



# Lecture 02: CNN, RNN and Variants

# Notes

- Great job! All of you have made the selection.
- We are going to release the reading list assignment by tonight!
- You can choose to send me the presentation, or we can present it using your laptop.
- Please keep the time ~15-20 mins for AI paper, and ~30-35mins for AI hardware paper.

# Recap

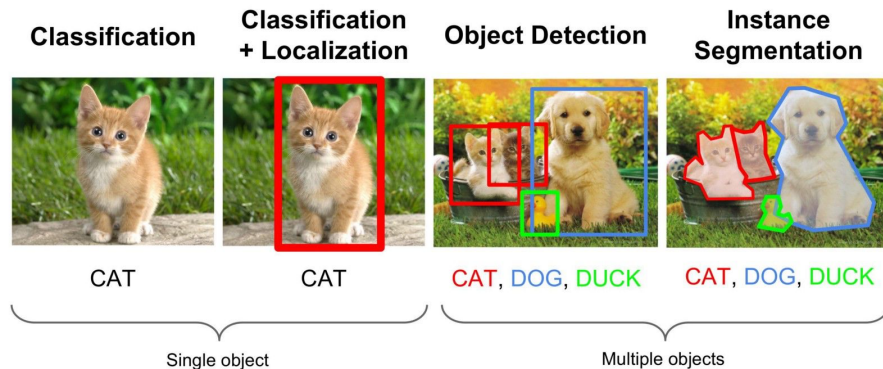
- DNN basics
  - Multilayer perceptron
    - Linear layer, activation function, softmax layer
  - Loss functions
  - Weights decay
  - Dropout
  - Optimizer
  - Learning rate scheduler
  - Weight Initialization

# Topics

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

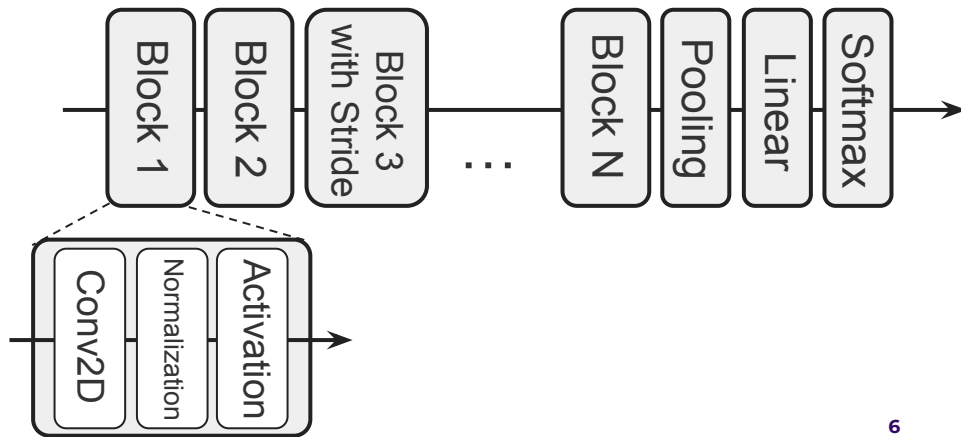
# Convolutional Neural Networks

- 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

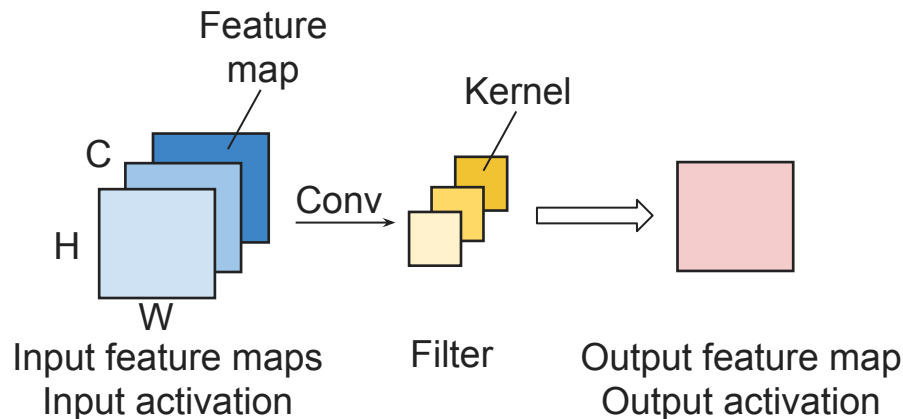


# Convolutional Neural Networks

- 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

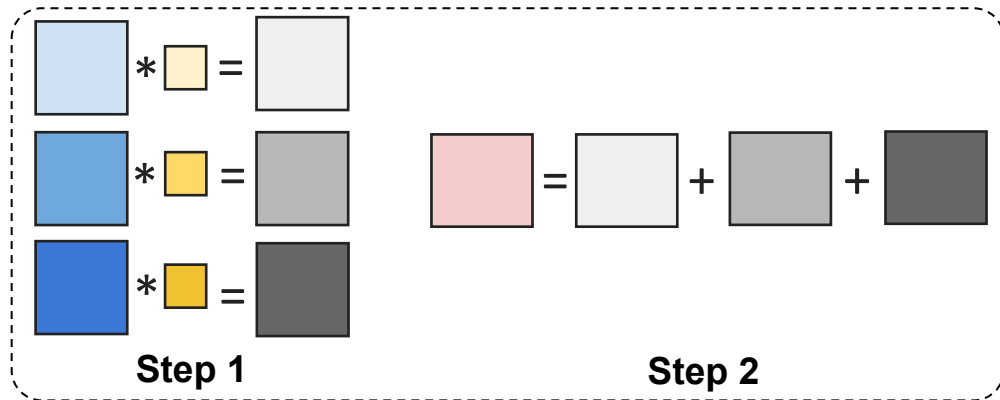
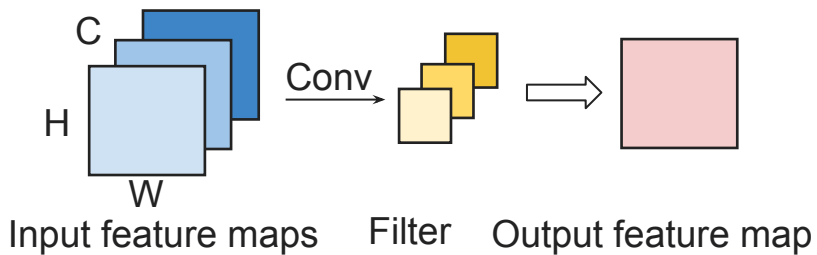


# Convolutional Layers: Terminology



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

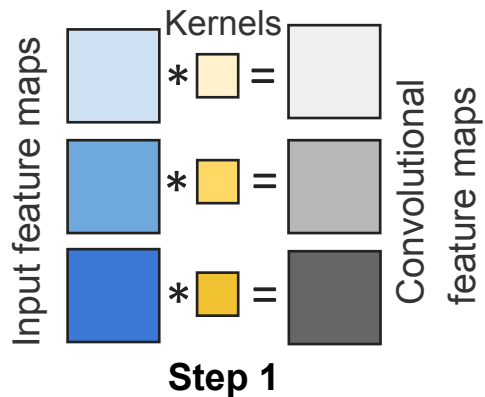
# Convolutional Layers



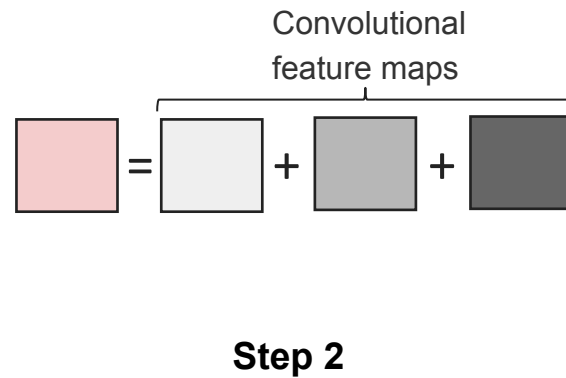
- Core building block of a CNN, it is also the most computational intensive layer.



# Convolutional Layers

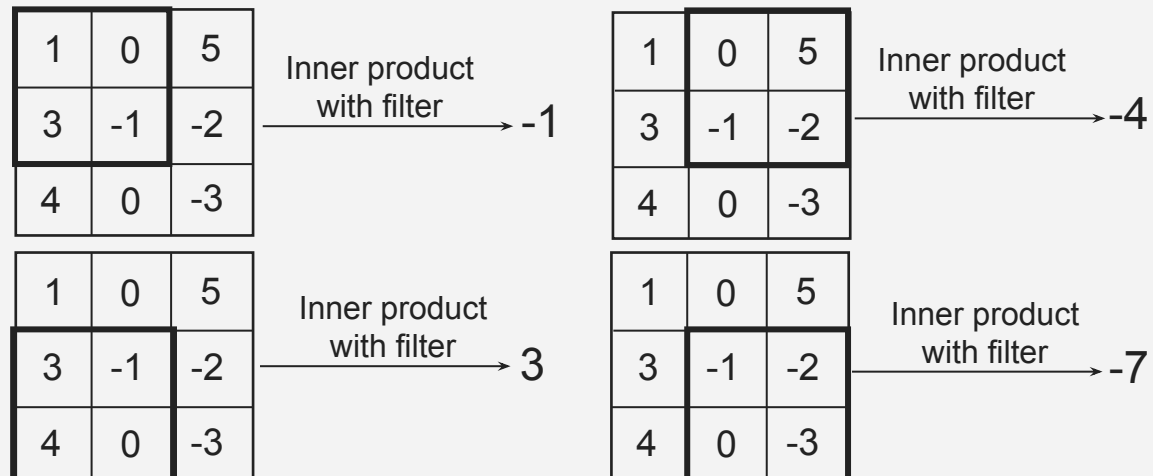
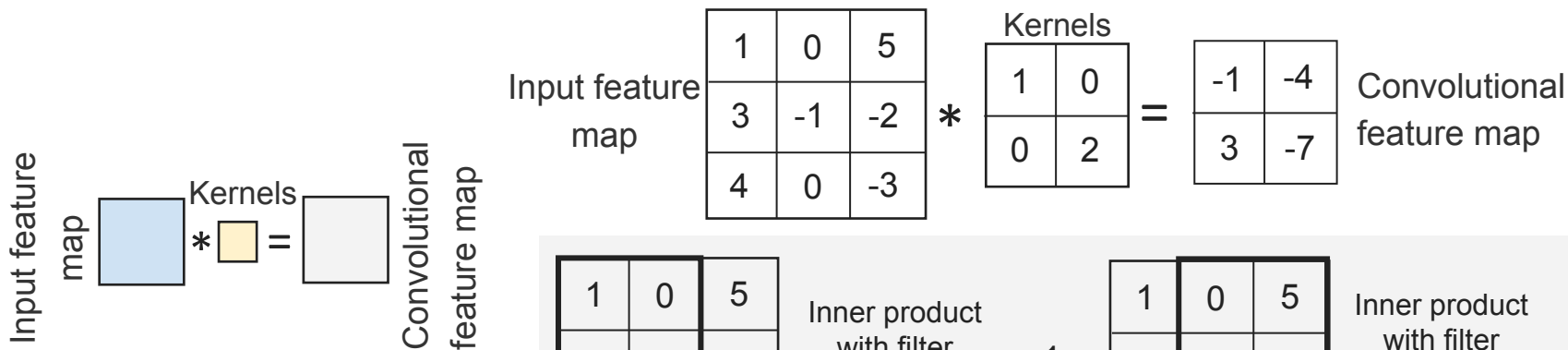


- Each kernel moves across the spatial dimensions of feature maps in the input activations, analyzing the information within those spatial dimensions.



- 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



# Padding

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 preserve 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

\*

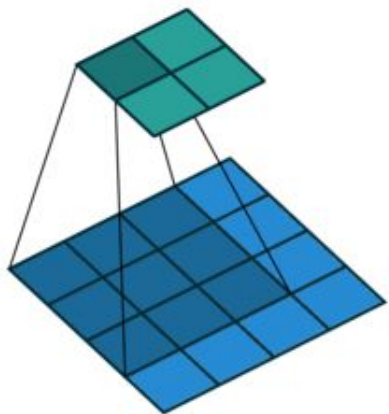
Filter

1	0	3
0	2	-2
1	3	-1

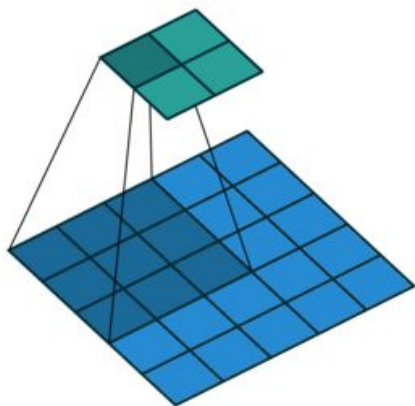
=

12	20	3
20	25	-10
5	3	-7

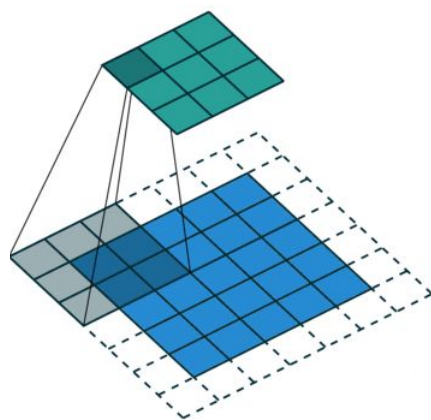
# Stride



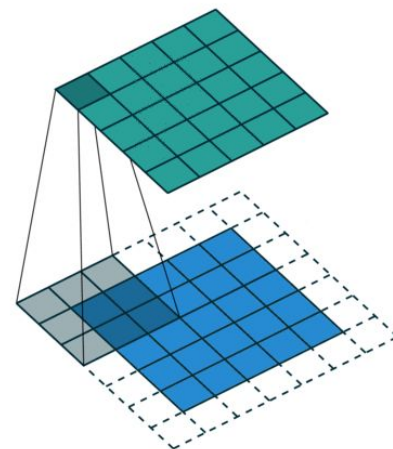
padding = 0, stride = 1



padding = 0, stride = 2



padding = 1, stride = 2



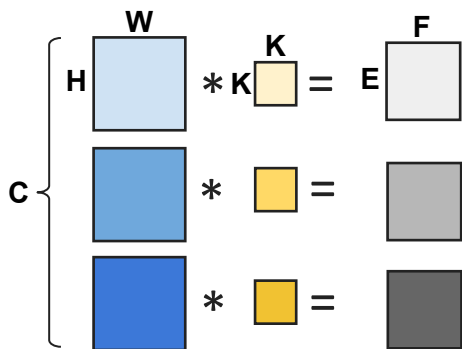
padding = 1, stride = 1

# 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  $224 \times 224 \times 3$ , weight kernel size is  $3 \times 3$ , padding size is 1 and stride size is 1, then the output size is  $(224 - 3 + 2) / 1 + 1 = 224$ .

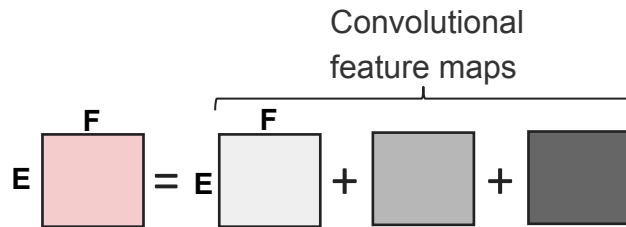
$$H_{in} \square * K \square = H_{out} \square$$

# 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.

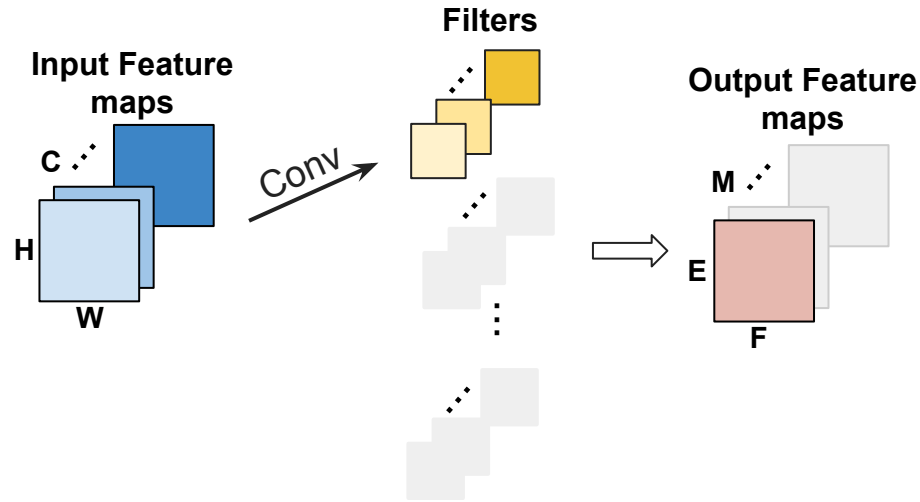


**Step 2**

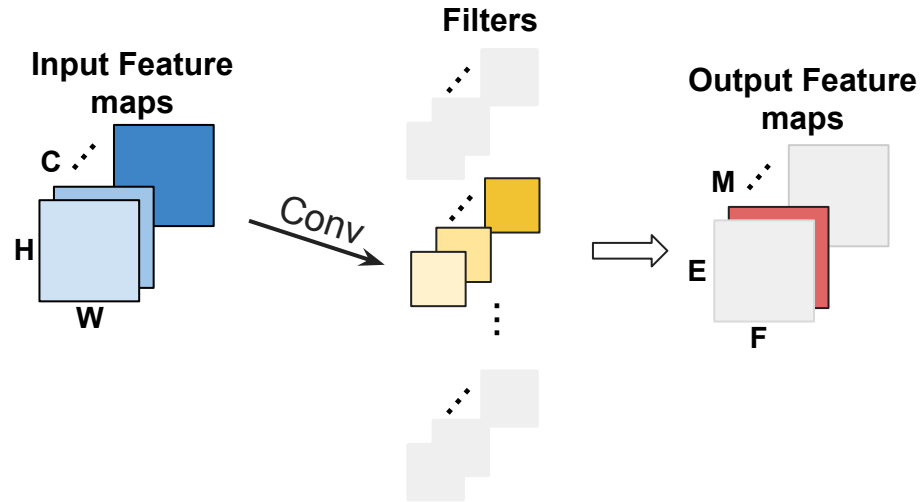
- The information from each feature maps are then aggregated by summing the Convolutional feature maps together.
- A bias may be introduced.

Computational cost in Multiply-accumulate operations (MAC):  $E \times F \times K \times K \times C$

# Convolution

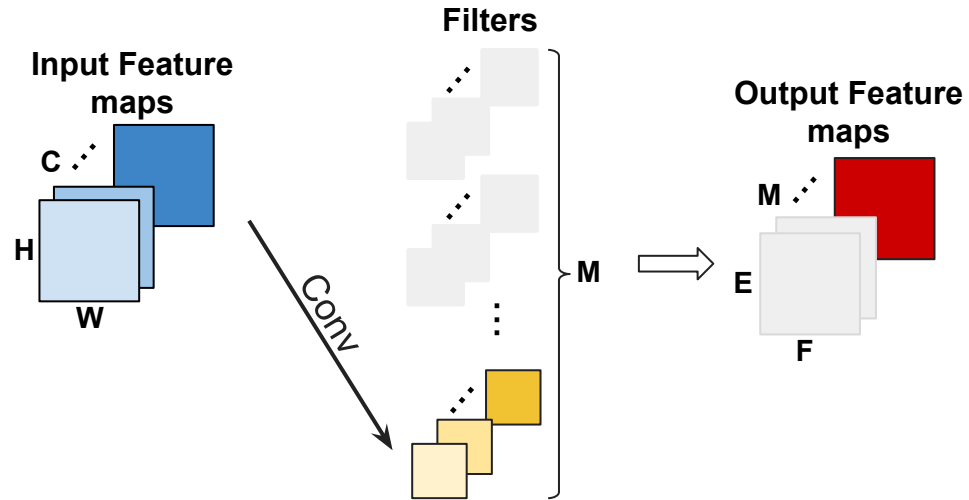


# Convolution

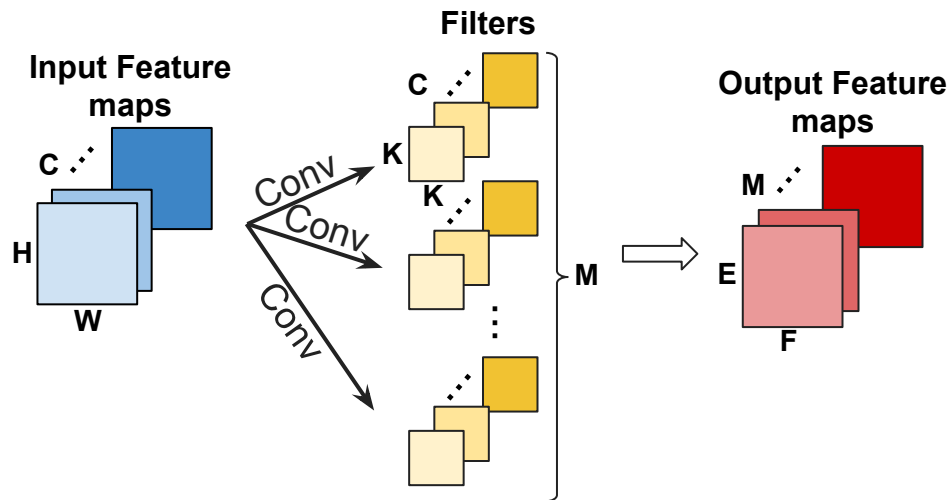




# Convolution



# Convolution



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

$C$ : number of input channels

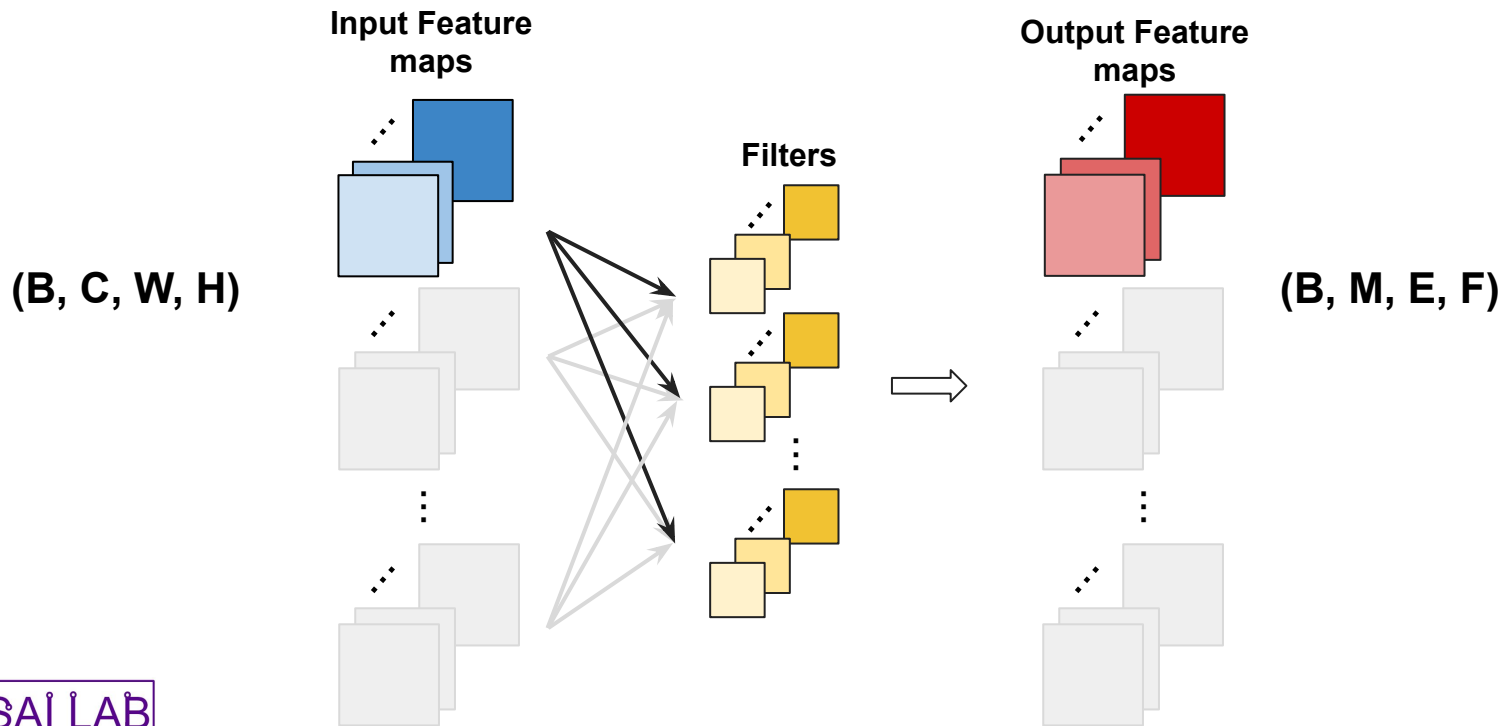
$H, W$ : size of the input feature maps

$M$ : number of weight filters

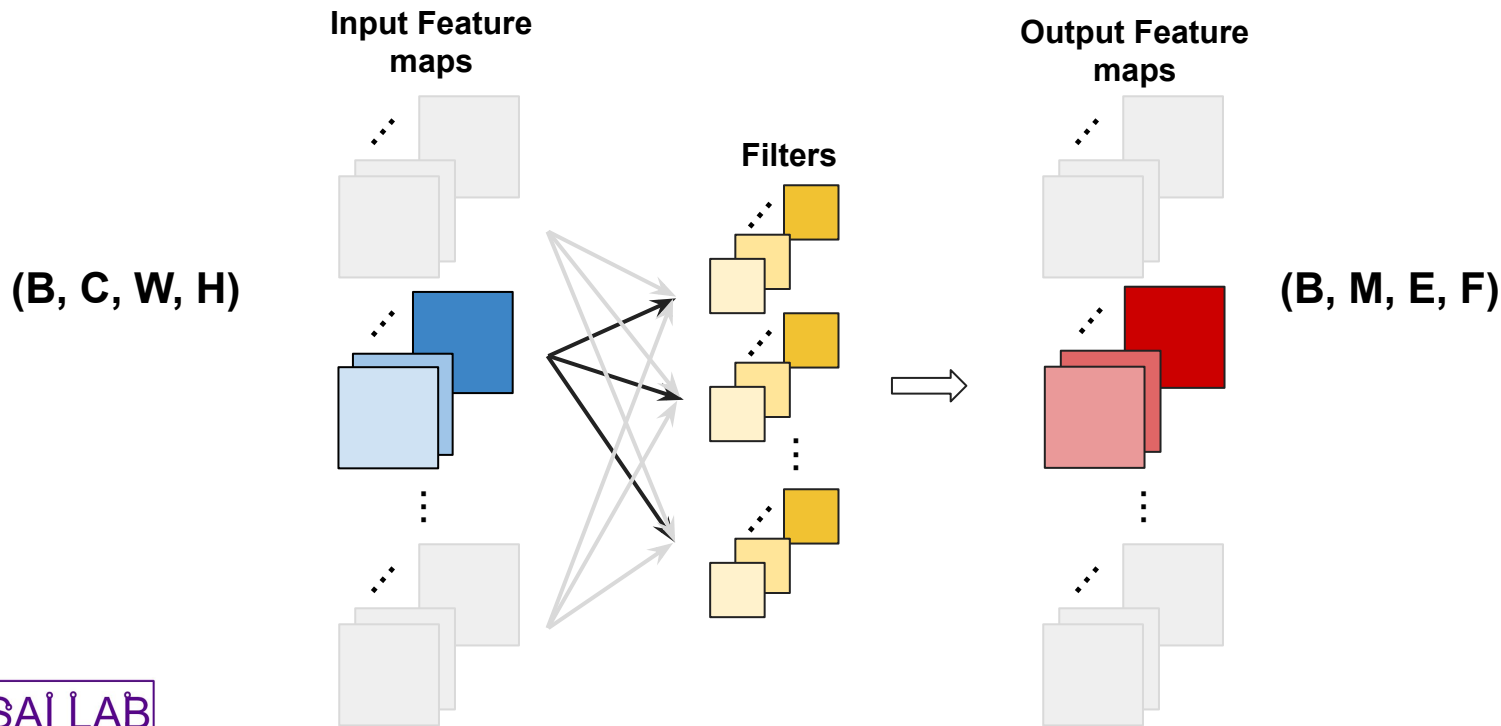
$K$ : weight kernel size

$E, F$ : size of the output feature maps

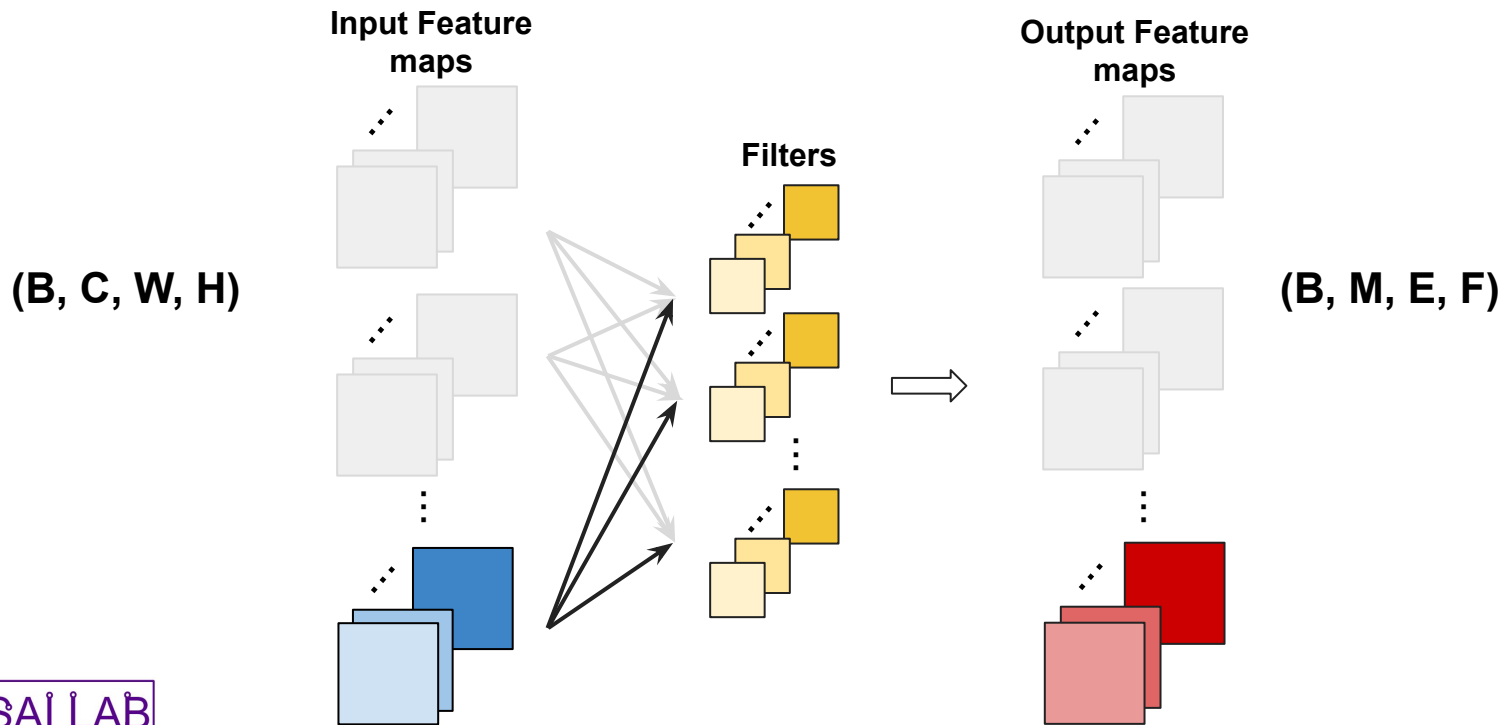
# Convolution



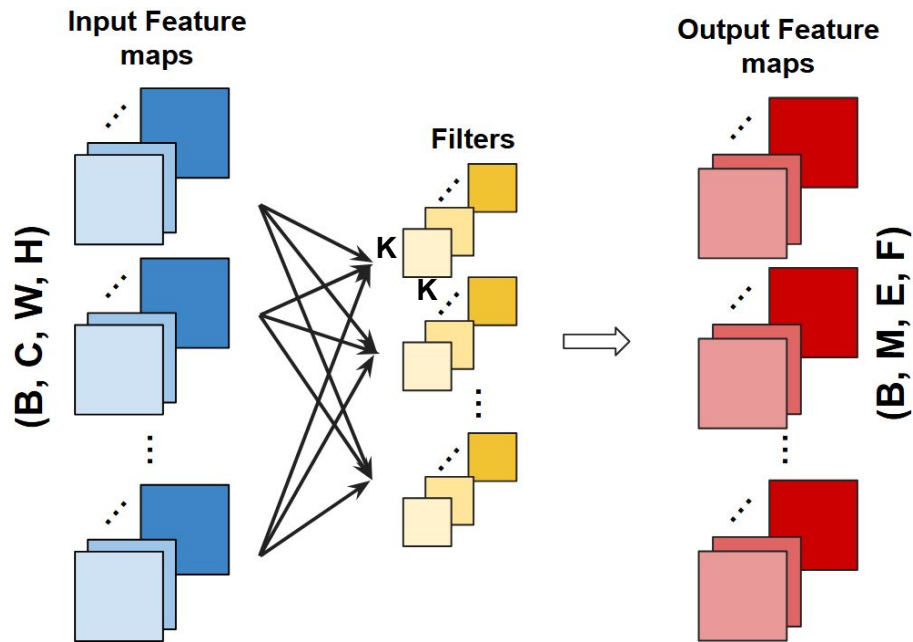
# Convolution



# Convolution



# Computational Cost: Standard Convolution



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

B: batch size

C: number of input channels

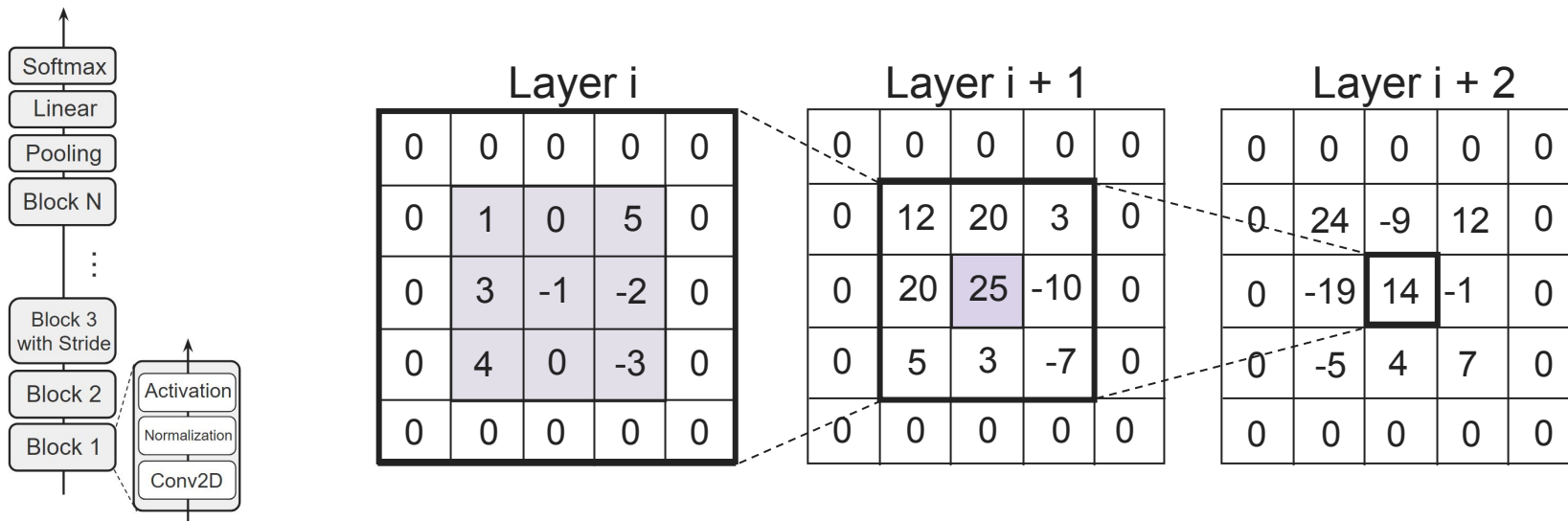
H,W: size of the input feature maps

M: number of weight filters

K: weight kernel size

E,F: size of the output feature maps

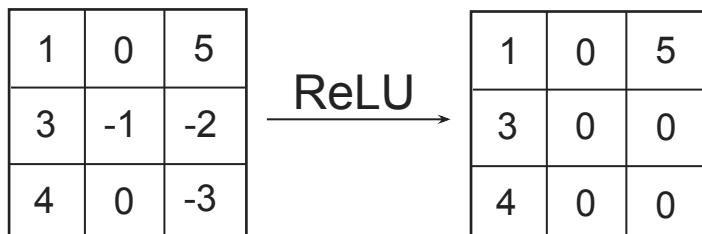
# Receptive Field of CNN across Layers



- 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.

# 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.





# Activation Functions: GeLU

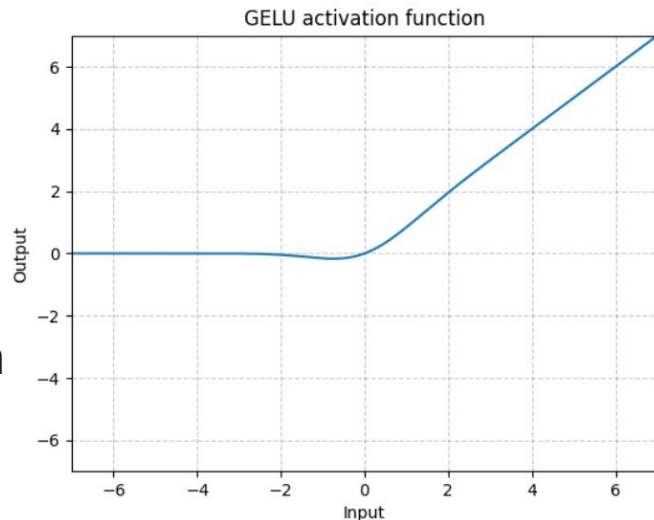
- Gaussian error linear unit (GeLU):

$$\text{GeLU}(x) = x\Phi(x)$$

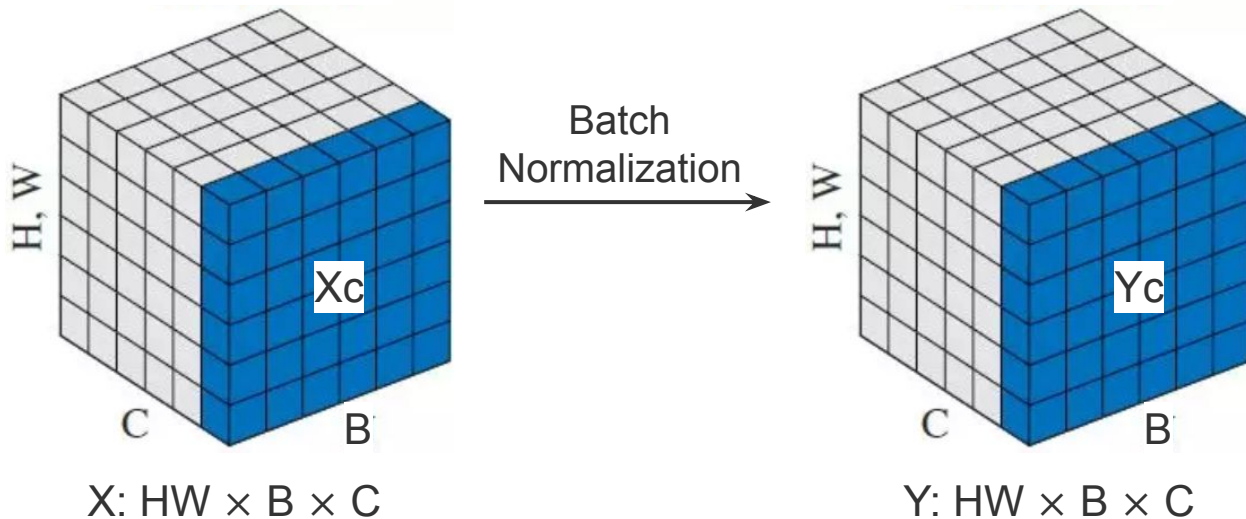
$$\Phi(x) = P(y \leq x), \text{ where } Y \sim N(0, 1)$$

$$0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$$

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

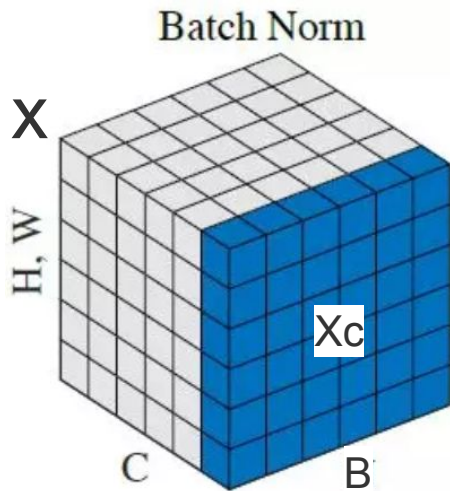


# Batch Normalization



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

# Batch Normalization



X: HW × B × C

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

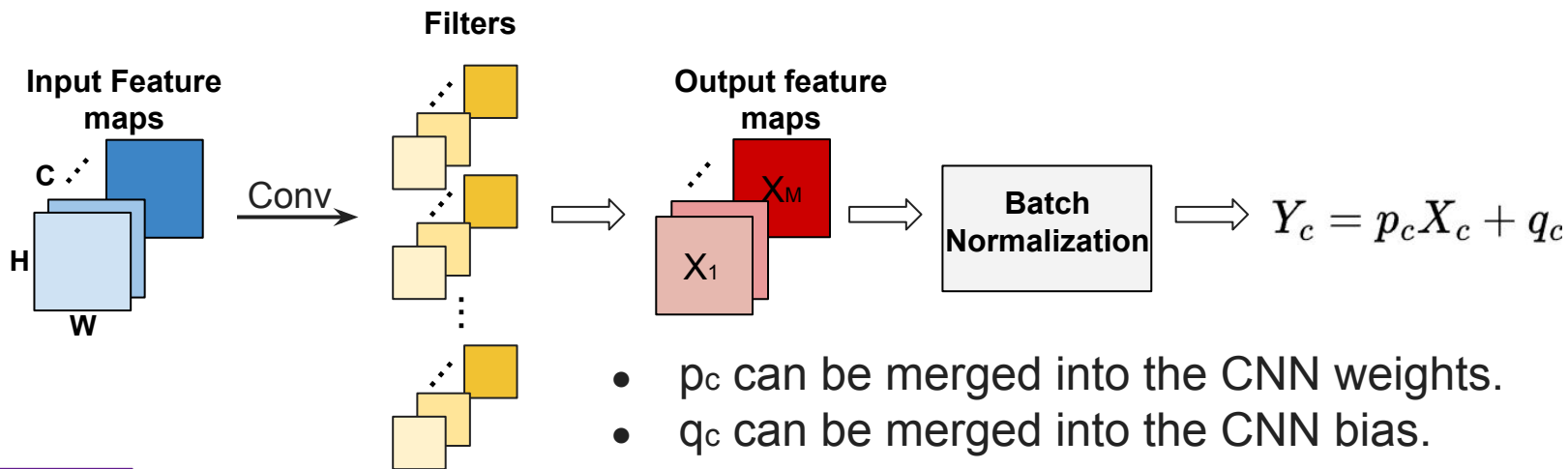
$$\alpha = \{\alpha_c\}, \beta = \{\beta_c\}, \mu = \{\mu_c\}, \sigma = \{\sigma_c\}$$

- For each channel  $c$ , we have:
  - $X_c$ : (HW × B)
  - $\mu_c$  and  $\delta_c$  are the mean and standard deviation of  $X_c$ .
  - $\alpha_c$  and  $\beta_c$  are learnable parameters
  - $\alpha_c, \beta_c, \mu_c, \delta_c$  are scalars
- Overall, we have:
  - $\mu, \delta, \alpha$  and  $\beta$  all have a length of  $C$
  - $\mu, \delta, \alpha$  and  $\beta$  are all fixed during the inference
  - $\mu, \delta$  are statistics based on the training dataset

# Batch Normalization: During Inference

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

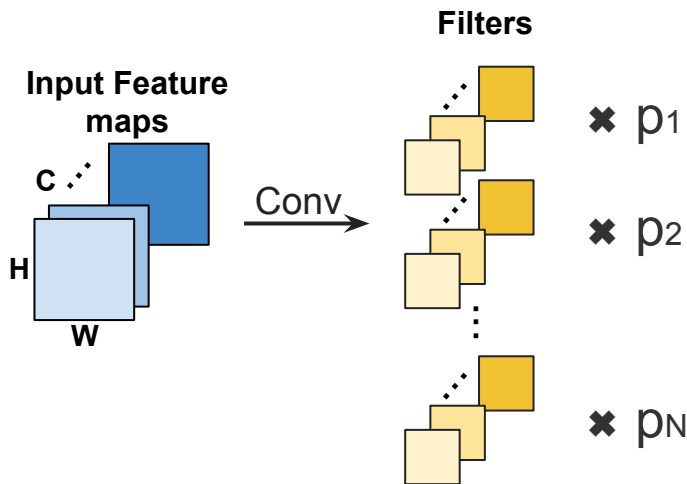
$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c = \frac{\alpha_c}{\sigma_c} X_c + \left( \beta_c - \frac{\alpha_c \mu_c}{\sigma_c} \right) \implies Y_c = p_c X_c + q_c$$



# Batch Normalization

- For each channel  $c$ , we have:

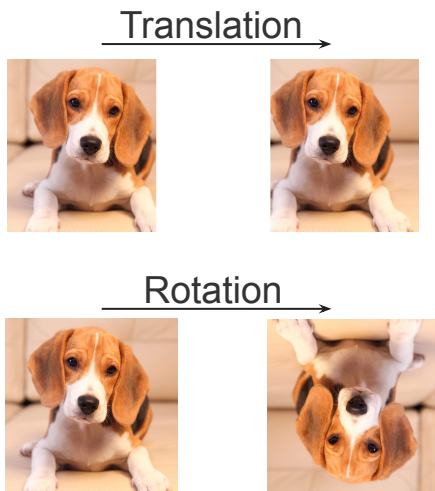
$$Y_c = \alpha_c \frac{X_c - \mu_c}{\sigma_c} + \beta_c = \frac{\alpha_c}{\sigma_c} X_c + \left( \beta_c - \frac{\alpha_c \mu_c}{\sigma_c} \right) \implies Y_c = p_c X_c + q_c$$



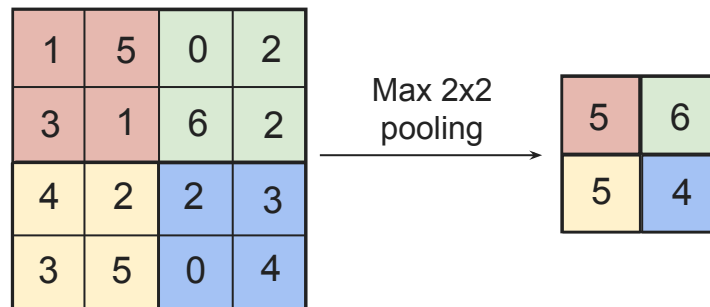
- We can fold in the  $p$  and  $q$  to the weights and bias of convolutional layer during inference and reduce the online computational cost.

# 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.

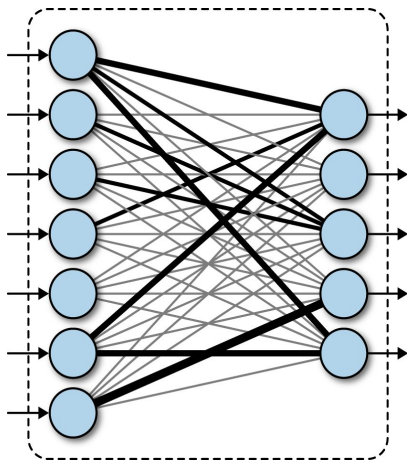


Should produce the same prediction result



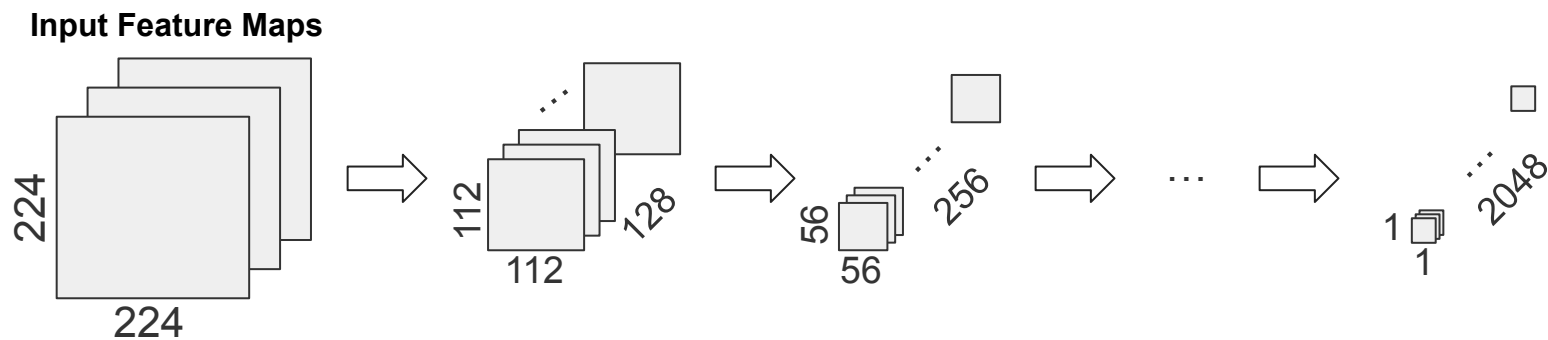
# Fully Connected Layers

- 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



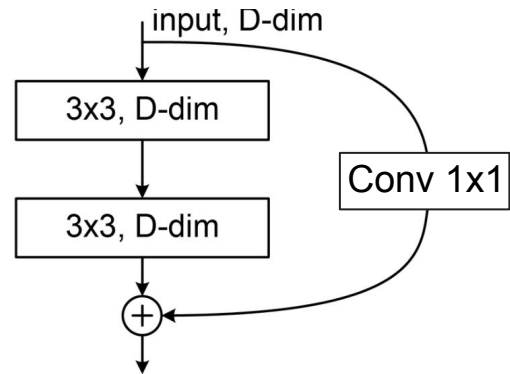
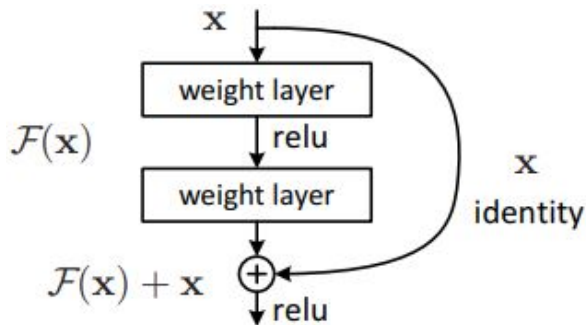
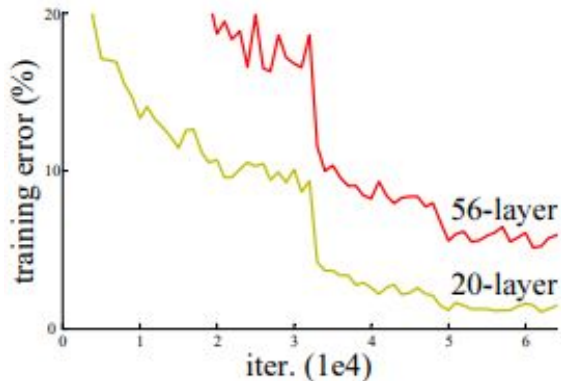
- For image classification task, during the forward propagation of CNN, the spatial size reduces while the number of channels increases.



# Topics

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

# 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.

# ResNet

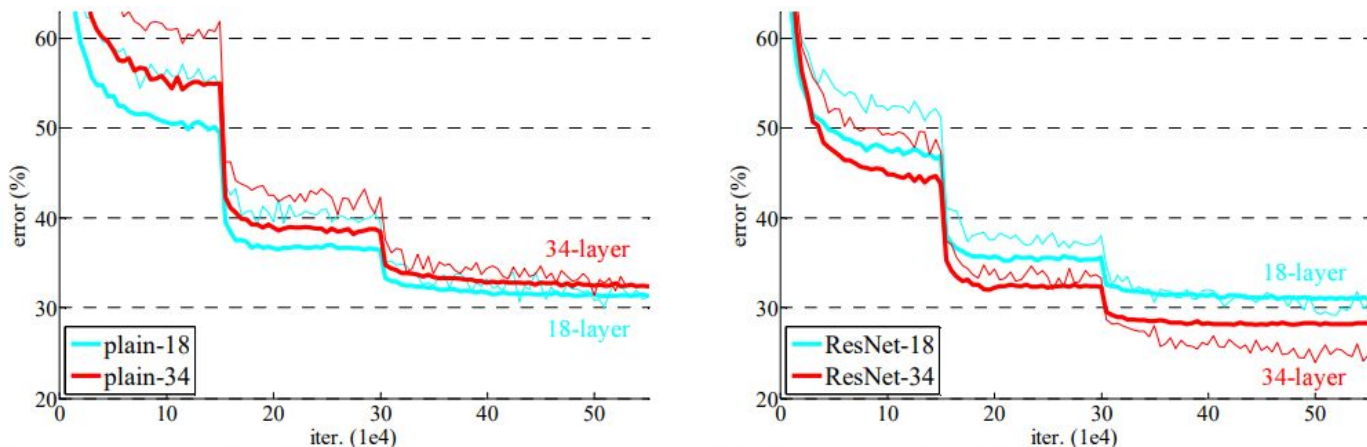


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

# ResNet Performance

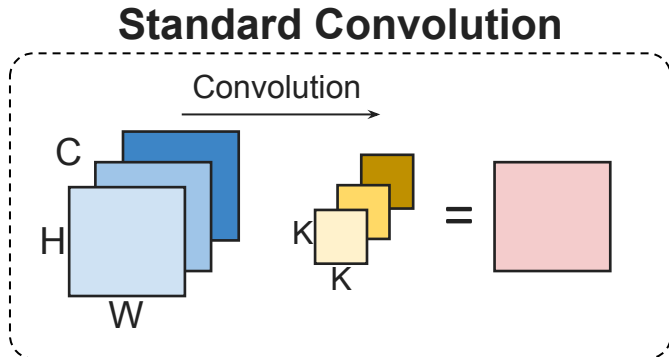
## Performance on ImageNet

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	<b>25.03</b>

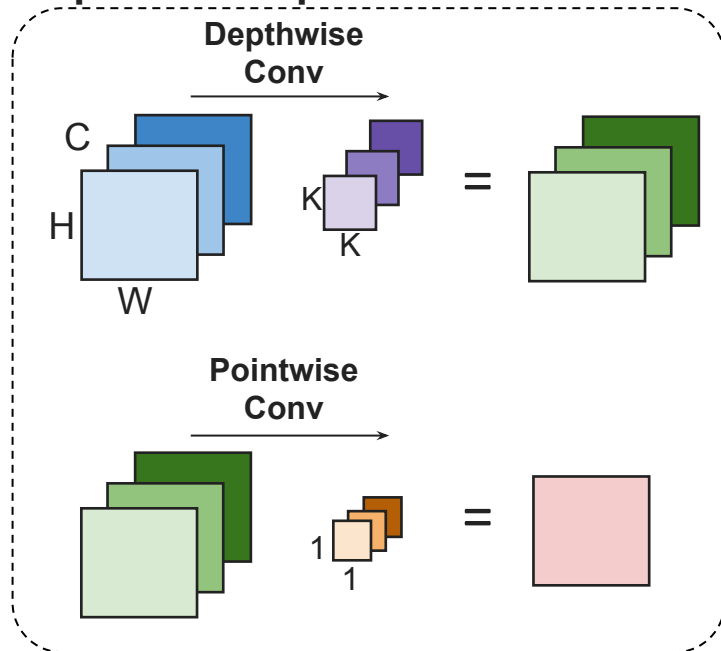
## Performance on CIFAR-10

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	<b>6.43</b> (6.61±0.16)
ResNet	1202	19.4M	7.93

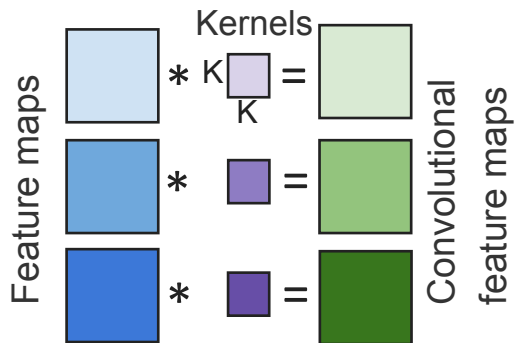
# MobileNet



## Depthwise Separable Convolution

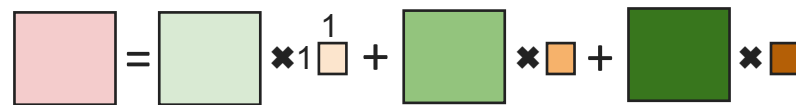


# MobileNet



## Step 1 Depthwise Convolution

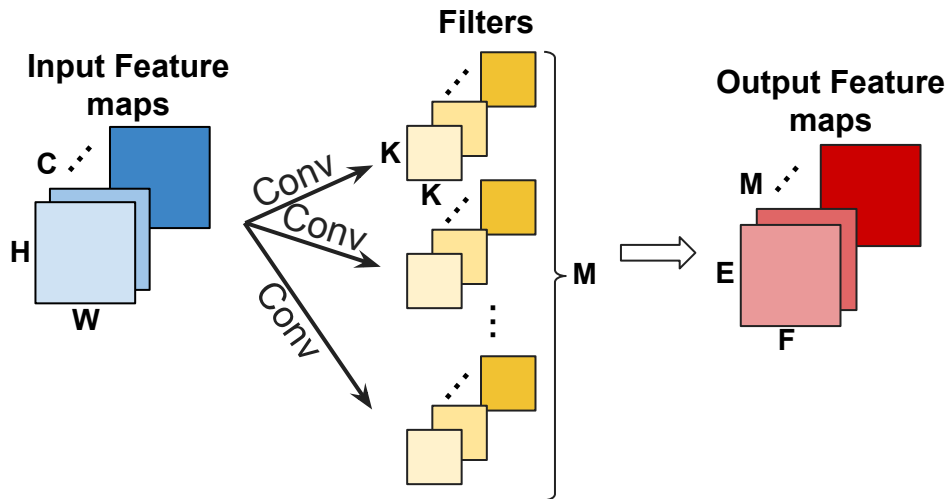
- Each kernel moves across the spatial dimensions of feature maps in the input activations, analyzing the information within those spatial dimensions.



## 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.

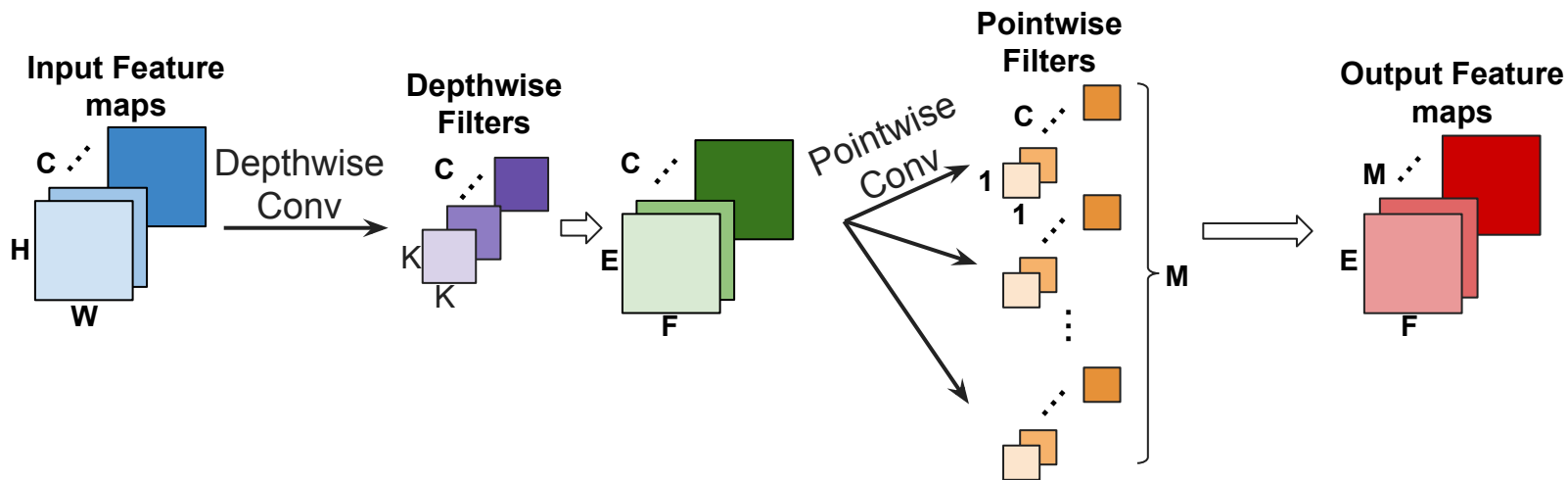
# Standard Convolution



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

$C$ : 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



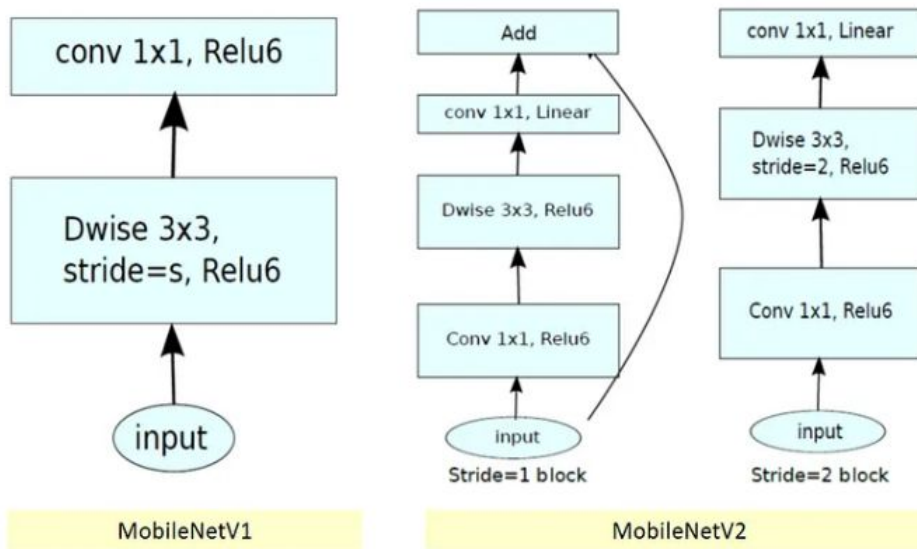
- Number of MACs:  $K \times K \times C \times E \times F + M \times C \times E \times F$
- Storage cost:  $32 \times (C \times H \times W + C \times K \times K + C \times E \times F + M \times C + M \times E \times F)$



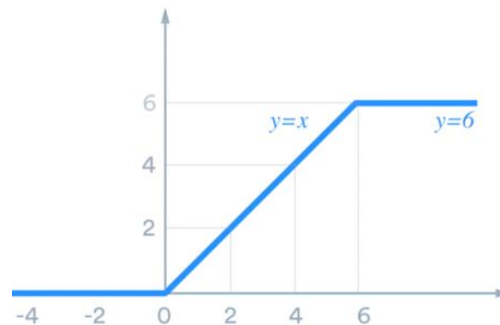
# Why Depthwise Conv is Cheaper?

- Number of MACs for depthwise separable Conv:  $K \times K \times C \times E \times F + M \times C \times E \times F$
- Number of MACs for standard Conv:  $M \times K \times K \times C \times E \times F$
- When M is large the computational saving is about  $K \times K$  (9) times.
  
- With a batch size of B, number of MACs are:
- Number of MACs:  $B \times K \times K \times C \times E \times F + B \times M \times C \times E \times F$
- Storage cost:  $32 \times (B \times C \times H \times W + C \times K \times K + B \times C \times E \times F + M \times C + B \times M \times E \times F)$

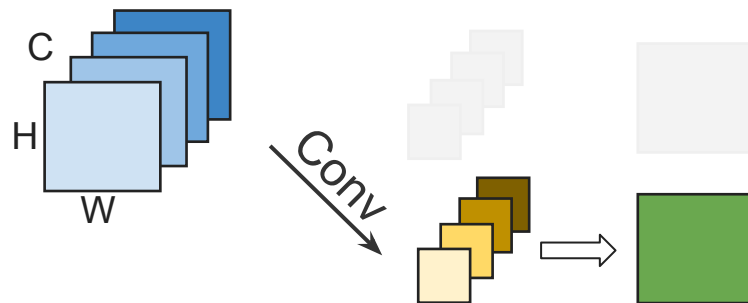
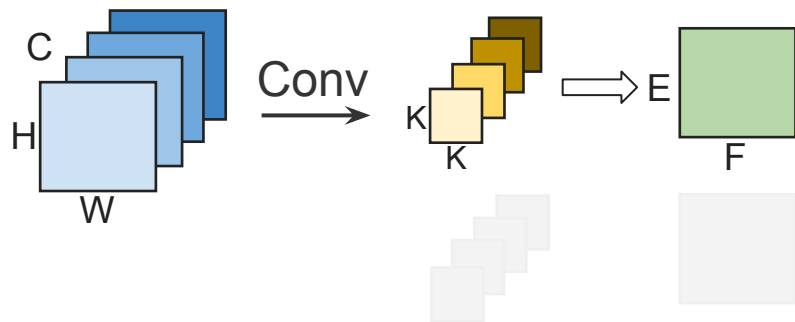
# MobileNet-V2



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

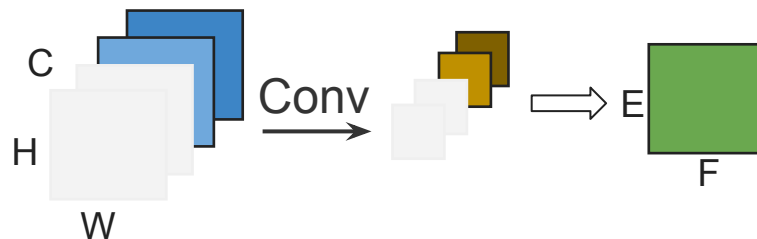
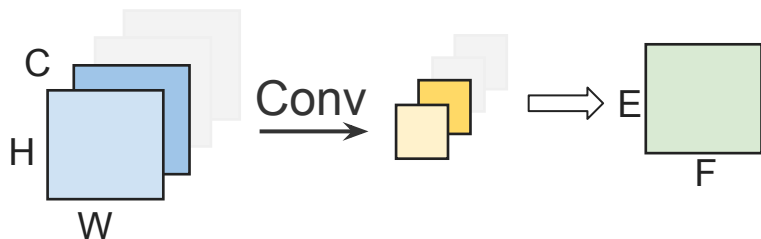


# Group Convolution



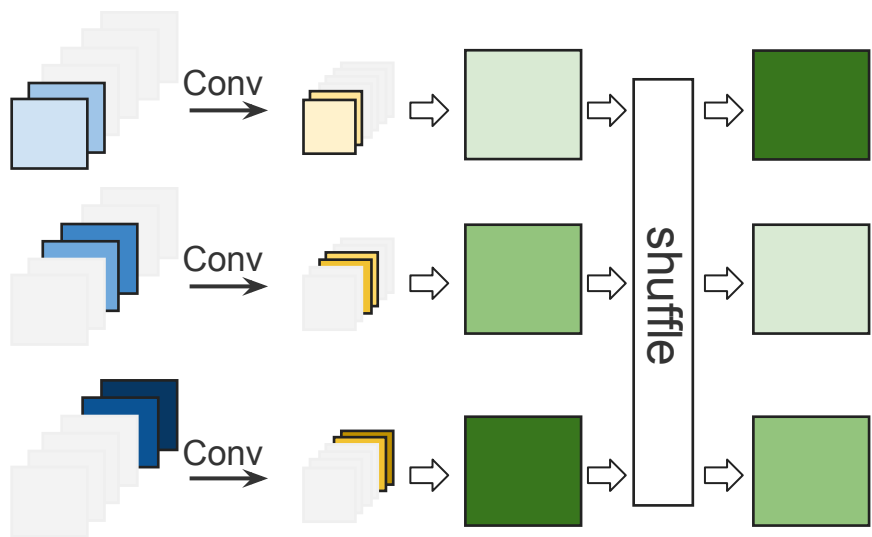
- The original MAC:  $E \times F \times K \times K \times C \times M$

# Group Convolution



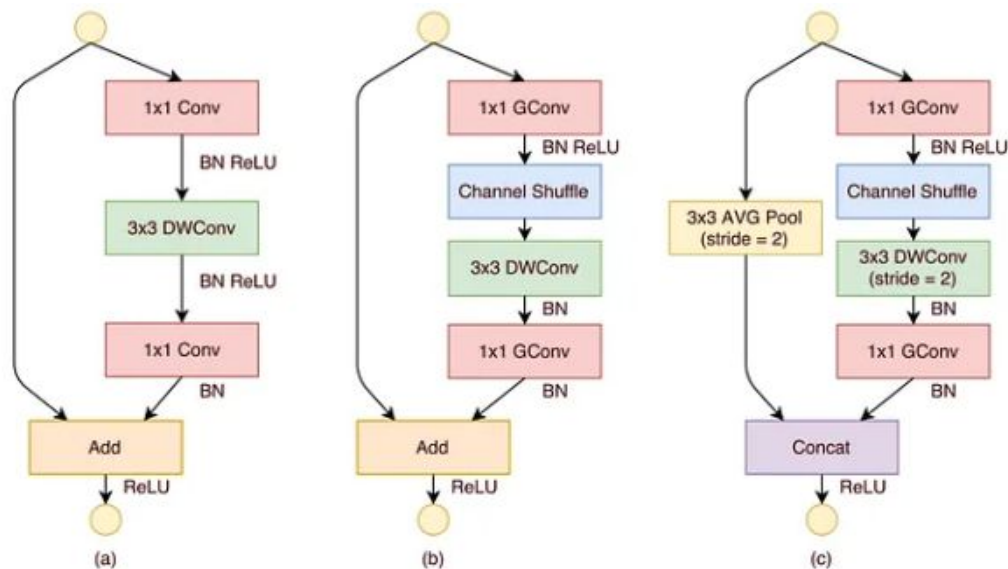
- Group size = 2
- Each group of feature maps within the input only convolved with partial weight kernels.
- This will lead to a large saving on memory consumption and computational cost.
- The number of MAC:  $E \times F \times K \times K \times C \times M/G$

# 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.

# ShuffleNet



# ShuffleNet

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

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

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

# ShuffleNet

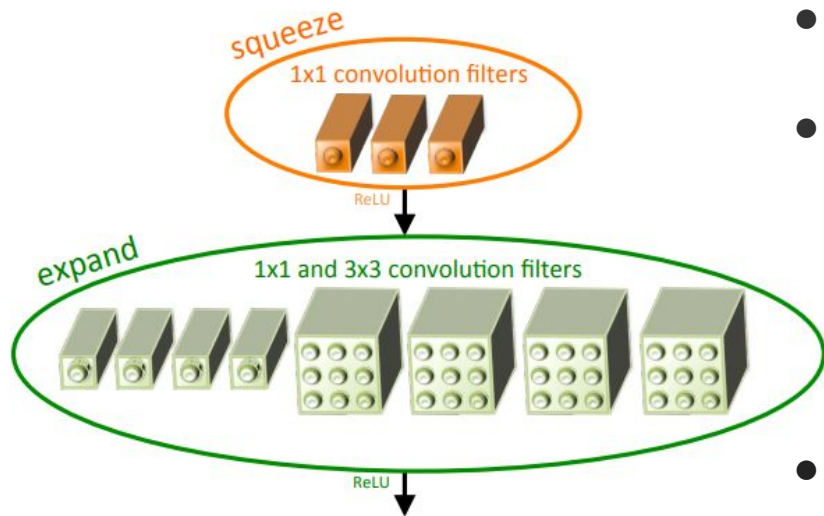
Model	Complexity (MFLOPs)	Cls err. (%)	$\Delta$ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet $2\times (g = 3)$	524	<b>26.3</b>	3.1
ShuffleNet $2\times$ (with SE[13], $g = 3$ )	527	<b>24.7</b>	4.7
0.75 MobileNet-224	325	31.6	-
ShuffleNet $1.5\times (g = 3)$	292	<b>28.5</b>	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet $1\times (g = 8)$	140	<b>32.4</b>	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet $0.5\times (g = 4)$	38	<b>41.6</b>	7.8
ShuffleNet $0.5\times$ (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.

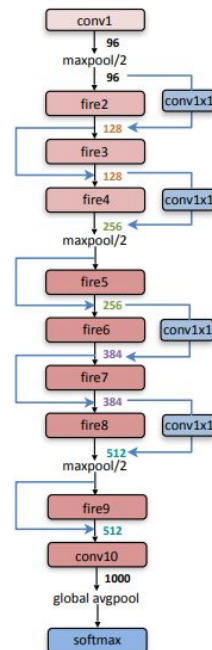
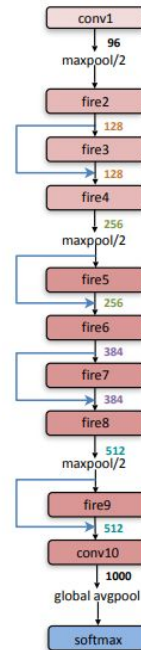
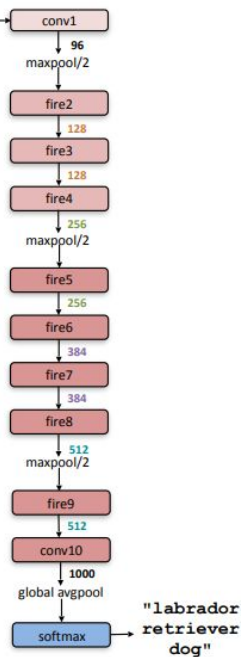
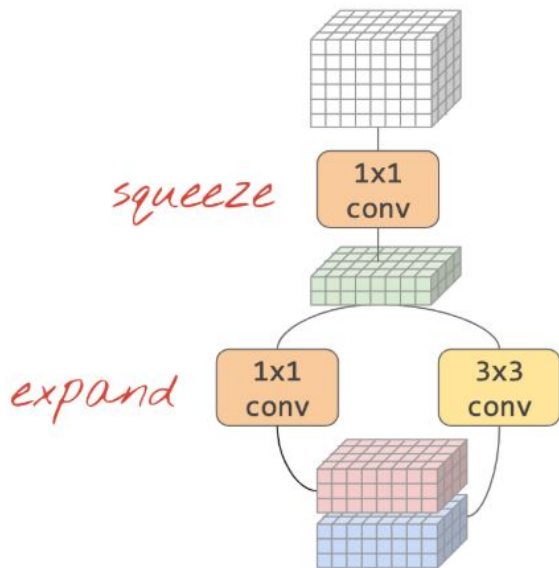


# SqueezeNet



- 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.

# SqueezeNet

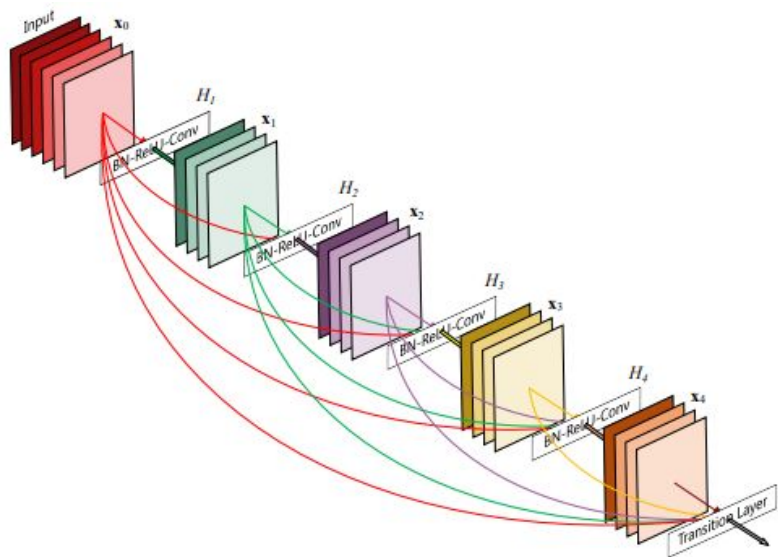


# SqueezeNet

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	<b>50x</b>	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	<b>363x</b>	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	<b>510x</b>	57.5%	80.3%

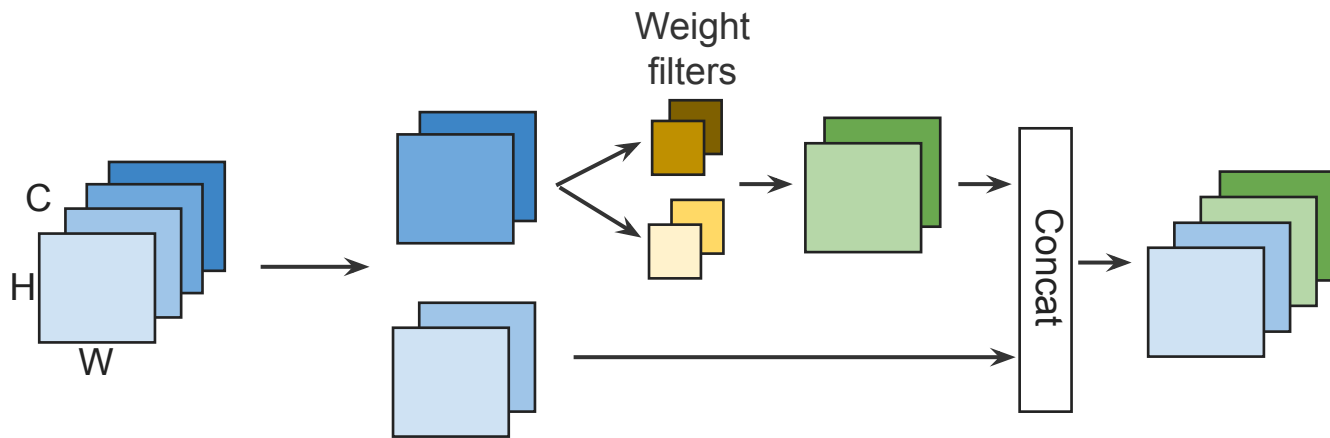
- Achieve a comparable performance as AlexNet, but still suboptimal compare against other architectures.
- ResNet 50: 100MB, Vision Transformer base > 300MB.

# DenseNet

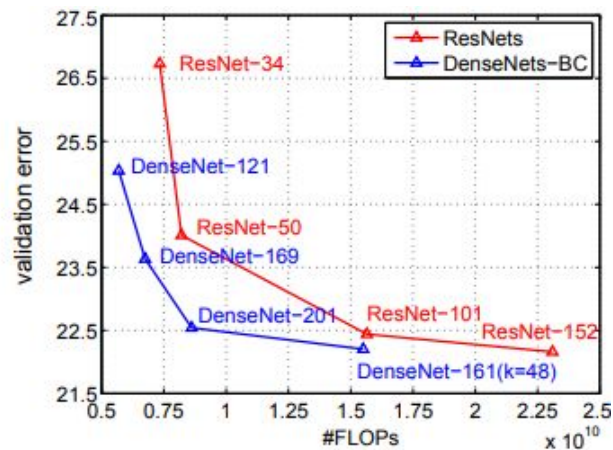
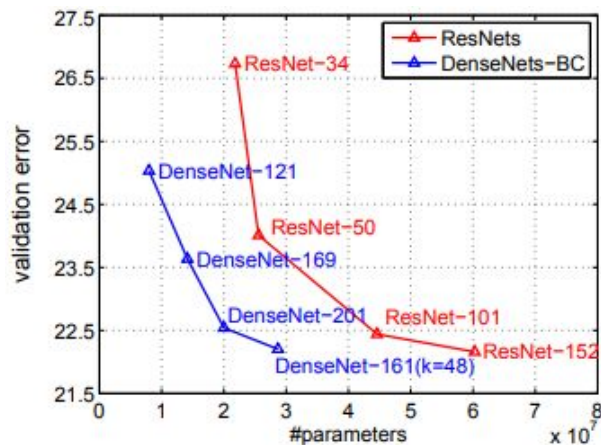


- ResNet:  
$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}$$
- DenseNet:  
$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\ell-1}])$$
- $H(\cdot)$  is the function of batch normalization, followed by ReLU and 3x3 Convolution.

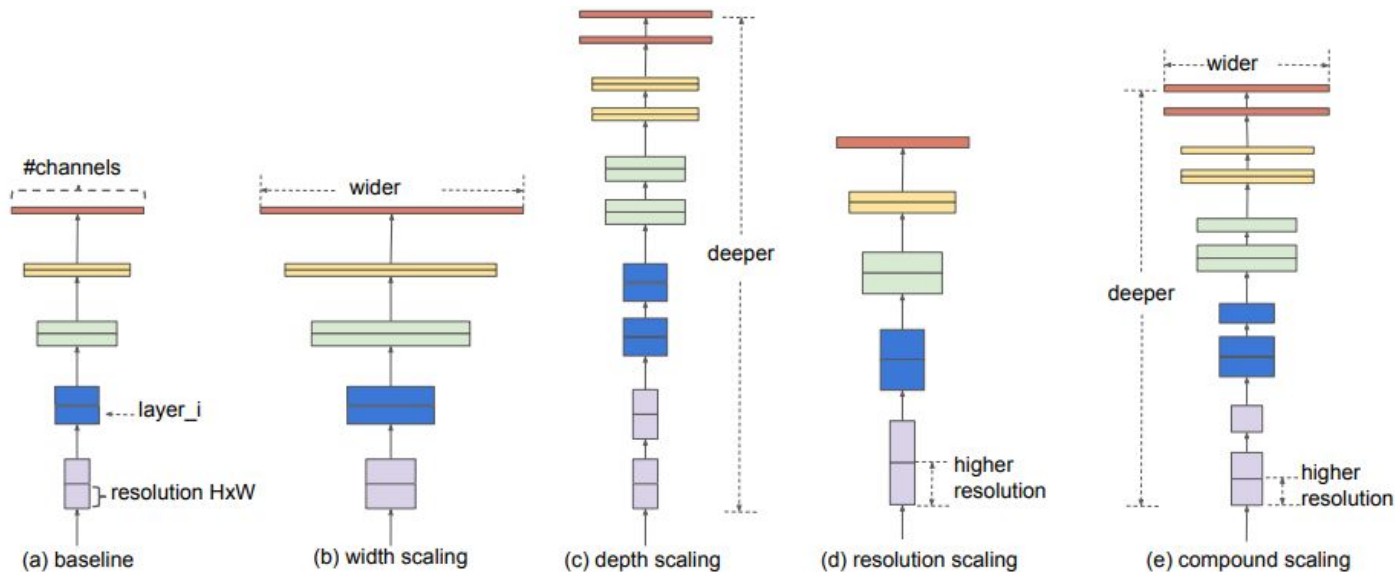
# DenseNet



# DenseNet

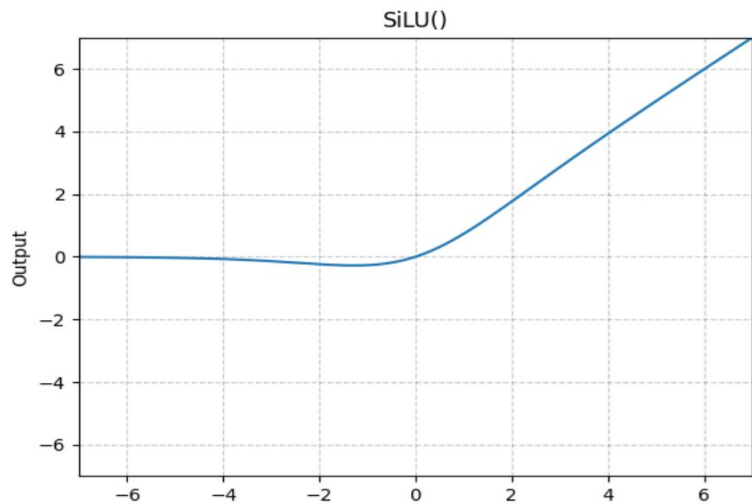


# EfficientNet

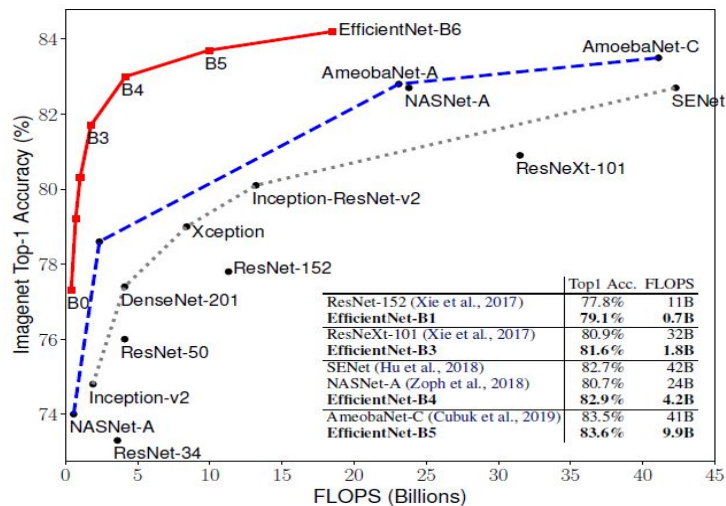


- It is critical to balance all dimensions of network width/depth/resolution, and surprisingly such balance can be achieved by simply scaling each of them with constant ratio.

# EfficientNet

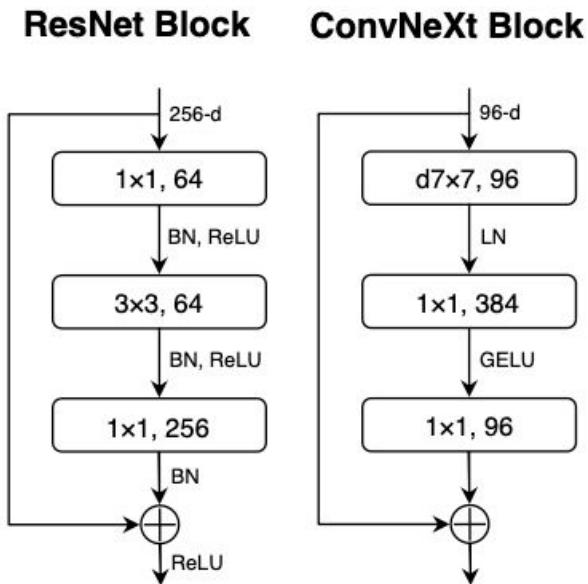


- SiLU is used in the EfficientNet architecture.
- $\text{SiLU}(x) = x * \sigma(x)$

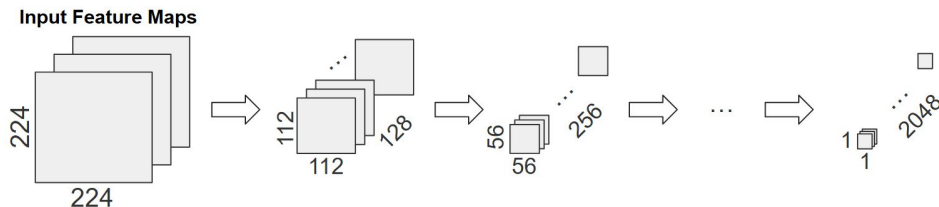




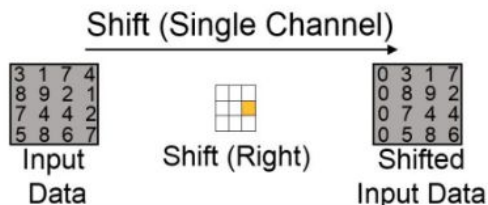
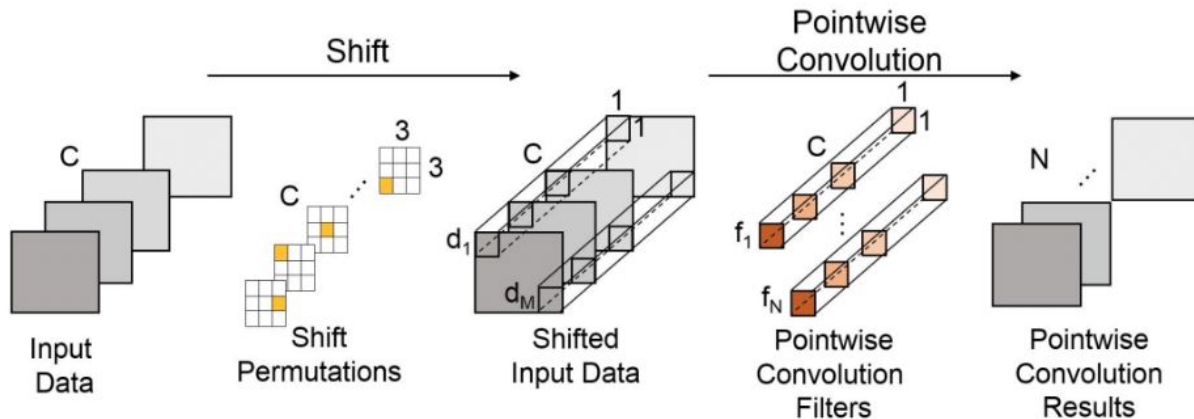
# ConvNext



- Leverage the insight of vision transformer 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.

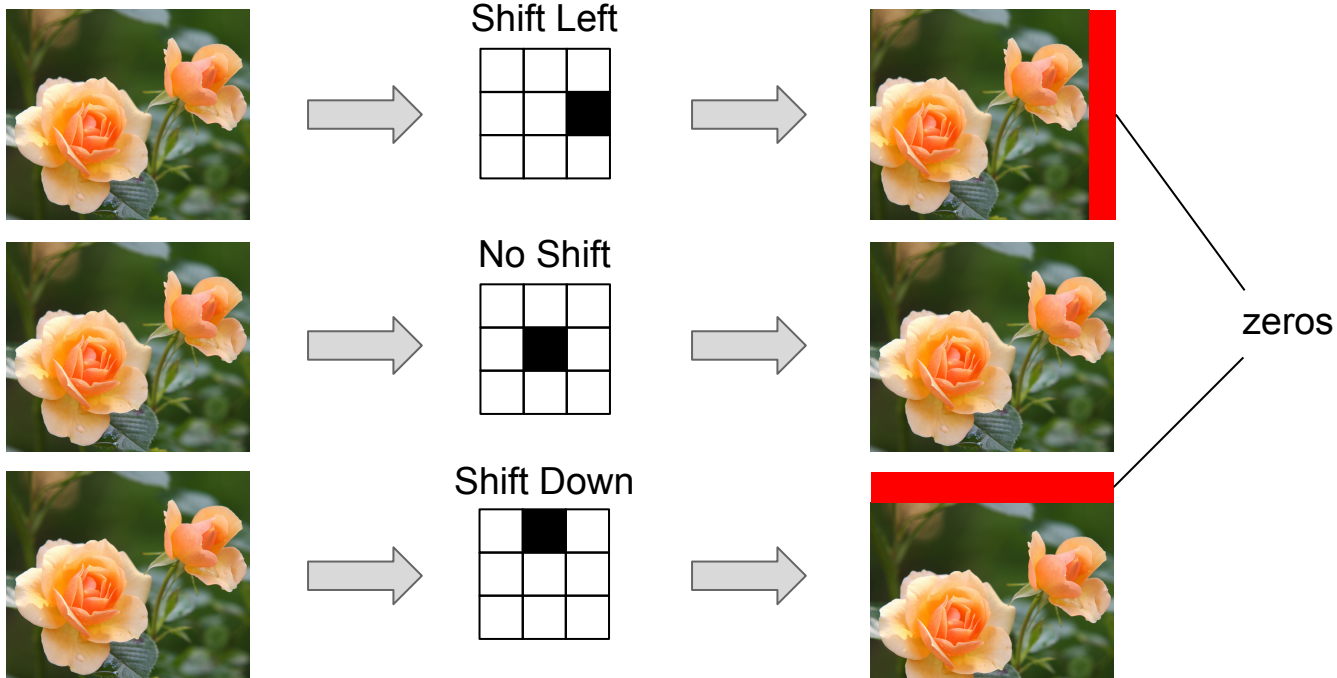


# ShiftNet

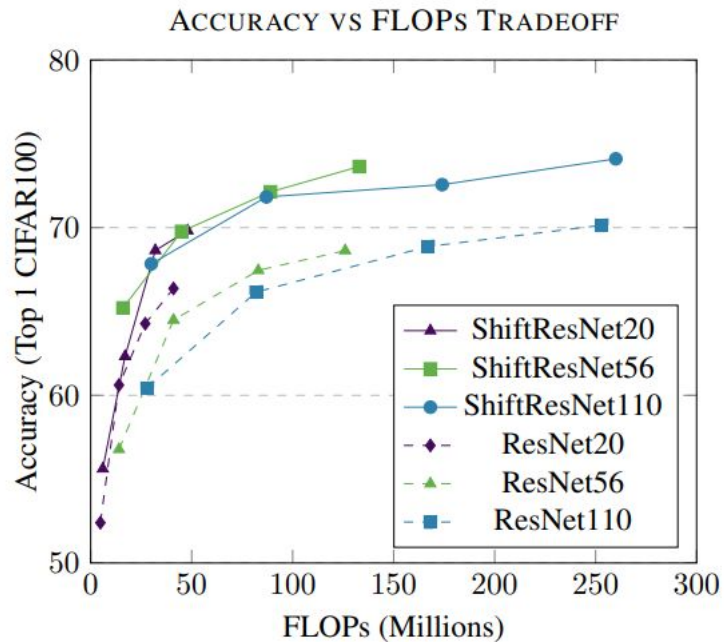
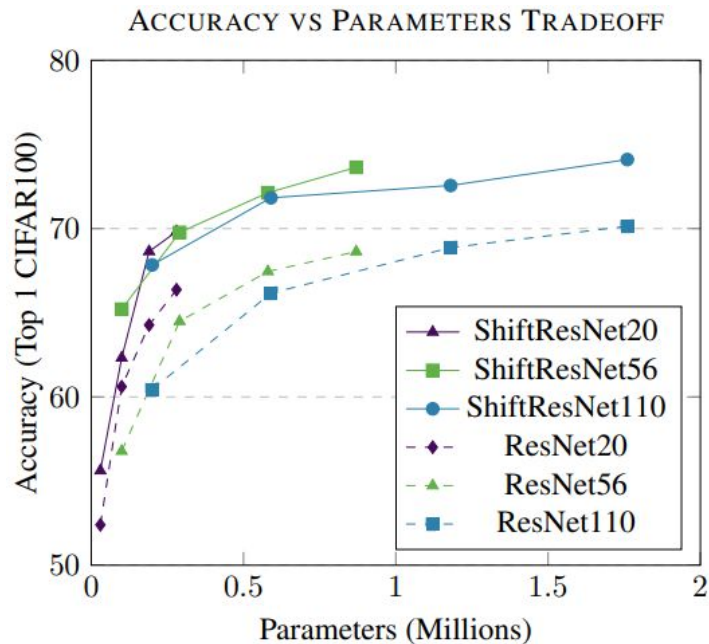


- Completely remove the computation for the depthwise convolution.
- The shift positions are predefined for each channel.

# ShiftNet

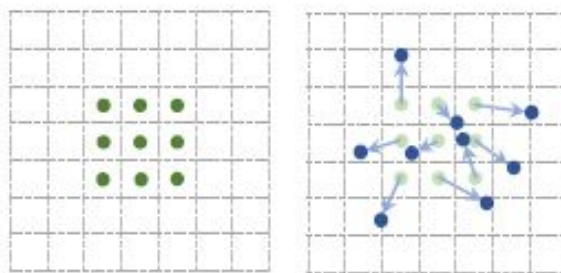


# ShiftNet



# 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”.



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

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

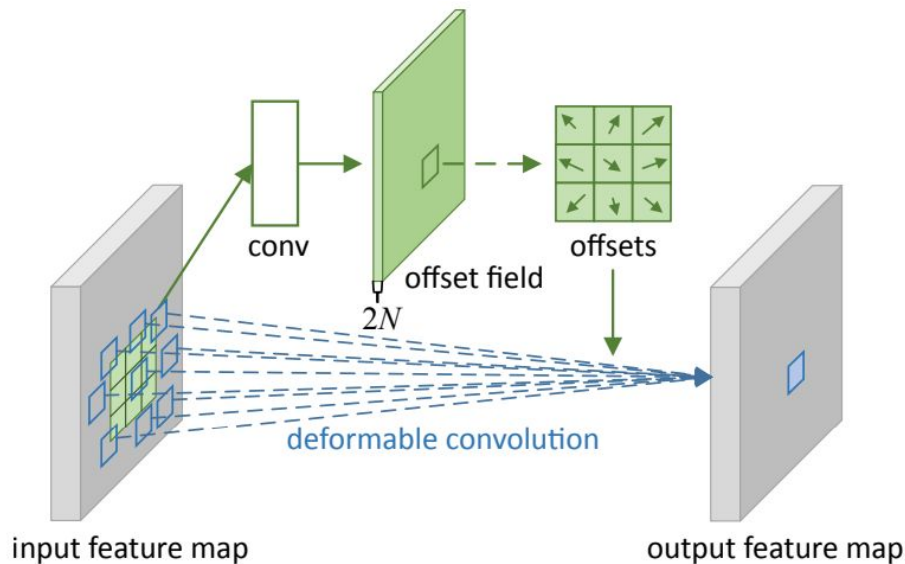
# Deformable Convolutional Networks

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



$$y(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} w(\mathbf{p}_n) \cdot x(\underbrace{\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n}_{\text{Can be fractional}})$$

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



# 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)	<b>75.2</b>	<b>75.2</b>	74.5	57.2	78.6	63.3	81.4	64.7
res5 & res4b22,b21,b20 (6)	74.8	75.1	<b>74.6</b>	<b>57.7</b>	<b>78.7</b>	<b>64.0</b>	<b>81.5</b>	<b>65.4</b>

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

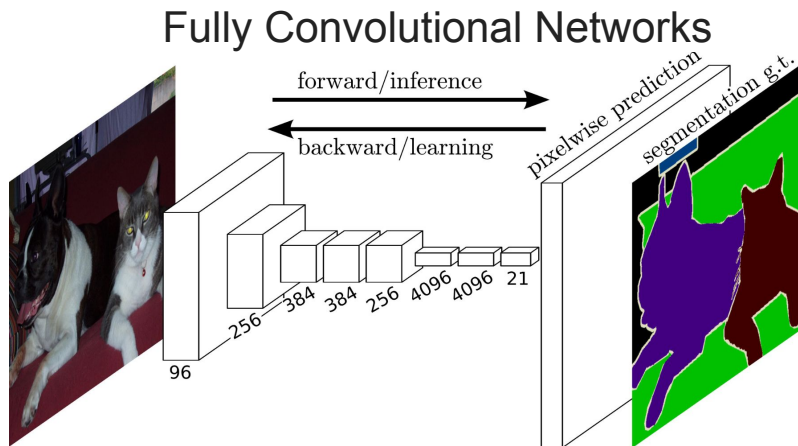


# Topics

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



# CNNs for Other Tasks: Image Segmentation



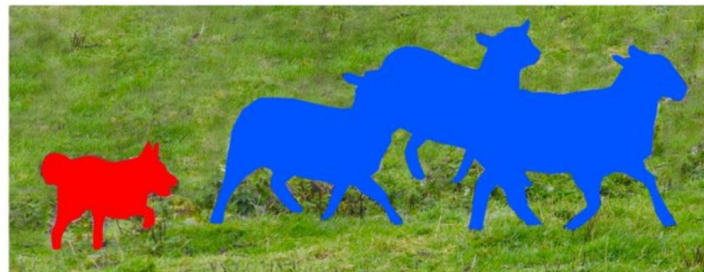
- A fully convolutional based DNN for image segmentation.

- 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.

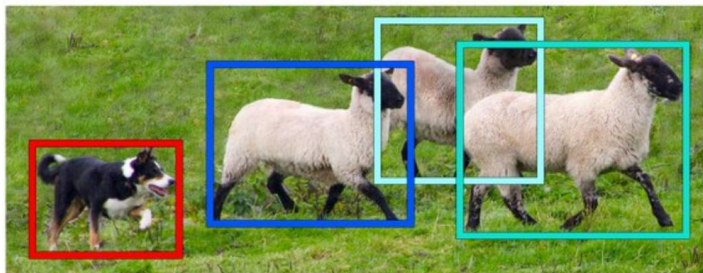
# Image Segmentation



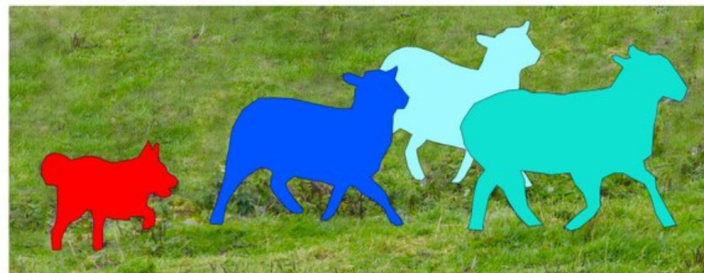
Image Recognition



Semantic Segmentation

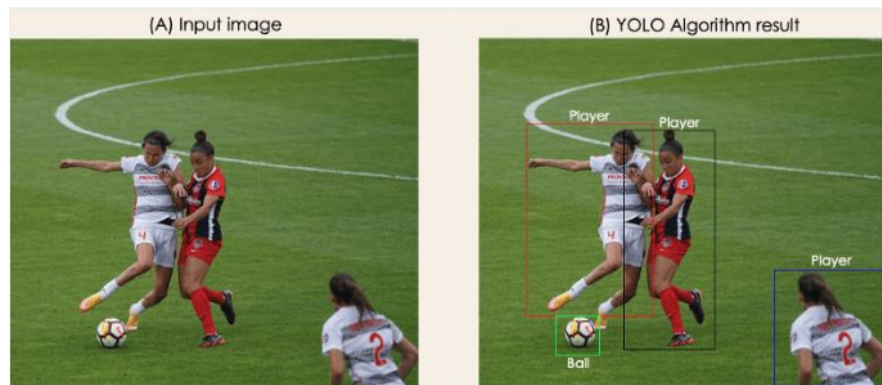
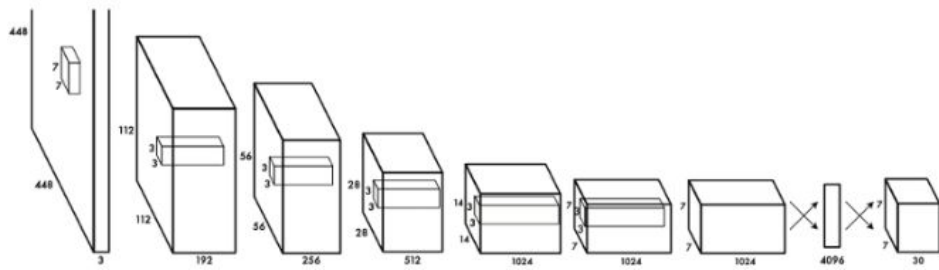


Object Detection



Instance Segmentation

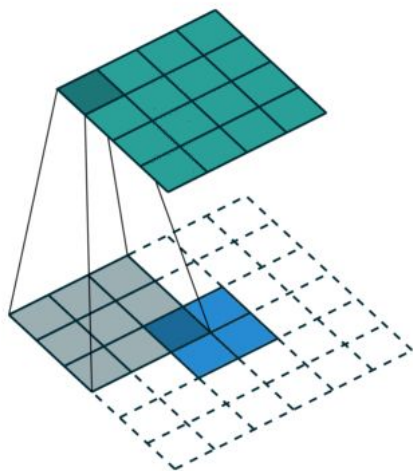
# CNNs for Other Tasks: Object Detection



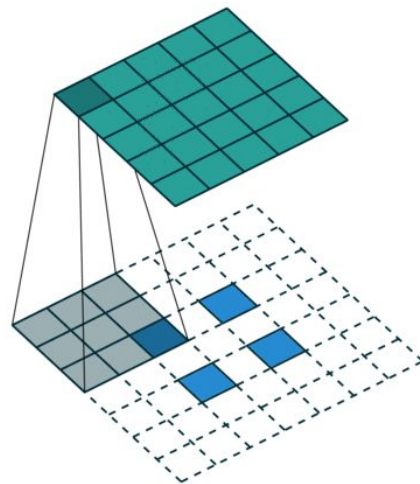
- YOLO will generate the likelihood of each anchor point and the coordinates of its bounding box.
- Another branch will produce the category of each bounding box

# Transposed Convolution

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

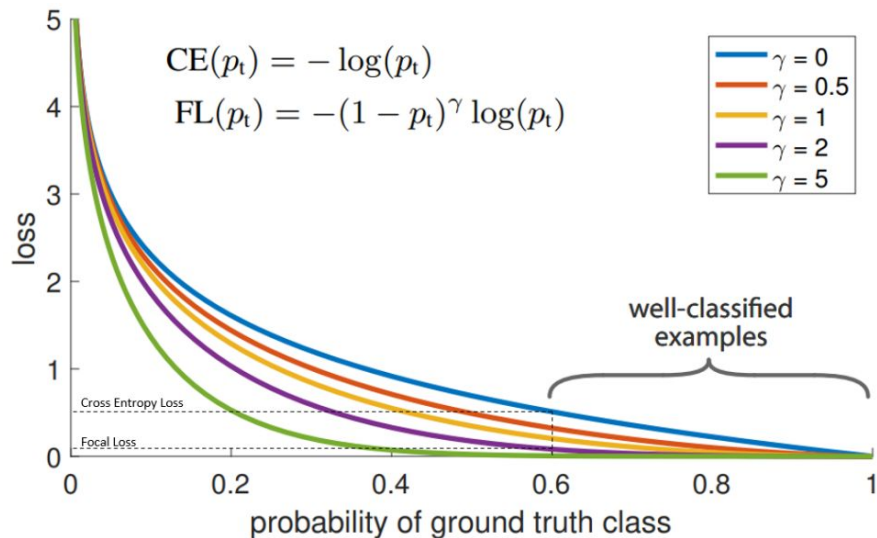


Stride = 1



Stride = 2

# Focal Loss

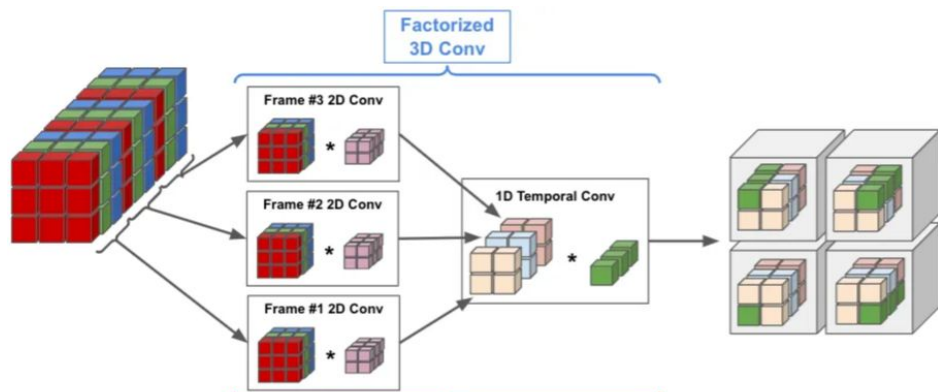


- A modified cross-entropy designed to perform better with class imbalance.
- Often used in the problem of object detection and image segmentation.
  - Down-weight easy examples and thus focus training on hard negatives

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

- $\gamma$  controls the shape of the curve
- $\alpha$  controls the class imbalance and introduce weights to each class.

# CNNs for Other Tasks: Video Processing



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

# Presentations

- [You Only Look Once: Unified, Real-Time Object Detection](#)
- [EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](#)
- [Focal Loss for Dense Object Detection](#)
- [TSM: Temporal Shift Module for Efficient Video Understanding](#)

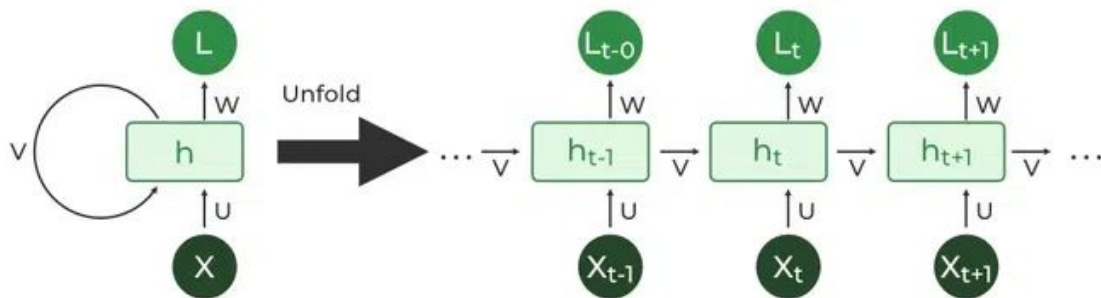
# Topics

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

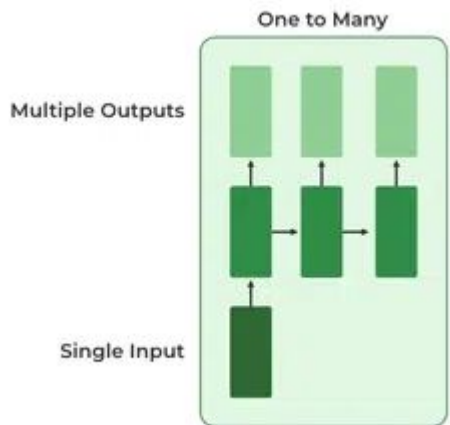


# Recurrent Neural Networks (RNNs)

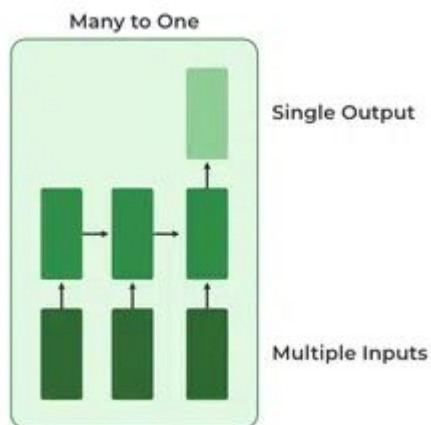
- Recurrent Neural Network (RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step.
- RNN is widely used to process Sequential data
  - Text
  - Time series
  - Video
- The output corresponding to the current input is related to all the previous inputs and outputs.



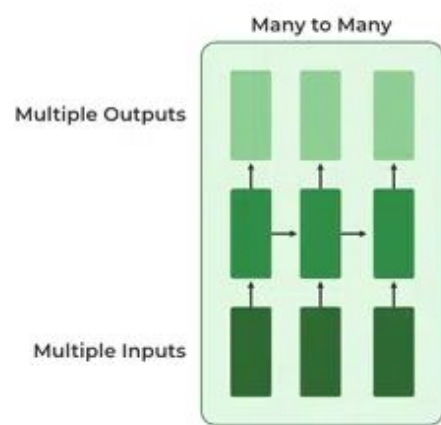
# Recurrent Neural Networks (RNNs)



One-to-many  
Image captioning



Many-to-one  
Text classification



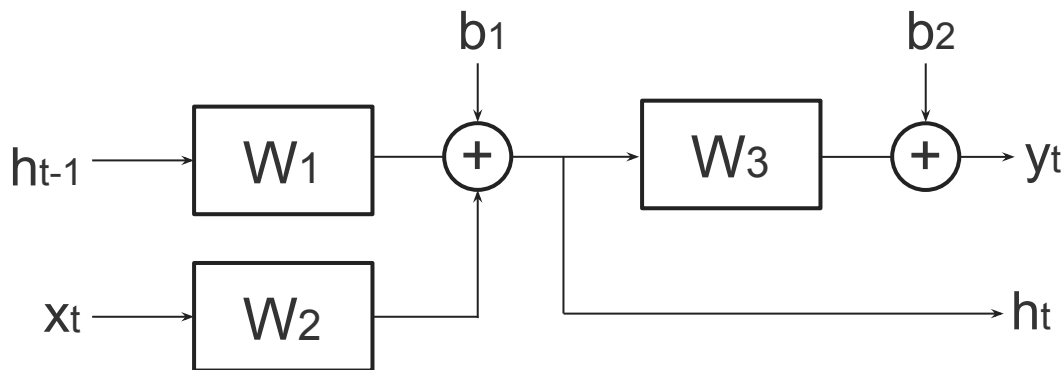
Many-to-many  
Translation

# Recurrent Neural Networks (RNNs)

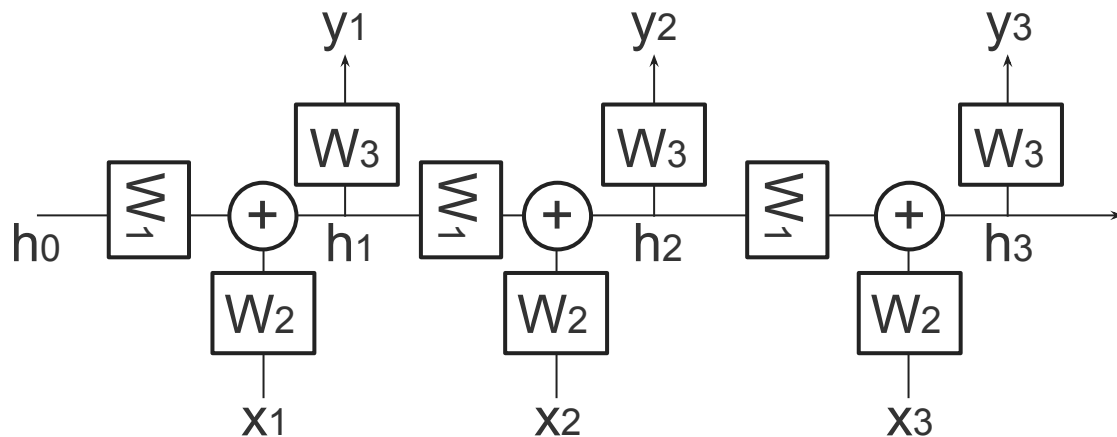
$$h_t = f_1(W_1 h_{t-1} + W_2 x_t + b_1)$$

$$y_t = f_2(W_3 h_t + b_2)$$

- $y_t, x_t$ : output/input at timestep  $t$
- $h_t$ : hidden state
- $W_1, W_2, W_3$ : parameters
- $b_1, b_2$ : biases



# Recurrent Neural Networks (RNNs)



$$\frac{dL}{dW_2} = \frac{dL}{dh_3} x_3 + \frac{dL}{dh_3} \frac{dh_3}{dh_2} x_2 + \frac{dL}{dh_3} \frac{dh_3}{dh_2} \frac{dh_2}{dh_1} x_1$$

# Problems of RNNs

- **Vanishing Gradient:** It occurs when the gradients used to update the network's weights become exceedingly small, effectively preventing the network from learning long-range dependencies.
- **Gradient Explosion:** gradients grow excessively large during training, leading to unstable updates.
- To prevent this, gradient clipping is used to cap gradients at a predefined threshold, ensuring stable and effective training.