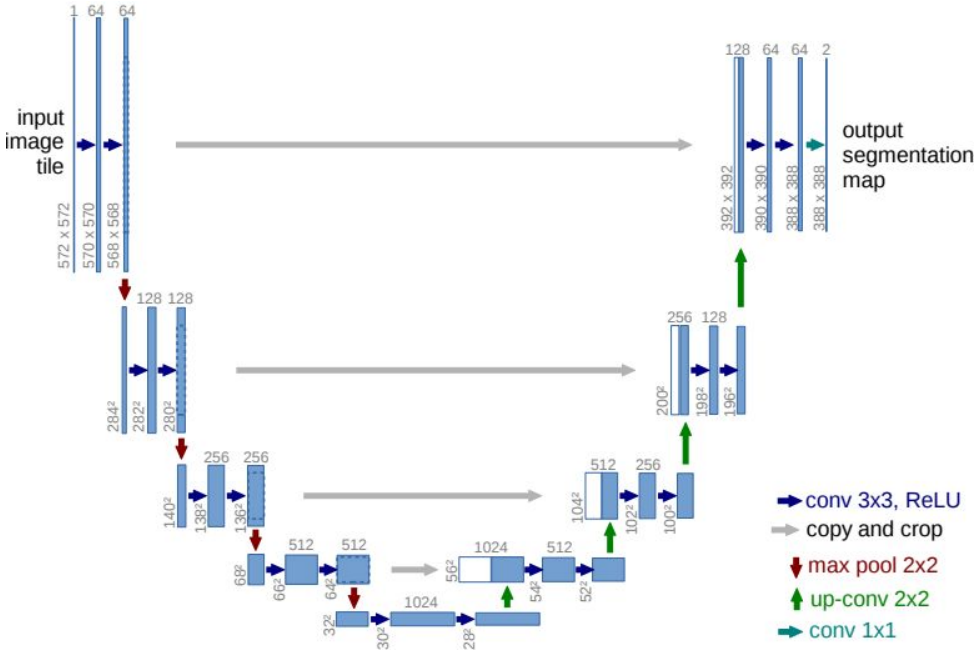
Instance Segmentation

Segmentation



- Segmentation is a pixel-level task in which each pixel is assigned an output label.
- The loss function (cross-entropy loss) is applied on each pixel.

U-Net



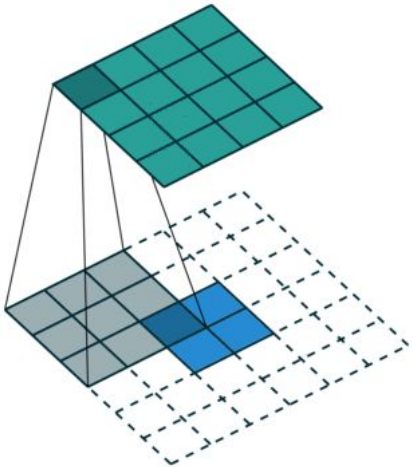
- The direct path sends feature maps from the encoder directly to the corresponding decoder layers, allowing the decoder to recover spatial precision.
- This stabilizes training and improves convergence.

NYU SAI LAB

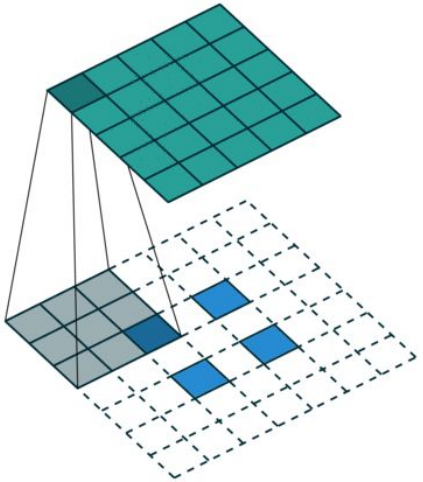
Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *International Conference on Medical image computing and computer-assisted intervention*. Cham: Springer international publishing, 2015.

Transposed Convolution

- To upsample the input, we can apply transposed convolution.



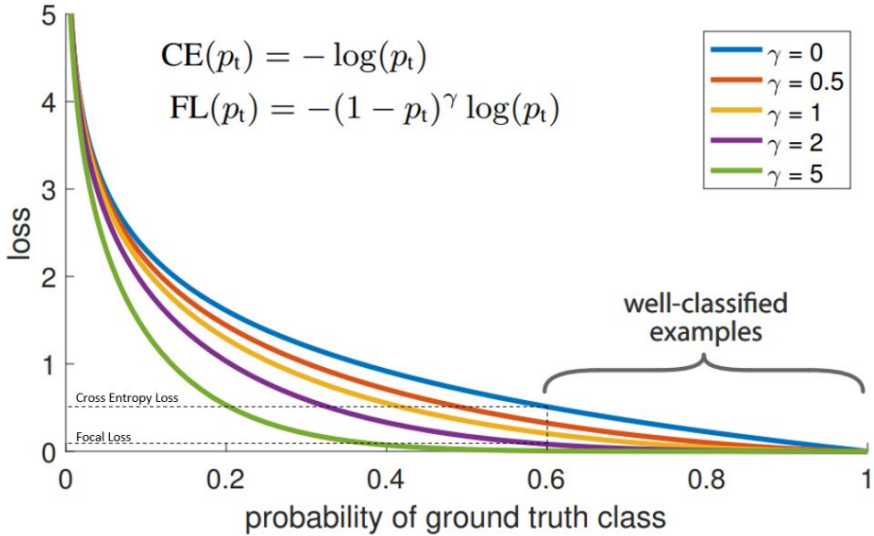
Stride = 1



Stride = 2

https://hannibunny.github.io/mlbook/neuralnetworks/convolutionDemos.html

Focal Loss



- A modified cross-entropy designed to perform better with class imbalance.
- Often used in the problem of object detection and image segmentation.
  - Down-weight easy examples and thus focus training on hard negatives

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t)$$

- γ controls the shape of the curve
- a controls the class imbalance and introduce weights to each class.

NYU SAI LAB

Lin, Tsung-Yi, et al. "Focal loss for dense object detection." *Proceedings of the IEEE international conference on computer vision*. 2017.

Dice Loss

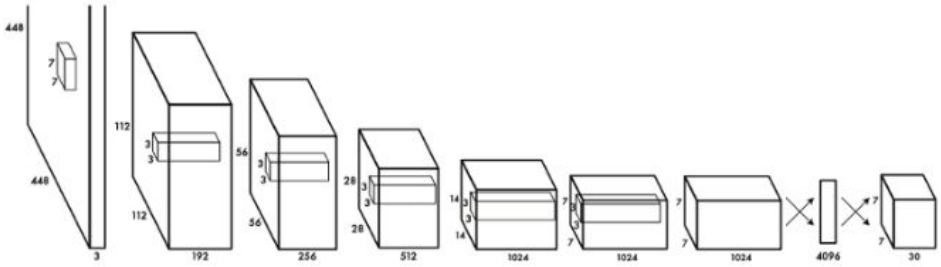
- The Dice loss is widely used in segmentation tasks (especially in medical imaging) where class imbalance is common.

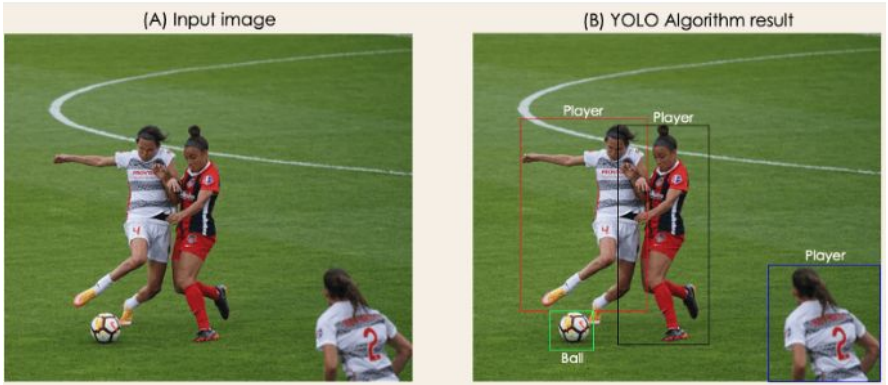
$$\text{Dice Loss} = 1 - \frac{2\sum_i p_i g_i}{\sum_i p_i + \sum_i g_i + \epsilon}$$

- pi: predicted probability for pixel i.
- gi: ground truth label (0 or 1) for pixel i.
- ϵ: small constant to avoid division by zero.

Sudre, Carole H., et al. "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations." *International Workshop on Deep Learning in Medical Image Analysis*. Cham: Springer International Publishing, 2017.

CNNs for Other Tasks: Object Detection

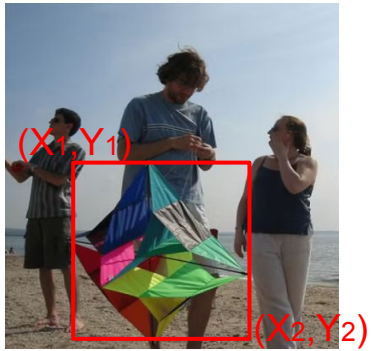


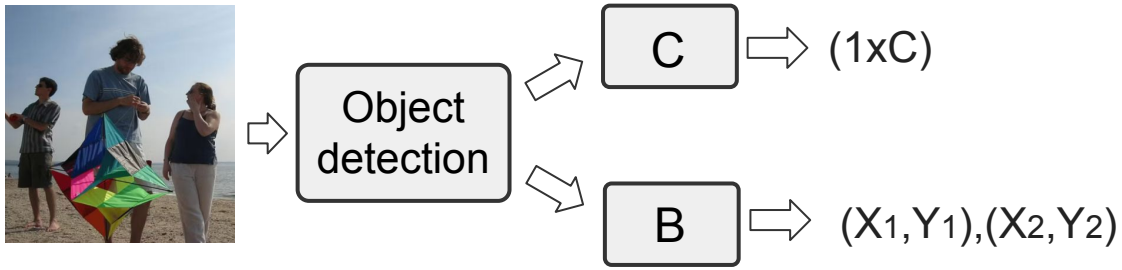


- NN will generate the likelihood of each anchor point and the coordinates of its bounding box.
- Another branch will produce the category of each bounding box

NYU SAI LAB

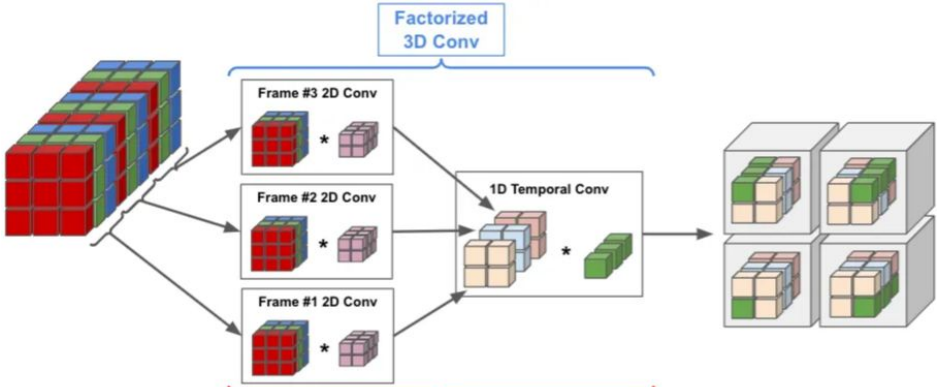
CNNs for Other Tasks: Object Detection





- The bounding box is defined by its top-left and bottom-right coordinates, and the object detection network also outputs a 1×C classification vector.

CNNs for Other Tasks: Video Processing



- To process video, we can concatenate the consecutive frames together and use 2D convolution to process it.