Chapter 1 - Section 3

CNN & High-level Feature Extraction

Dr. Jifeng Dai

Tuesday, March 8, 2022

Acknowledge : Song Guanglu , Liu Boxiao , Zhang Manyuan

3.1 Neural Network Basics

Dr. Jifeng Dai

Tuesday, March 8, 2022

商汤 sensetime

清華大學 Tsinghua University

Outline

Part 1	Neural Network Overview
Part 2	Activation functions and gradient descent
Part 3	Deep L-layer neural network
Part 4	Regularization and optimization

商汤 sensetime

清華大學 Tsinghua University

Highlights

Learn the forward and backward propagation of neural network

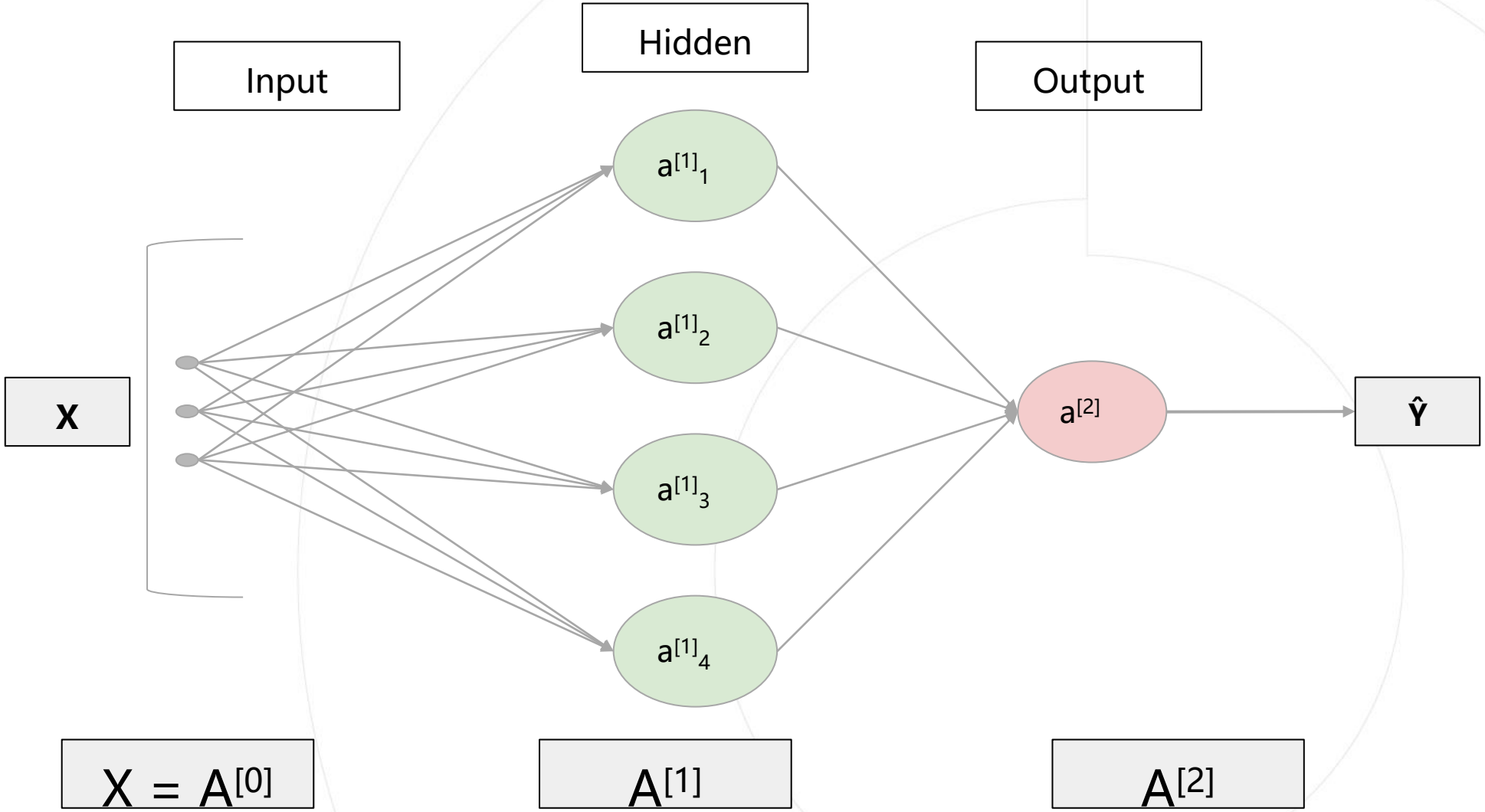
Learn the activation functions in neural networks

Learn the hyper-parameters in neural network training

Understand the regularization methods and optimization methods in neural network

Understand the phenomenon of overfitting in neural network training and its solution

Neural Network Overview



Neural Network Overview

Input → Hidden

$$\left.\begin{matrix} x \\ W^{[1]} \\ b^{[1]} \end{matrix}\right\} = z^{[1]} = W^{[1]}x + b^{[1]} \Longrightarrow a^{[1]} = \sigma\left(z^{[1]}\right)$$

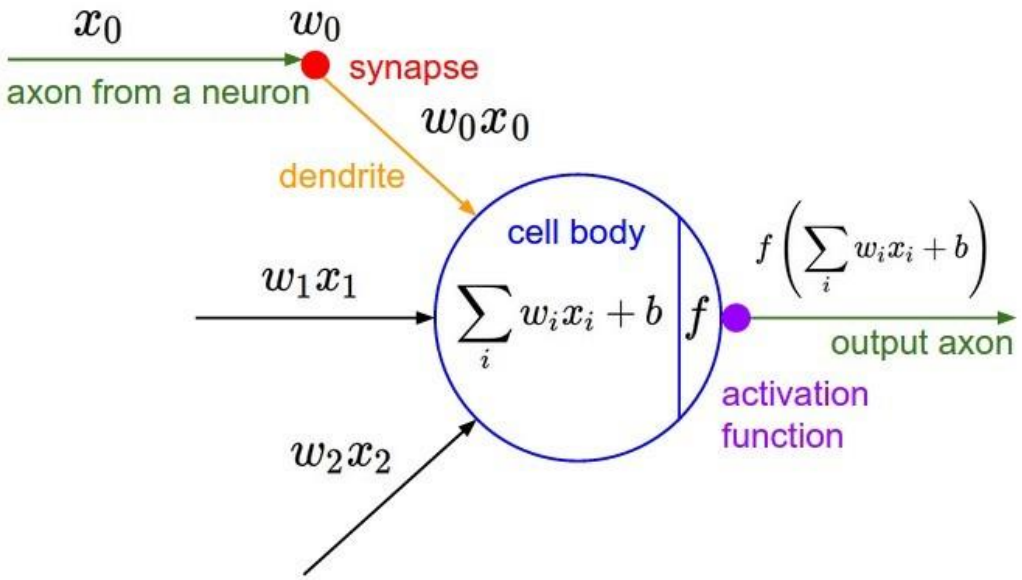
Hidden → Output

$$\left.\begin{matrix} a^{[1]} = \sigma\left(z^{[1]}\right) \\ W^{[2]} \\ b^{[2]} \end{matrix}\right\} \Rightarrow z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \Longrightarrow a^{[2]} = \sigma\left(z^{[2]}\right) \Longrightarrow L\left(a^{[2]}, y\right)$$

Derivation

$$\left.\begin{matrix} da^{[1]} = d\sigma\left(z^{[1]}\right) \\ dW^{[2]} \\ db^{[2]} \end{matrix}\right\} \Longleftarrow dz^{[2]} = d\left(W^{[2]}a^{[1]} + b^{[2]}\right) \Longleftarrow da^{[2]} = d\sigma\left(z^{[2]}\right) \Longleftarrow dL\left(a^{[2]}, y\right)$$

- Computing a Neural Network's output



$$z = w^T x + b$$

$$a = \sigma(z)$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma\left(z^{[1]}\right)$$

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \overbrace{\begin{bmatrix} \cdots W_1^{[1]T} \cdots \\ \cdots W_2^{[1]T} \cdots \\ \cdots W_3^{[1]T} \cdots \\ \cdots W_4^{[1]T} \cdots \end{bmatrix}}^{W^{[1]}} * \overbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}^{input} + \overbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}^{b^{[1]}}$$

Neural Network Overview

- Vectorizing across multiple examples

$$x = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \qquad A^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \alpha^{[1](1)} & \alpha^{[1](2)} & \cdots & \alpha^{[1](m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \qquad Z^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$W^{[1]}x = \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ w^{(1)}x^{(1)} & w^{(1)}x^{(2)} & w^{(1)}x^{(3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = Z^{[1]}$$

$$\left.\begin{aligned} z^{[1](i)} &= W^{[1](i)}x^{(i)} + b^{[1]} \\ \alpha^{[1](i)} &= \sigma\left(z^{[1](i)}\right) \\ z^{[2](i)} &= W^{[2](i)}\alpha^{[1](i)} + b^{[2]} \\ \alpha^{[2](i)} &= \sigma\left(z^{[2](i)}\right) \end{aligned}\right\} \Rightarrow \begin{cases} A^{[1]} = \sigma\left(z^{[1]}\right) \\ z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} = \sigma\left(z^{[2]}\right) \end{cases}$$

$a^{[2](i)}, (i)$ 指第个i 训练样本，而[2]是指第二层

商汤 sensetime

清華大學 Tsinghua University

Outline

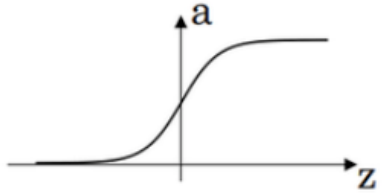
Part 1	Neural Network Overview
Part 2	Activation functions and gradient descent
Part 3	Deep L-layer neural network
Part 4	Regularization and optimization

Activation functions

- Four activation functions

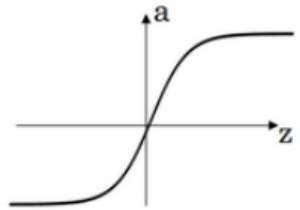
1) sigmoid activation function

$$g(z) = \frac{1}{1 + e^{-z}}$$



2) Tanh activation function

$$g(z) = \tanh(z) = \frac{e^{z} - e^{z}}{e^{y} + e^{-z}}$$

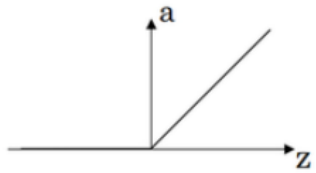


$$a^{[1]} = \sigma\left(z^{[1]}\right)$$

Activation functions

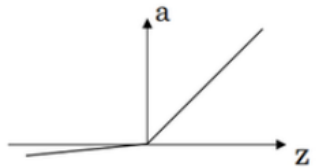
3) Rectified Linear Unit (ReLU)

$$g(z) = \max(0, z)$$



4) Leaky linear unit (Leaky ReLU)

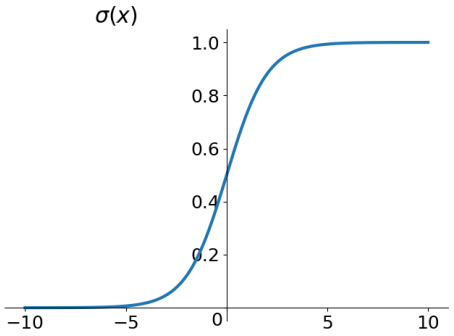
$$g(z) = \max(0.01z, z)$$

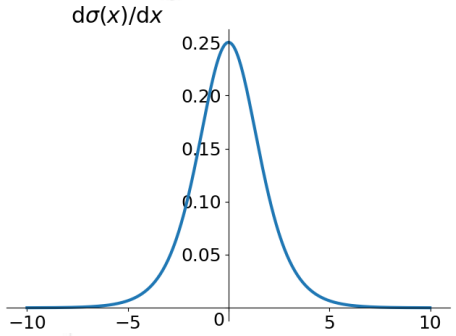


Activation functions

- Derivatives of activation functions

1) sigmoid activation function



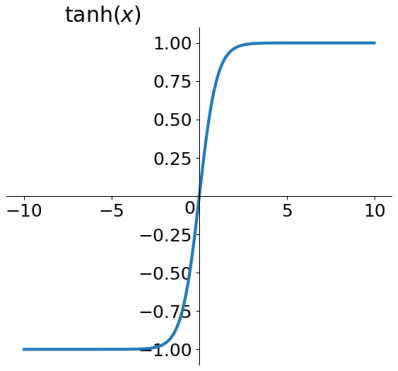


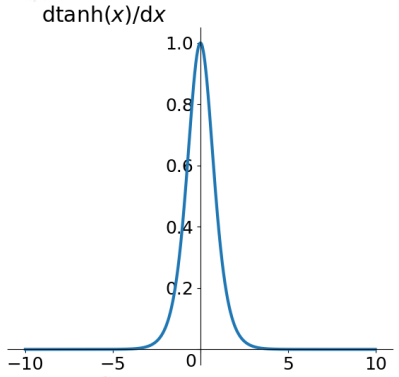
$$\frac{d}{dz}g(z) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(z)(1-g(z))$$

Drawbacks:

(1) gradient vanishing
(2) output is not zero-centered (slow the convergence)
(3) power operation is time-consuming

2) Tanh activation function





$$\frac{d}{dz}g(z) = 1 - (\tanh(z))^2$$

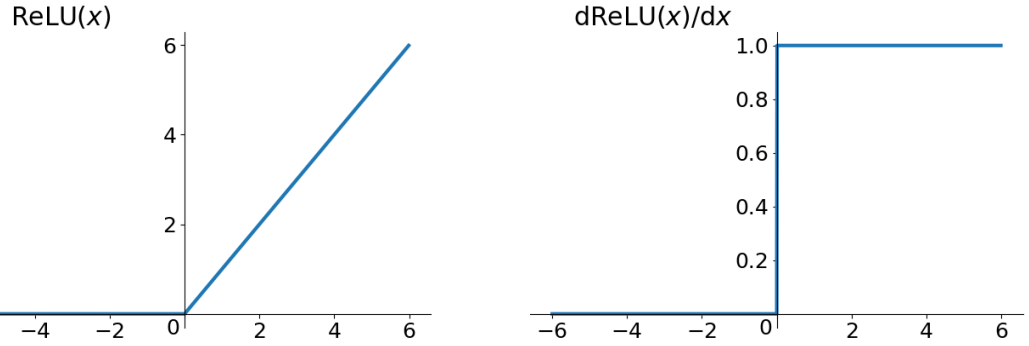
Drawbacks:

(1) gradient vanishing
(2) power operation is time-consuming

Activation functions

- Derivatives of activation functions

3) Rectified Linear Unit (ReLU)



$$g(z)' = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$$

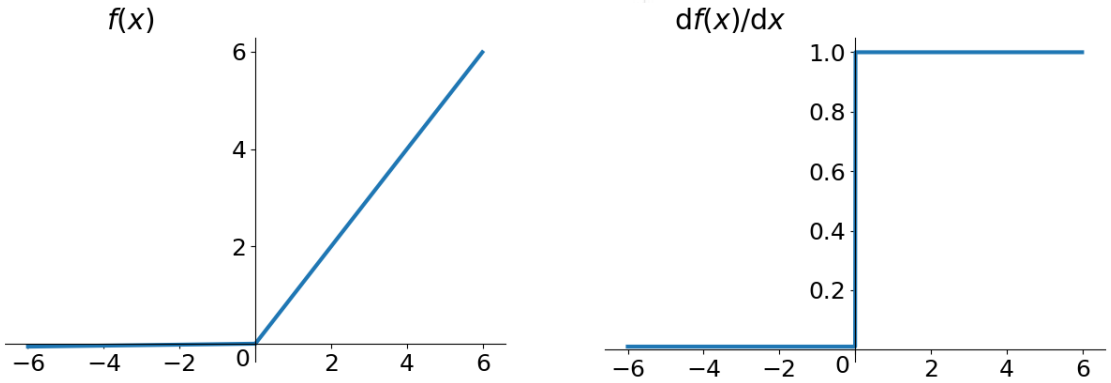
Features:

(1) eliminate gradient vanishing
(2) fast computation
(3) fast convergence than sigmoid and tanh

Drawbacks:

(1) Dead ReLU Problem

4) Leaky linear unit (Leaky ReLU)



$$g(z)' = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$$

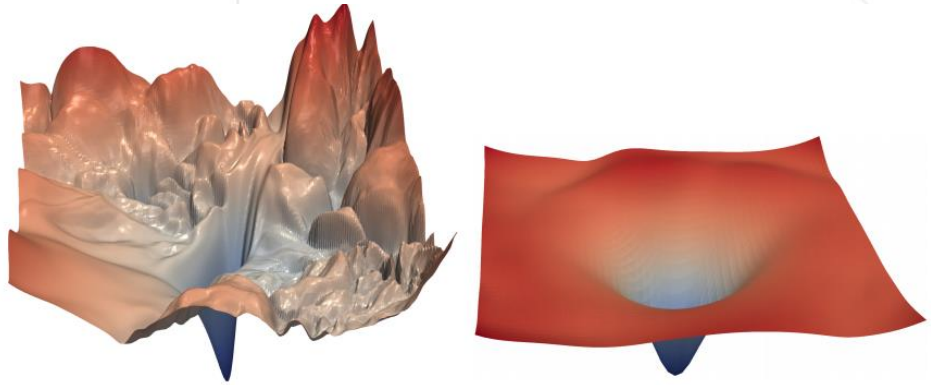
Gradient descent

- Gradient descent for one layer neural networks

For one layer neural network

$$\hat{y} = \sigma\left(w^T x + b\right)$$
$$\sigma(z) = \frac{1}{1+e^{-z}}$$
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$
$$= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log\hat{y}^{(i)} + \left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)$$

maximum likelihood



The loss surfaces of ResNet-56 with/without skip connections.

$$\underbrace{x, w, b} \iff \underbrace{z = w^T x + b} \iff \underbrace{a = \sigma(z) \Longleftarrow L(a,y)}$$

$$dw{=}dz\cdot x, db{=}dz \qquad dz{=}da\cdot g'(z), g(z){=}\sigma(z), \frac{dL}{dz}{=}\frac{dL}{da}\cdot\frac{da}{dz}, \frac{d}{dz}g(z){=}g'(z) \qquad da{=}\frac{d}{da}L(a,y){=}(-y\log\alpha{-}(1{-}y)\log(1{-}a))'{=}-\frac{y}{a}{+}\frac{1-y}{1-a}$$

商汤 sensetime

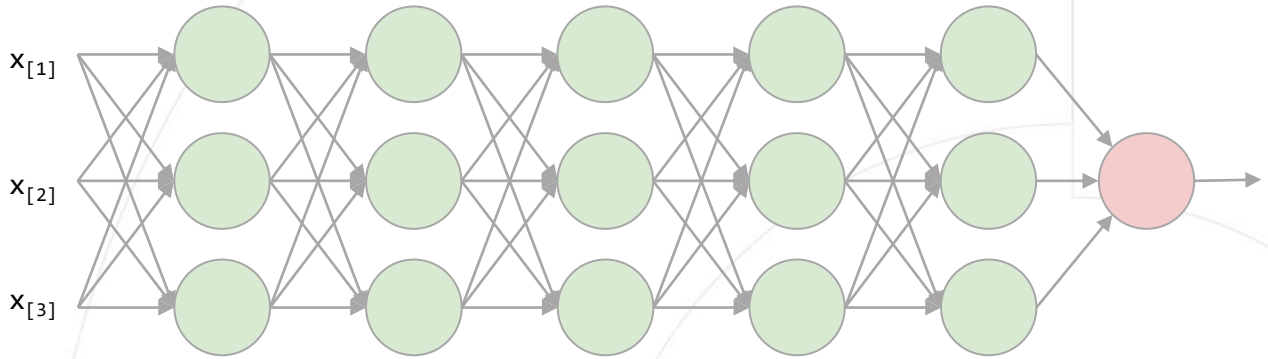
清華大學 Tsinghua University

Outline

Part 1	Neural Network Overview
Part 2	Activation functions and gradient descent
Part 3	Deep L-layer neural network
Part 4	Regularization and optimization

Deep L-layer neural network

- Forward and backward for Deep L-layer neural network



$$Forward$$
$$input: a^{[l-1]}$$
$$output: a^{[l]}$$
$$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}\left(z^{[l]}\right)$$

$$Backward$$
$$input: da^{[l]}$$
$$output: da^{[l-1]}, dw^{[l]}, db^{[l]}$$
$$dz^{[l]} = da^{[l]} * g^{[l]'}\left(z^{[l]}\right)$$
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$
$$db^{[l]} = dz^{[l]}$$
$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$
$$dz^{[l]} = w^{[l+1]T} dz^{[l+1]} \cdot g^{[l]'}\left(z^{[l]}\right)$$

商汤 sensetime

清華大學 Tsinghua University

Outline

Part 1	Neural Network Overview
Part 2	Activation functions and gradient descent
Part 3	Deep L-layer neural network
Part 4	Regularization and optimization

Regularization and optimization

- Parameters vs Hyperparameters

- Parameters

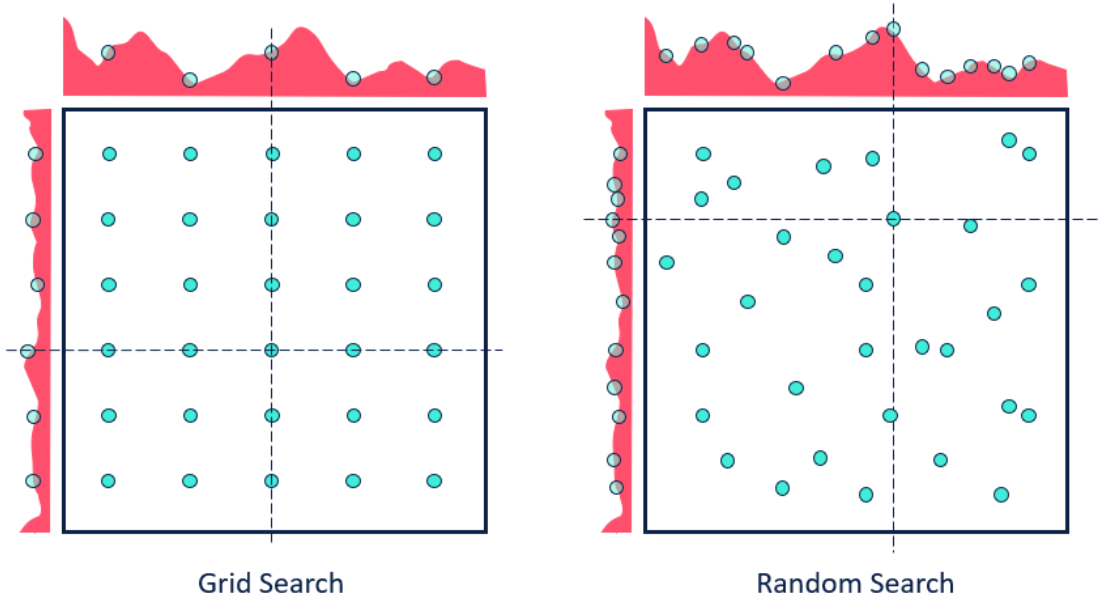
$$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \ldots$$

- Hyperparameters

  - learning rate
  - iterations
  - *L* (Number of hidden layers)
  - *n* (Number of hidden layer neurons)
  - choice of activation function
  - momentum
  - mini batch size
  - regularization parameters

  … …

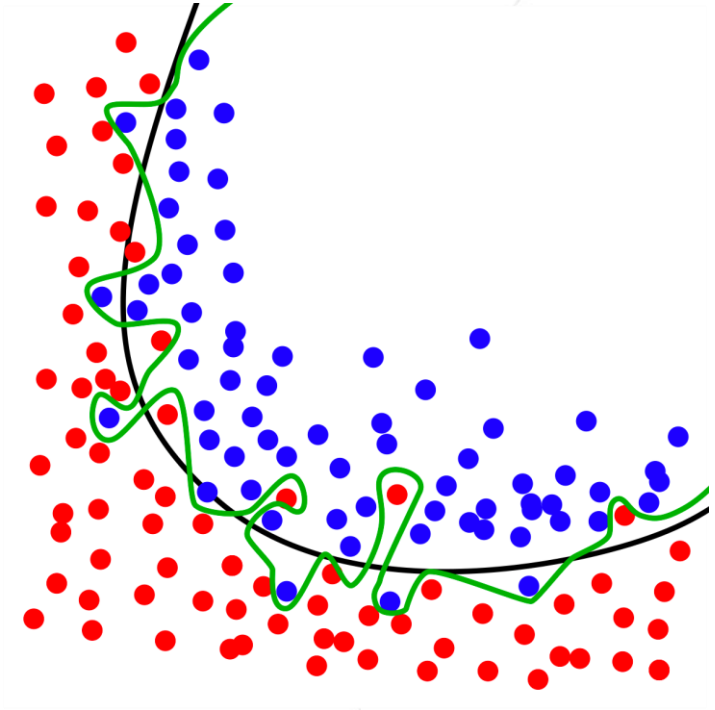
- Tune hyperparameters with grid search or random search



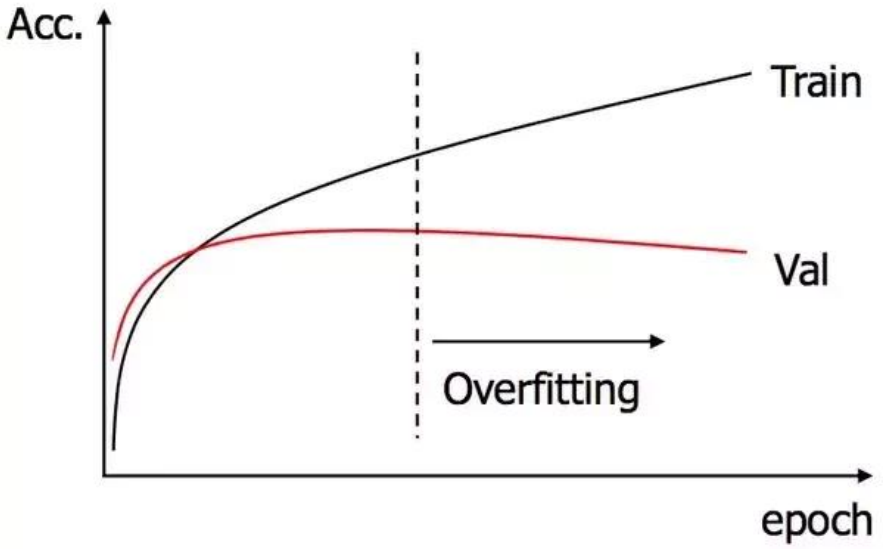
https://community.alteryx.com/t5/Data-Science/Hyperparameter-Tuning-Black-Magic/ba-p/449289

Regularization and optimization

- Overfitting



https://en.wikipedia.org/wiki/Overfitting



https://www.quora.com/Which-signals-do-indicate-that-the-convolutional-neural-network-is-overfitted

Regularization and optimization

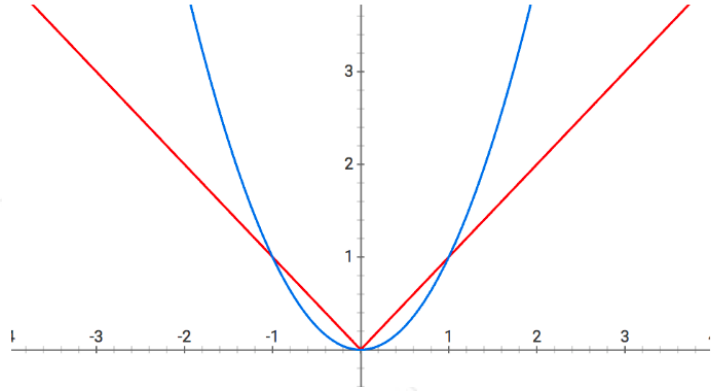
- Regularization

L1 penalty

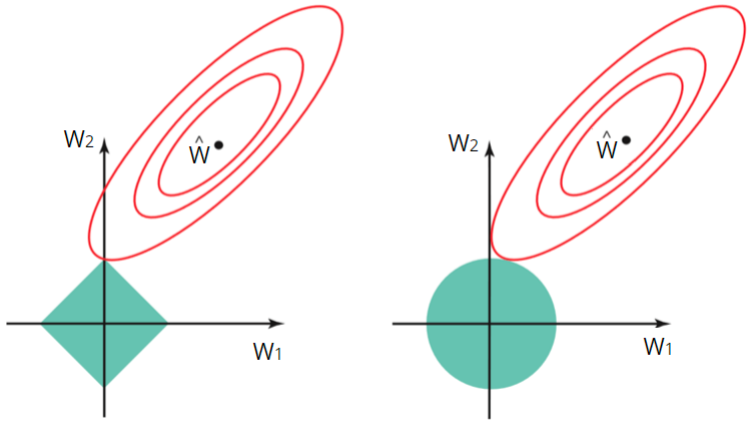
$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n}|\theta_i|$$

L2 penalty

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n}\theta_i^2$$



L1 penalty	**L2 penalty**
L1 penalizes sum of absolute values of weights.	L2 penalizes sum of square values of weights.
L1 generates model that is simple and interpretable.	L2 regularization is able to learn complex data patterns.
L1 is robust to outliers.	L2 is not robust to outliers.



Source: An Introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Regularization and optimization

- Weight Initialization matters

  - $W^{[l]}$-weight matrix of dimension (size of layer l, size of layer l -1)
  - $b^{[l]}$-bias vectors of dimension (size of layer l)

- Initialization with value 0?

  No!
  If the weight is zero, the outputs of all neural node are same.
  The gradient is same! The weight update is same!
  We can't accept this.

Regularization and optimization

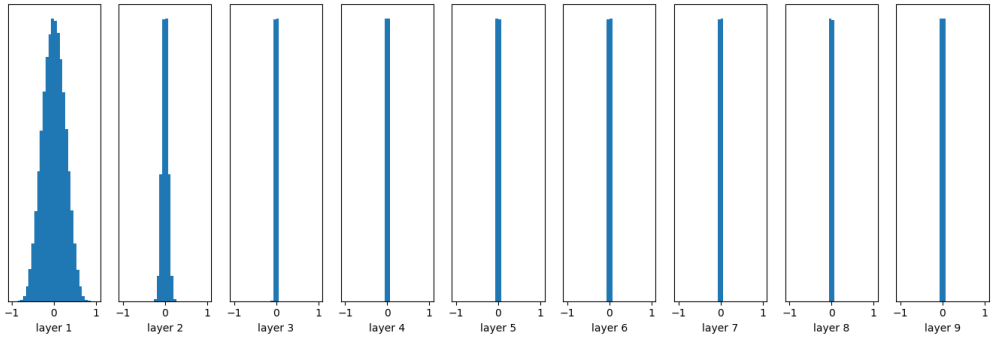
- Weight Initialization matters

  - $W^{[l]}$-weight matrix of dimension (size of layer l, size of layer l-1)
  - $b^{[l]}$-bias vectors of dimension (size of layer l)

- Random initialization

```
W = np.random.randn(node_in, node_out)
```

e.g. We create a neural network with 10 layers and adopt the tanh activation function.
We initialize the W with a mean of 0 and a standard deviation of 0.01.



At the end of neural network, the output is close to 0.
This leads to a small gradient and hard to update the W.

Regularization and optimization

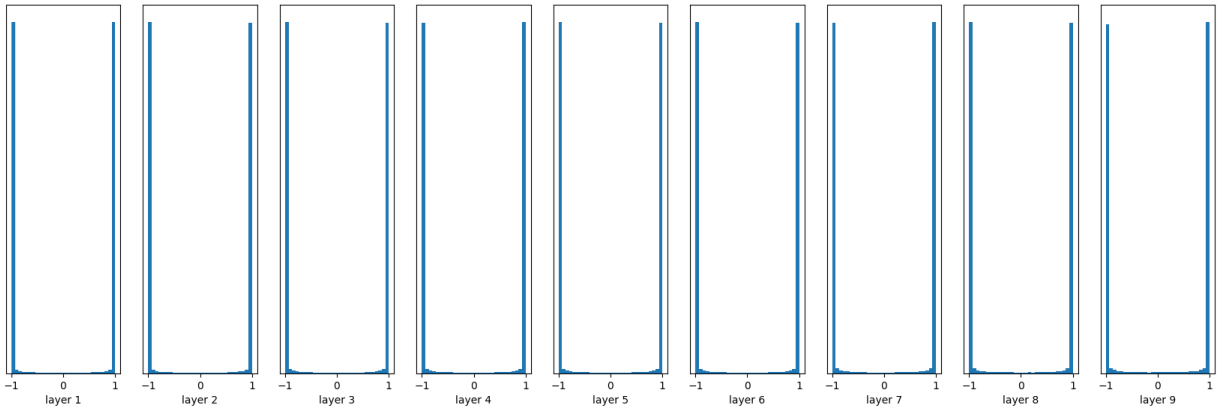
- Weight Initialization matters

  - $W^{[l]}$-weight matrix of dimension (size of layer l, size of layer l -1)
  - $b^{[l]}$-bias vectors of dimension (size of layer l)

- Random initialization

```
W = np.random.randn(node_in, node_out)
```

e.g. We create a neural network with 10 layers and adopt the tanh activation function.
We initialize the W with a mean of 0 and a standard deviation of 1.



The output is close to -1 or 1.
The gradient of tanh is close to 0
This leads to a small gradient and hard to update the W.

Regularization and optimization

- Analysis

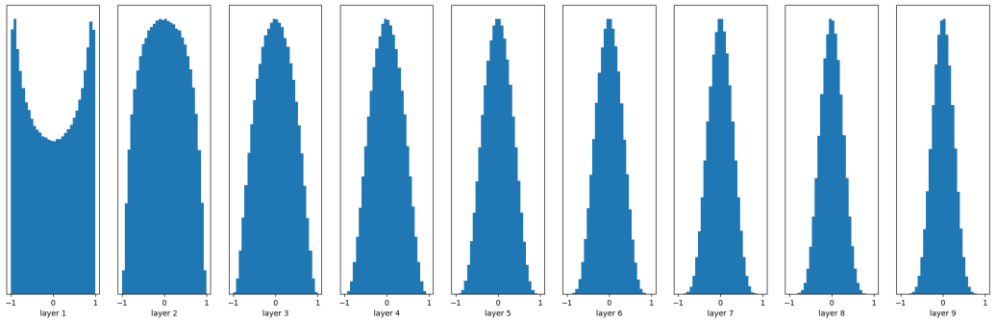
  Deep NN models have difficulties in converging when the weights are initialized using Normal Distribution with *fixed standard deviation*. This is because the variance of weights is not taken care of, which leads to very large or small activation values, resulting in exploding or vanishing gradient problem during backpropagation. This problem worsens as the depth of NN is increased.

- Xavier initialization

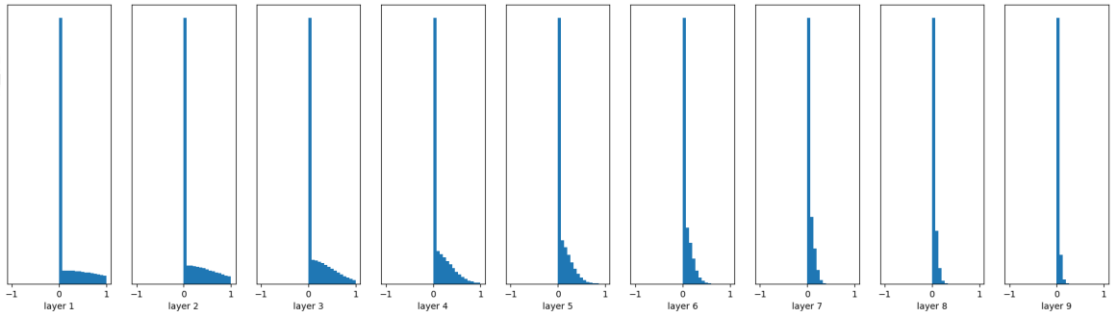
  It tries to keep variance of all the layers equal.

```
np.random.randn(node_in, node_out) / np.sqrt(node_in)
```

For tanh



For ReLU



- Xavier initialization – simple **derivation**

(1) It tries to keep variance of all the layers equal.

`np.random.randn(node_in, node_out) / np.sqrt(node_in)`

$$\boldsymbol{y} = w\boldsymbol{x} = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Input x and output y

$$\mathrm{Var}(W_i X_i) = [E(X_i)]^2 \,\mathrm{Var}(W_i) + [E(W_i)]^2 \,\mathrm{Var}(X_i) + \mathrm{Var}(X_i)\,\mathrm{Var}(W_i)$$

We have E(x) = 0, E(W) = 0

$$\mathrm{Var}(W_i X_i) = \mathrm{Var}(X_i)\,\mathrm{Var}(W_i)$$

$$\mathrm{Var}(\boldsymbol{y}) = \mathrm{Var}\left(\sum_i w_i x_i\right) = \sum_i \mathrm{Var}(w_i x_i) = \sum_i \mathrm{Var}(x_i)\,\mathrm{Var}(w_i) = n\,\mathrm{Var}(x_i)\,\mathrm{Var}(w_i)$$

To achieve (1), we have

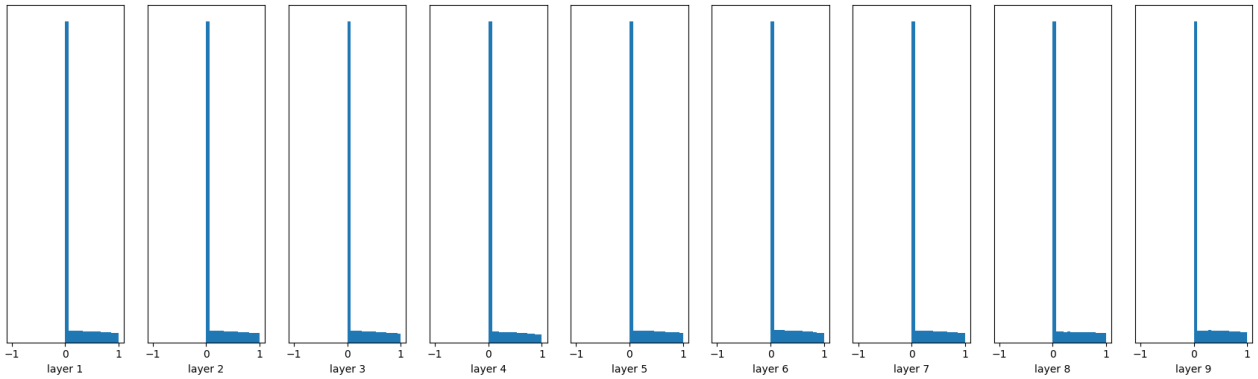
$$\mathrm{Var}(w_i) = \frac{1}{n} = \frac{1}{n_{\mathrm{in}}}$$

Regularization and optimization

- He initialization

np.random.randn(node_in,node_out) / np.sqrt(node_in/2)

For ReLU

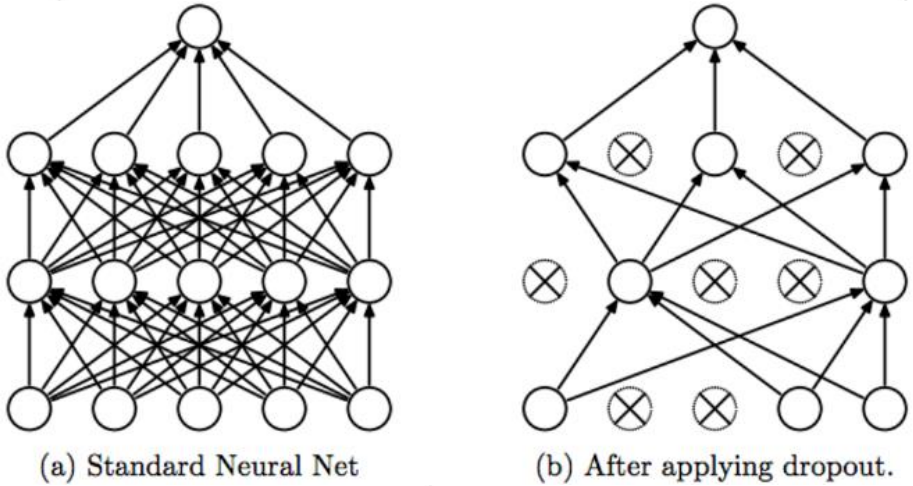


Assumptions (valid for each layer k)-

1. All elements in W^k share the same distribution and are independent of each other. Similarly for x^k and y^k.
2. each element of W^k and each element of x^k are independent of each other.
3. W^k and y^k have zero mean and are symmetrical around zero.
4. b^k is initialized to zero vector as we don't require any bias initially.

Regularization and optimization

- Dropout Regularization



(a) Standard Neural Net

(b) After applying dropout.

Training Phase

For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, p, of nodes (and corresponding activations).

Testing Phase

Use all activations, but reduce them by a factor p (to account for the missing activations during training).

[1] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

- Batch Normalization

why use batch normalization?

- reduce internal covariate shift
- network can use higher learning rate without vanishing or exploding gradients
- regularizing effect
- network becomes more robust to different initialization schemes and learning rates.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

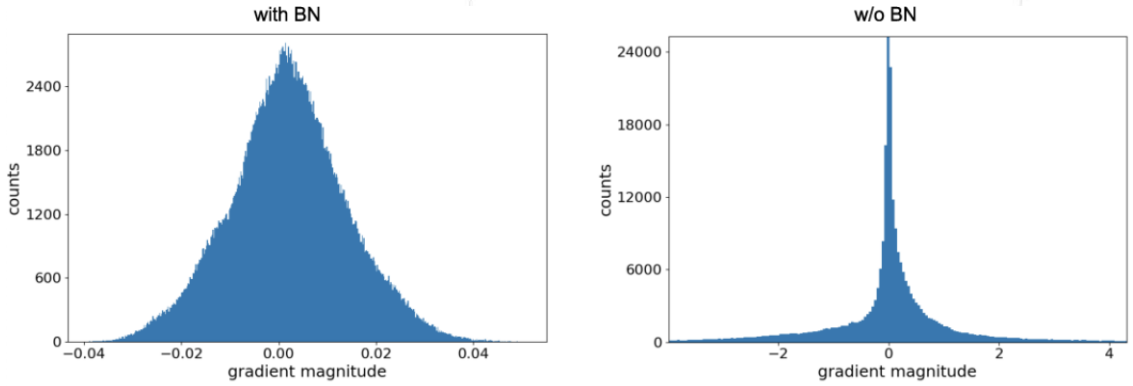
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.



Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

Ioffe S , Szegedy C . Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[J]. 2015.

3.2 Convolutional neural network and feature extraction

Dr. Jifeng Dai

Tuesday, March 8, 2022

商汤 sensetime

清華大學 Tsinghua University

Outline

Part 1	Introduction to CNN
Part 2	The Progress of CNN
Part 3	Analysis

商汤 sensetime

清華大學 Tsinghua University

Highlights

Learn the basic operators in CNN

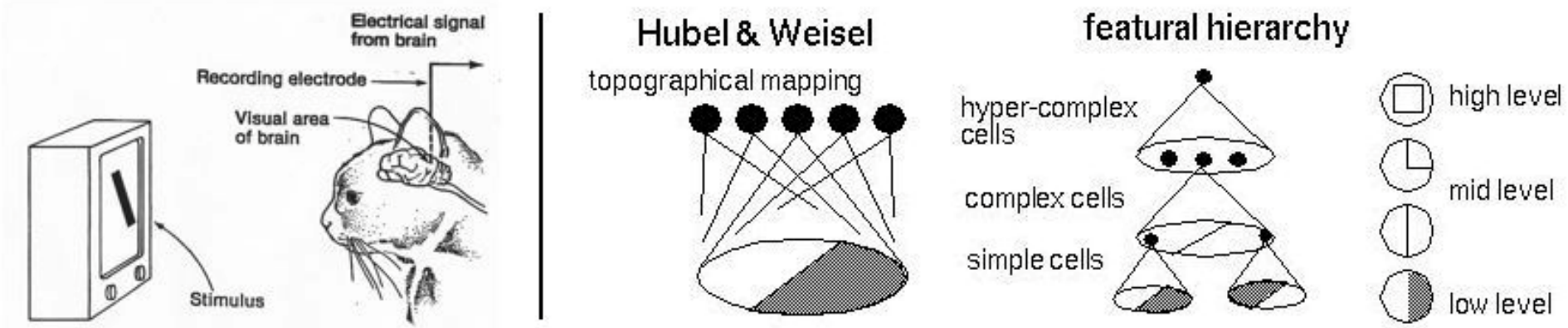
Learn the development process of convolutional neural network

Learn the training process of CNN

Understand the methods of neural network feature visualization

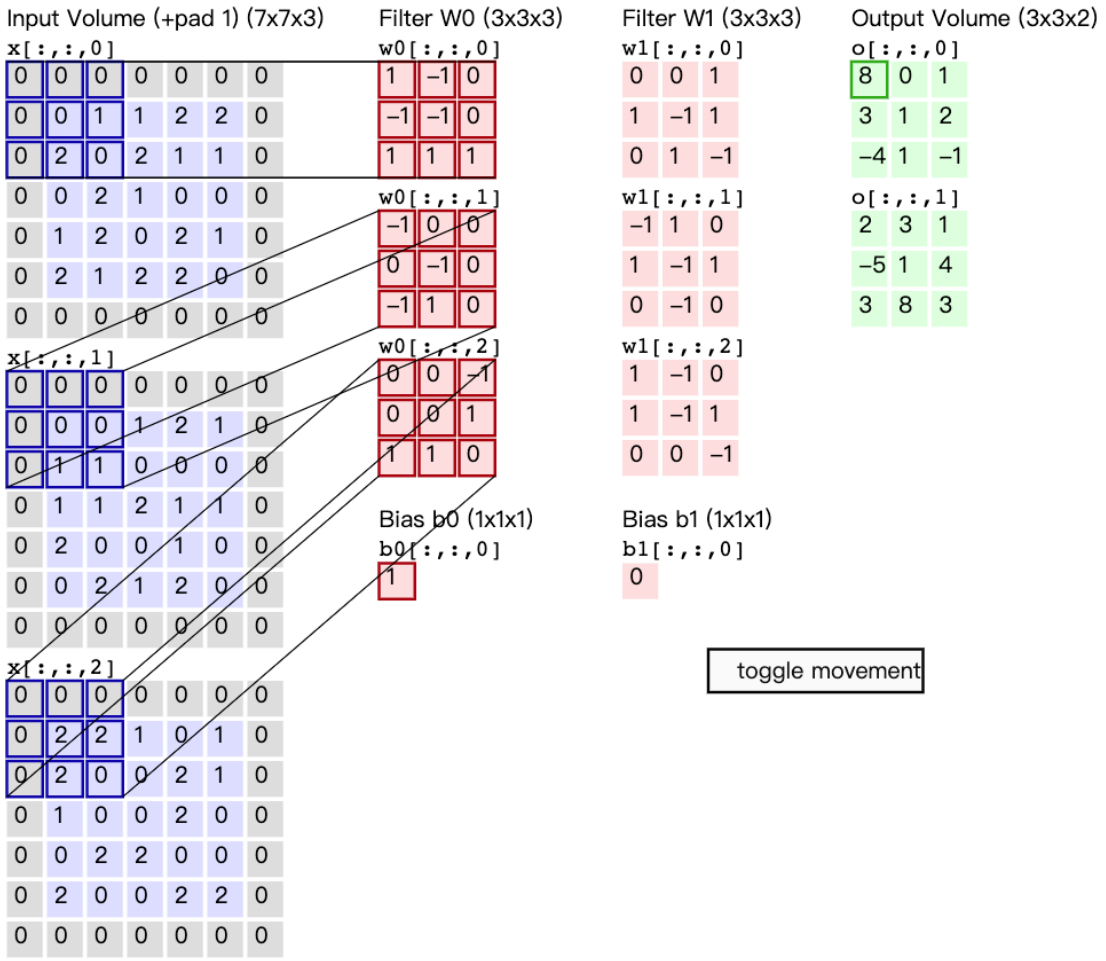
Understand the problems of large-batch training

- Inspired by nature



Hubel D H , Wiesel T N . Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. Journal of Physiology, 1962.

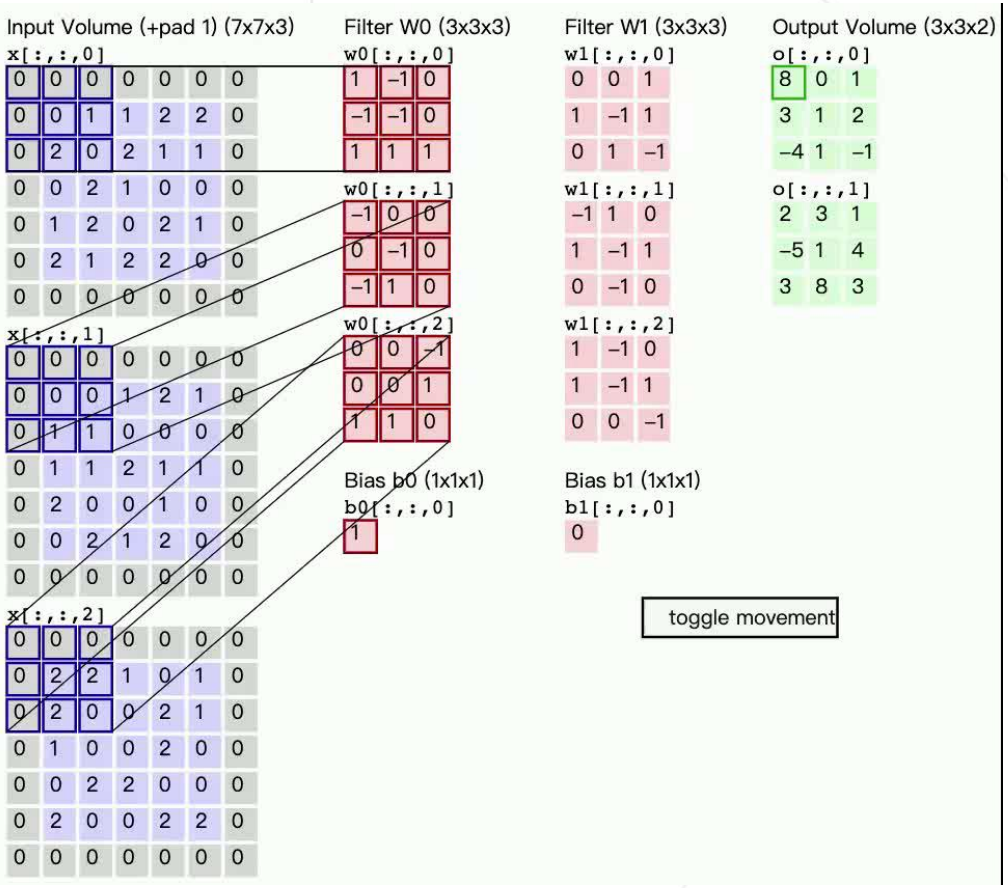
- Convolution



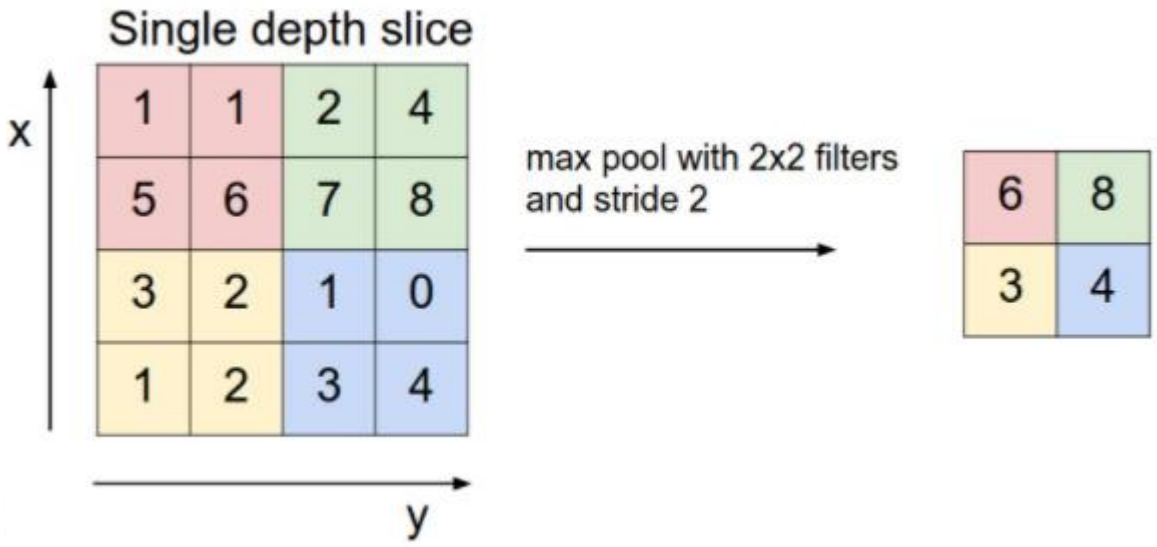
https://cs231n.github.io/convolutional-networks/

Introduction to CNN

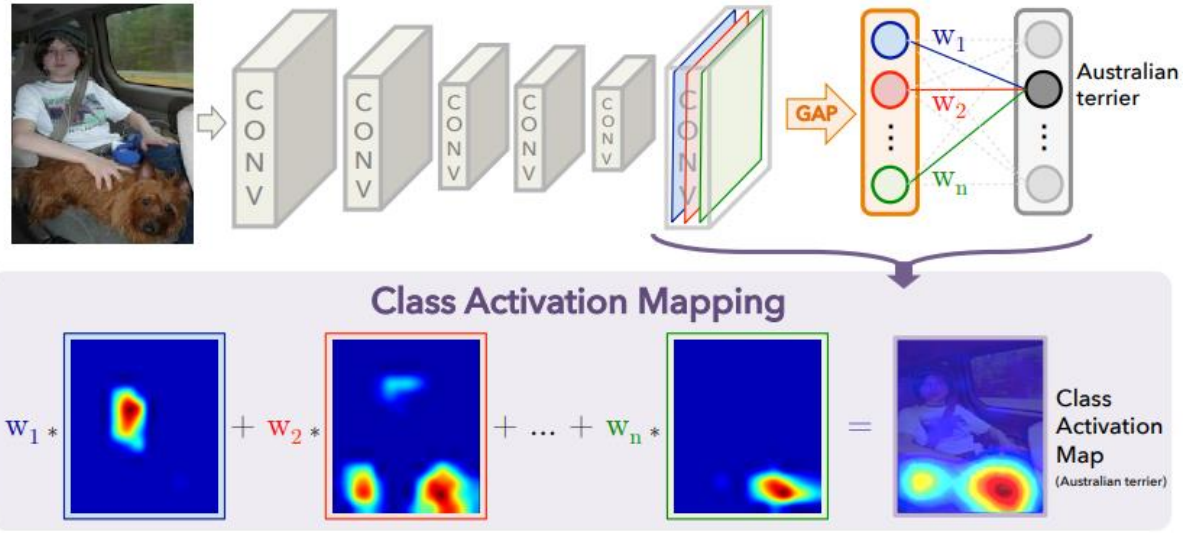
- Features of convolutional layer
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations
- kernel size
  - 3x3, 5x5, ...
- padding
  - zero padding
- stride
  - =2: input size: 5x5 -> output size: 3x3
  - $output\_size = \frac{input\_size + 2 * padding - kernel\_size}{stride} + 1$

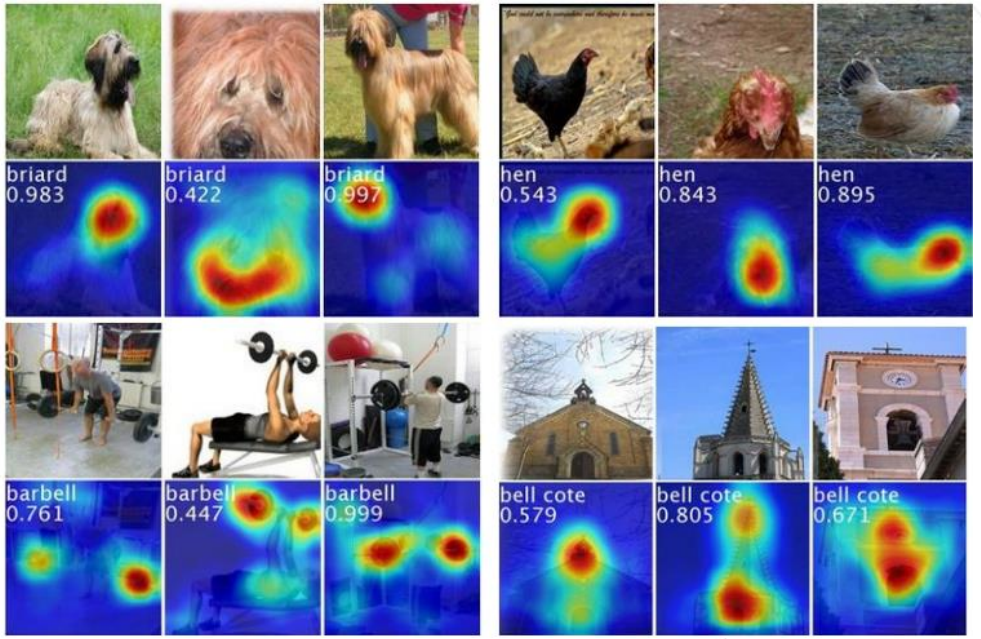


- Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

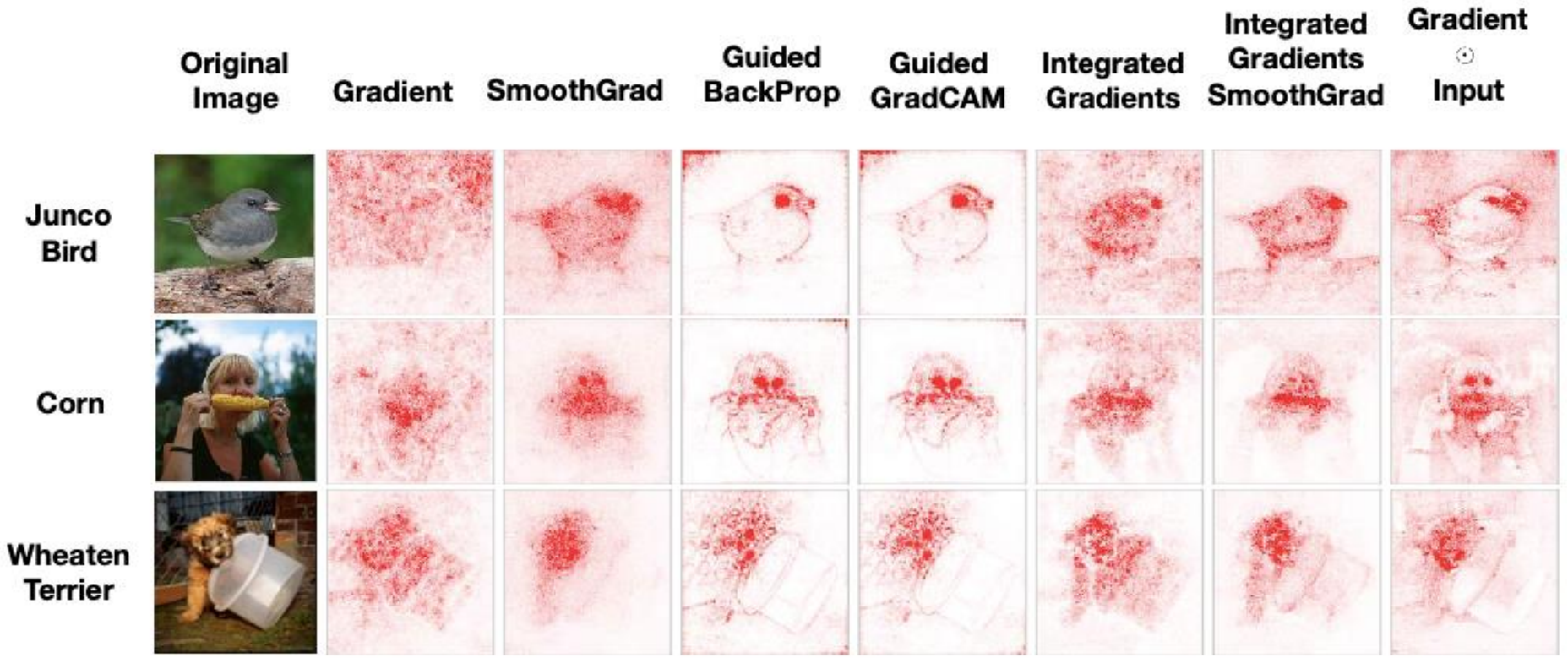
- CAM(class activation map)





Zhou B, Khosla A, Lapedriza A, et al. Learning deep features for discriminative localization[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 2921-2929.

- The interpretability of CNN
  - Saliency map can tell us where the CNN focus in an image.



Adebayo J, Gilmer J, et al. Sanity Checks for Saliency Maps. NIPS, 2018.

商汤 sensetime

清華大學 Tsinghua University

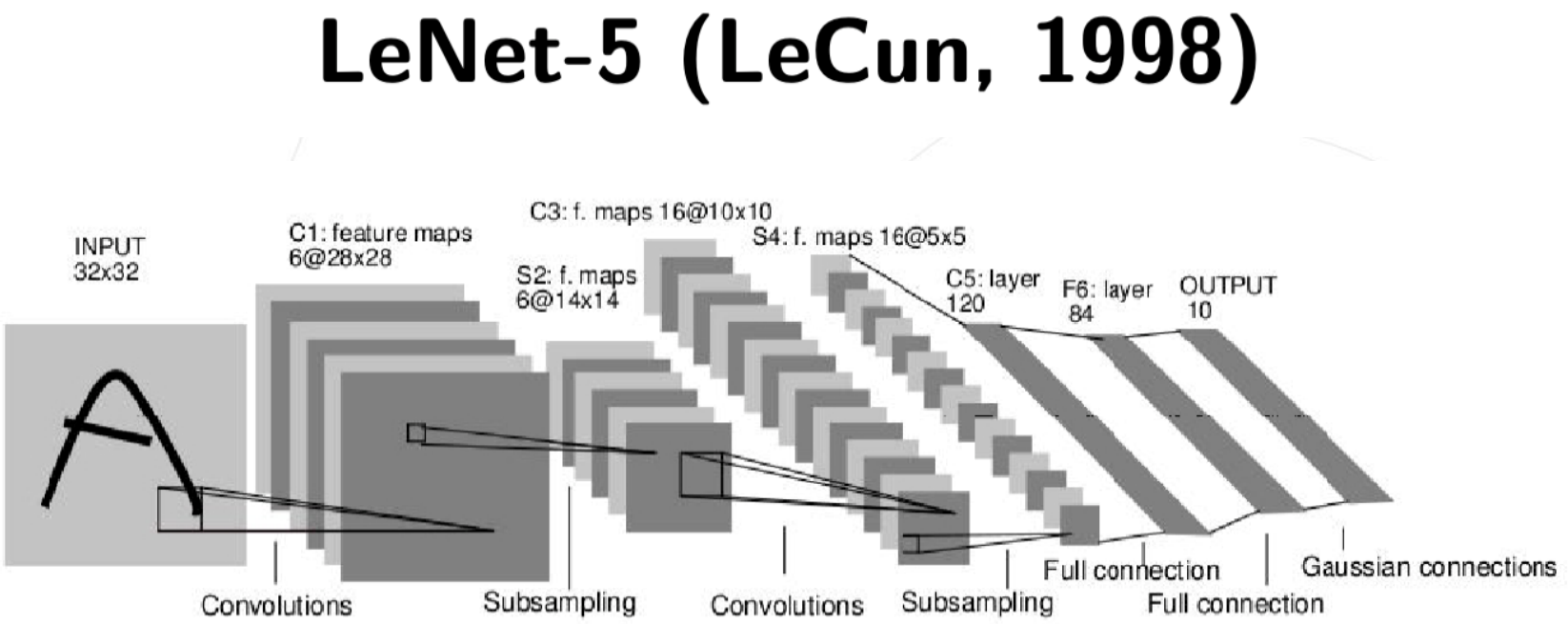
Outline

Part 1	Introduction to CNN
Part 2	The Progress of CNN Architecture
Part 3	Analysis

The Progress of CNN

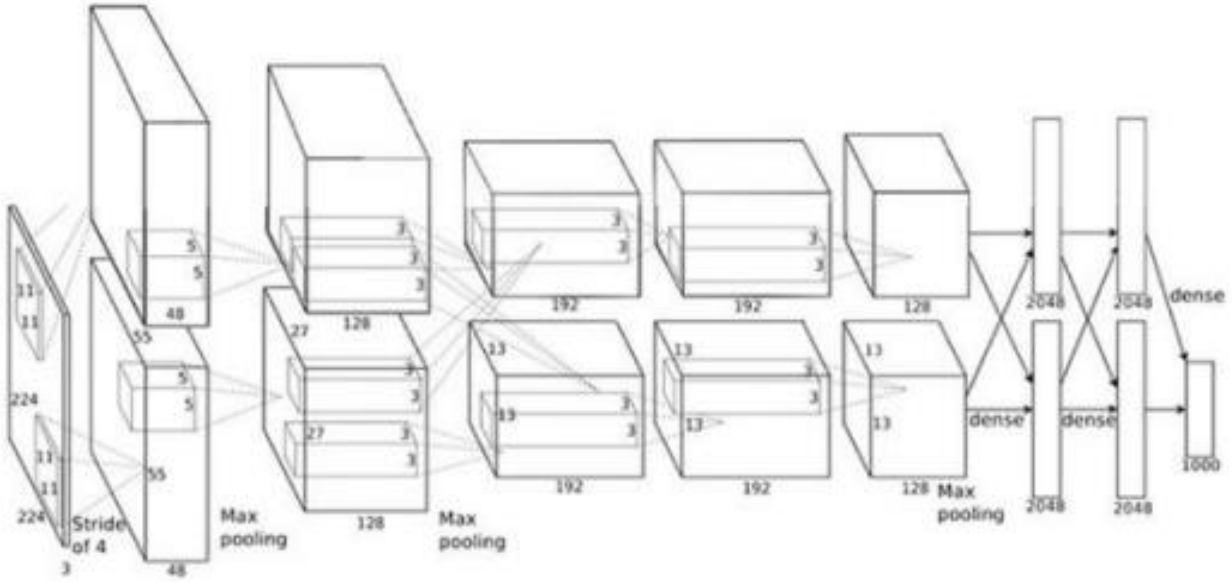
- **Progress**

- Practical CNN for Document Recognition



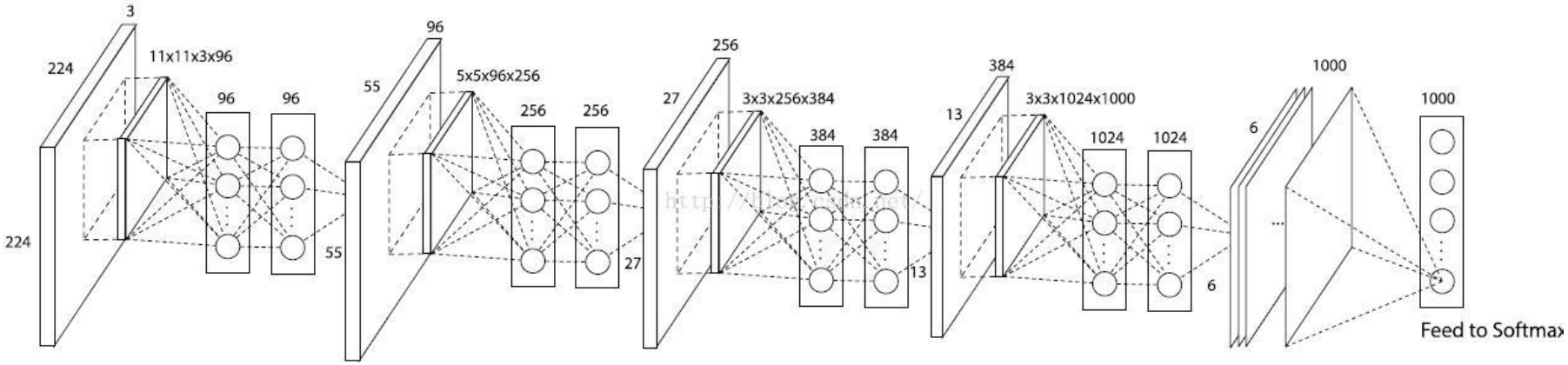
Lecun Y , Bottou L . Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998.

- **AlexNet**
  - Deeper
    - 8 layers
  - ReLU
    - alleviate the problem of saturation
  - Dropout
    - prevent over-fitting



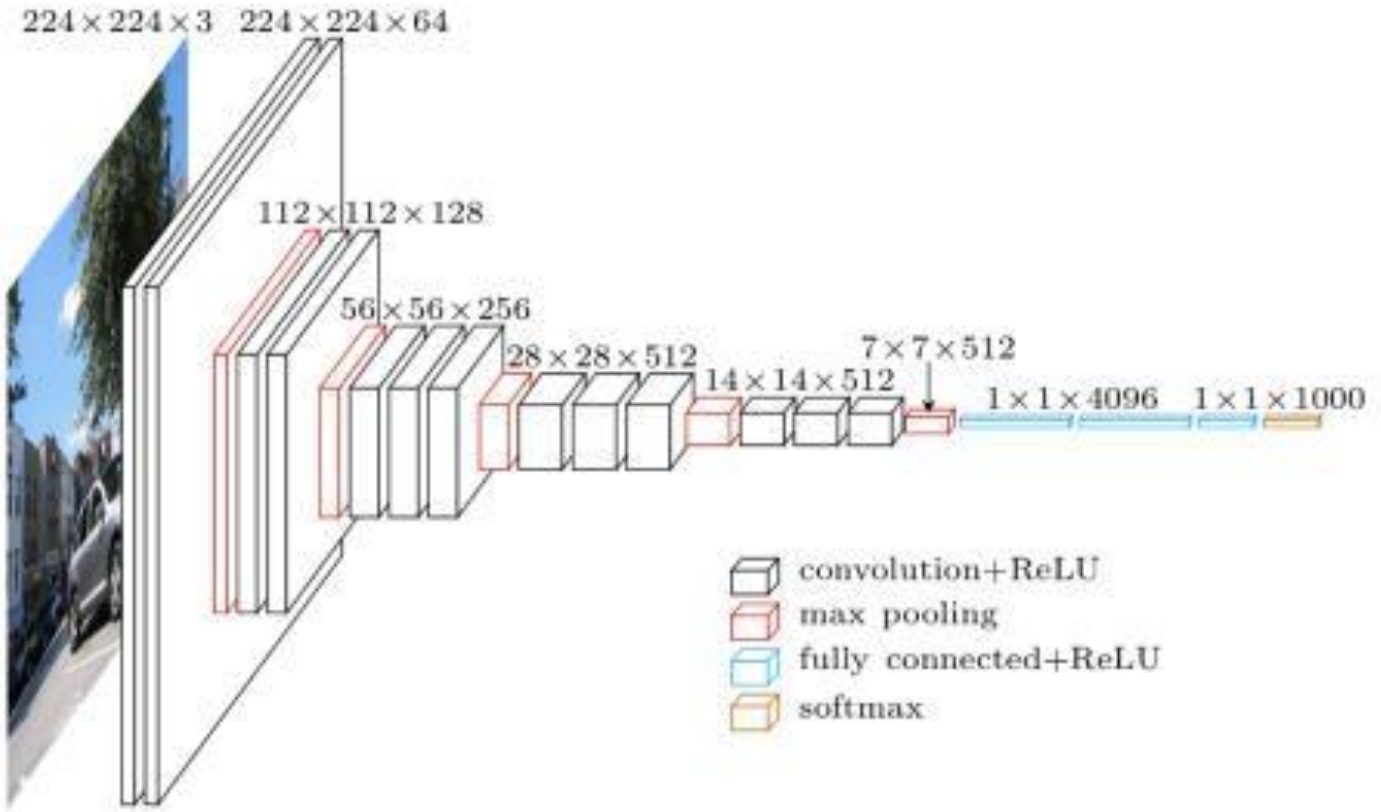
Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. NIPS, 2012.

- **NIN**
  - 1x1 conv
    - Improve the flexibility and capacity of CNN
  - Global average pooling (for image classification)
    - Efficient extraction of image feature



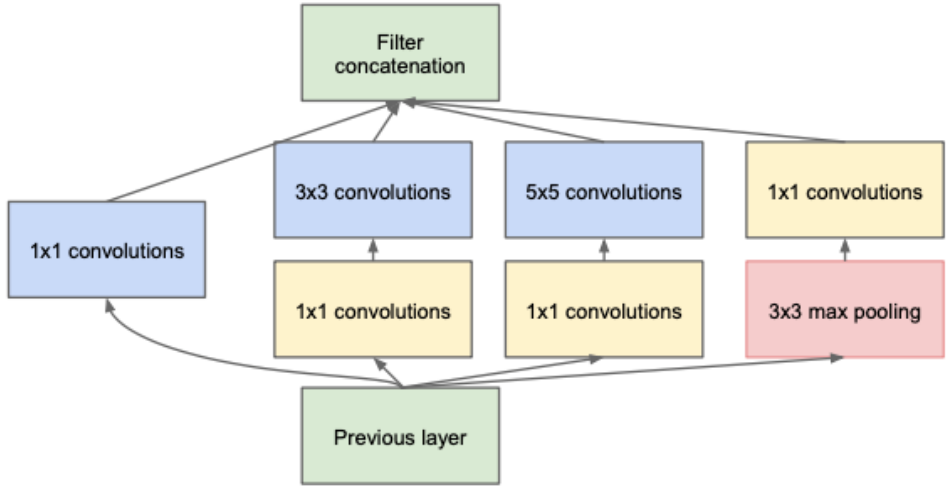
Lin M, Chen Q, Yan S. Network In Network. ICLR, 2014.

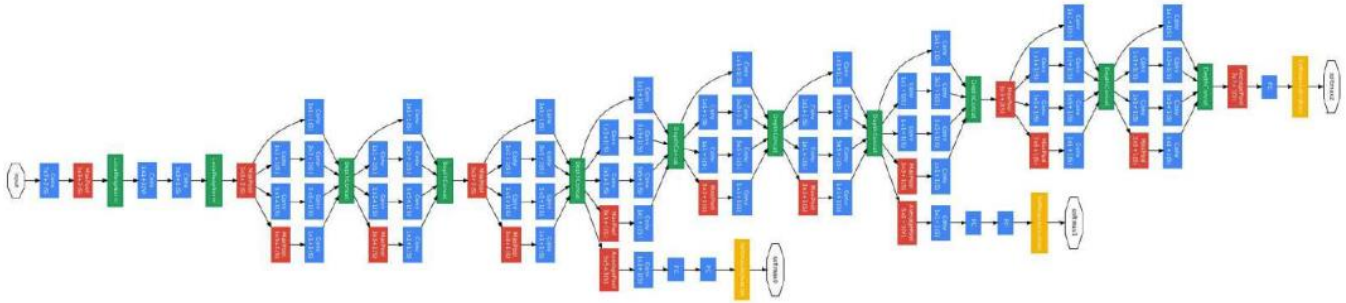
- **VGG**
  - Deeper
    - 16 layers typically
  - Small kernel
    - less parameters
    - more non-linearity
    - larger receptive-field



Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

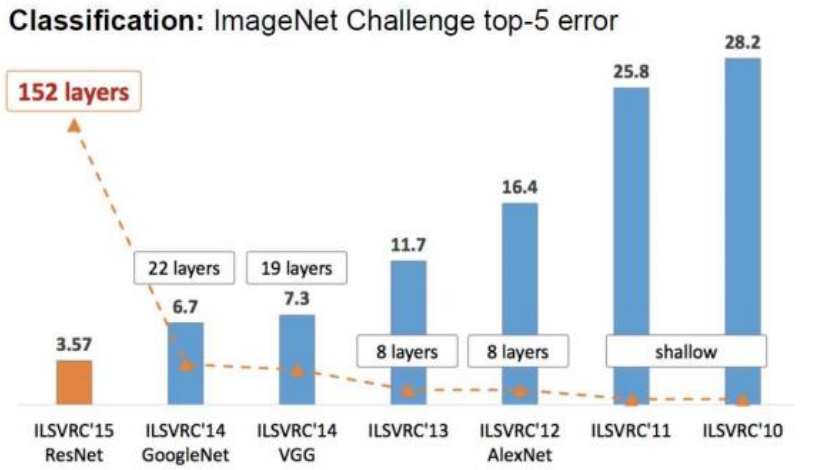
- **GoogLeNet**
  - Deeper
    - 22 layers
  - Inception Module
    - Increase in-block diversity
  - Deep supervision
    - alleviate gradient vanishment and explosion

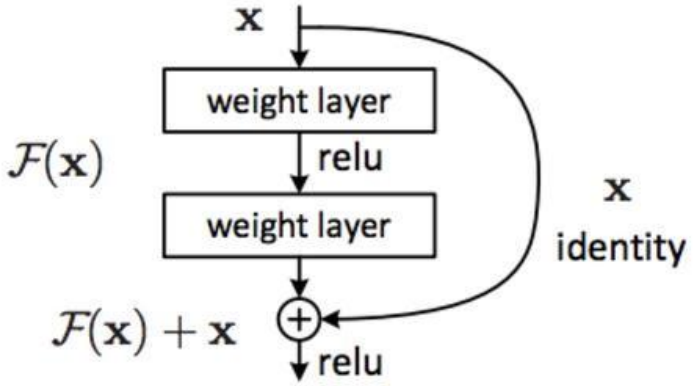




Szegedy C, Liu Q, Jia Y, et al. Going Deeper with Convolutions. CVPR, 2015.

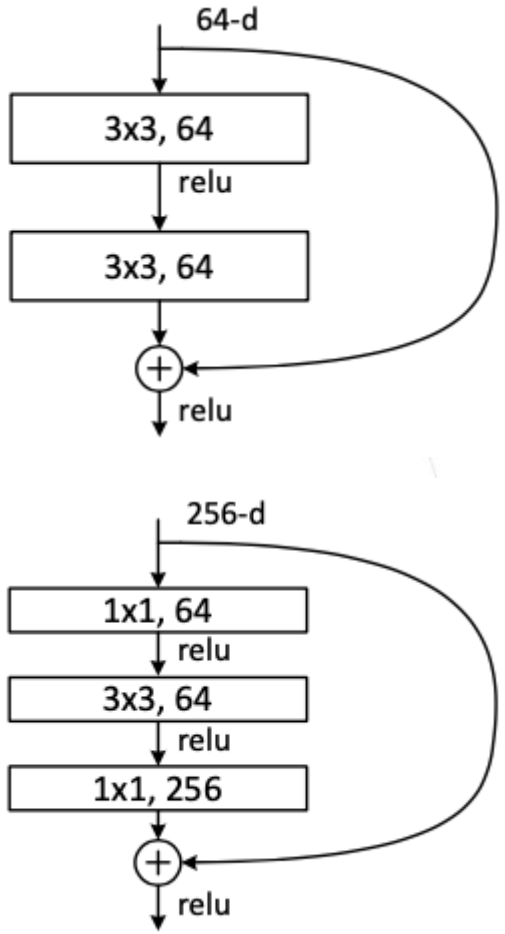
- **ResNet**
  - Much Deeper
    - up to ~1000 layers
  - Residual learning
    - Difficulty of learning identity mapping
    - Similar behavior like an ensemble of many shallow networks





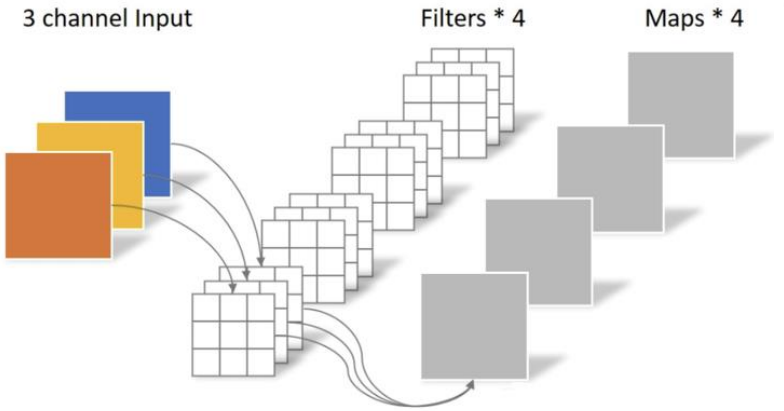
He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition. CVPR, 2016.

- **ResNet**
  - BasicBlock
    - Simple
    - Efficient utilization of GPU memory
  - Bottleneck
    - More channels in residual path to carry more information
    - Balance between 1x1 conv (channel) and 3x3 conv (spatial)

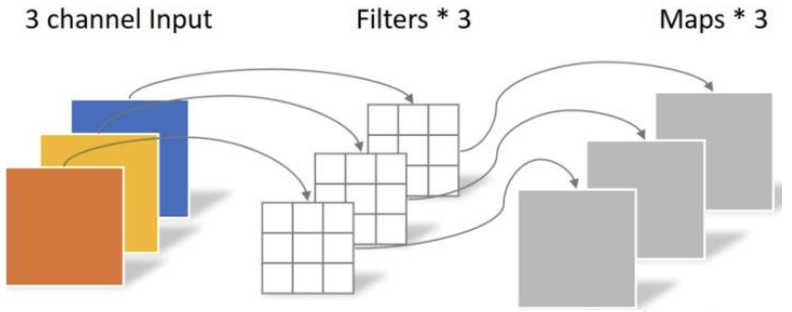


He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition. CVPR, 2016.

- **Light CNN**
  - MobileNet V1
  - Depthwise Separable Convolution
    - Significant reduction of computation
    - Foundamental component of light CNN
    - 3x3 DS conv + 1x1 conv ~ 3x3 conv



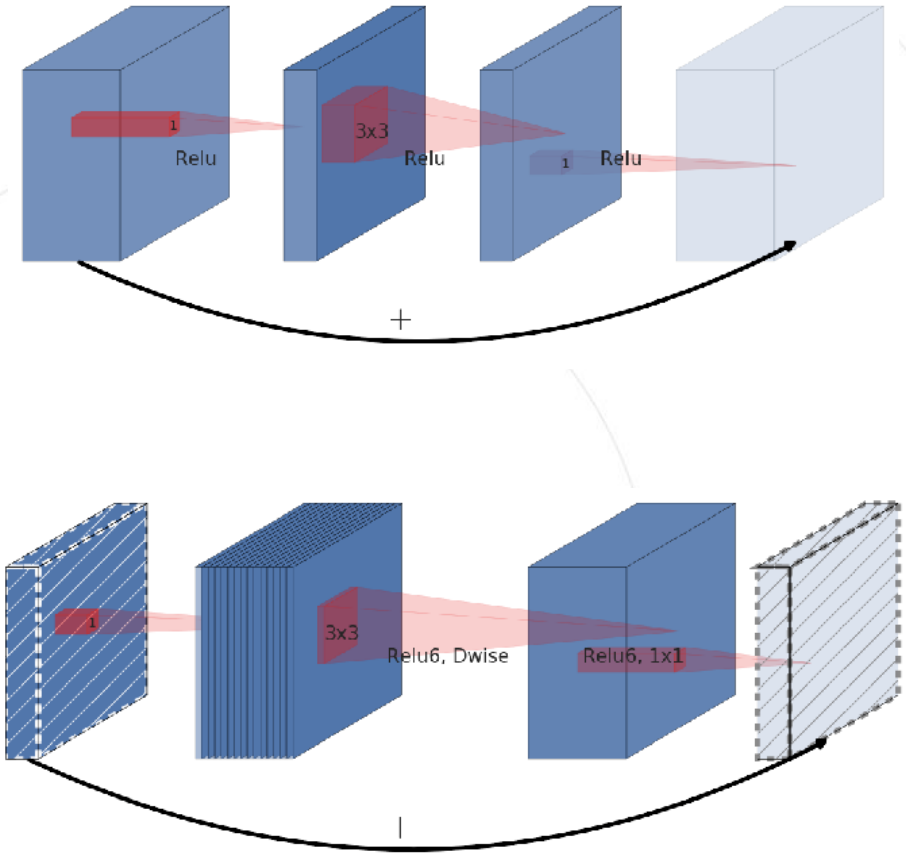
$$d_{out} \times d_{in} \times k \times k$$



$$d_{in} \times 1 \times k \times k$$

Howard A, Zhu M, Chen B, et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. CVPR, 2017.

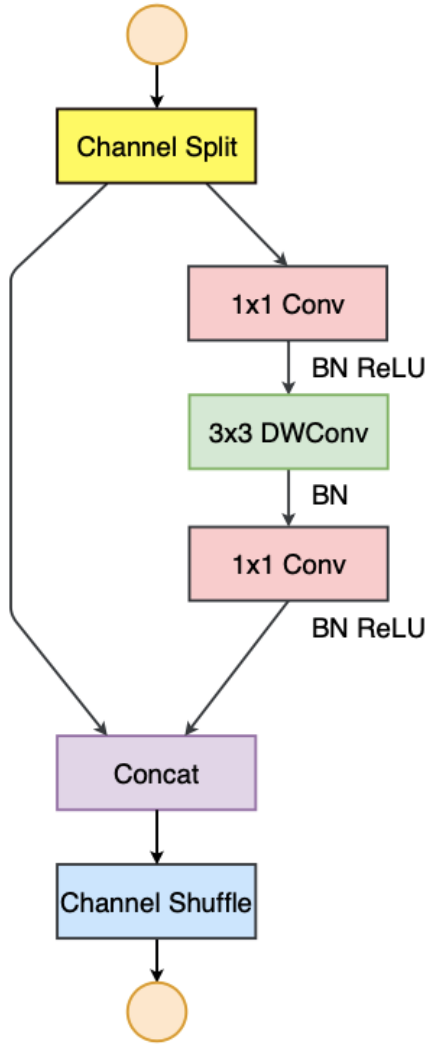
- **Light CNN**
  - MobileNet V2
  - Inverted Residual Block
    - Balance of the computation between 3x3 DS conv and 1x1 conv
  - Architecture design



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

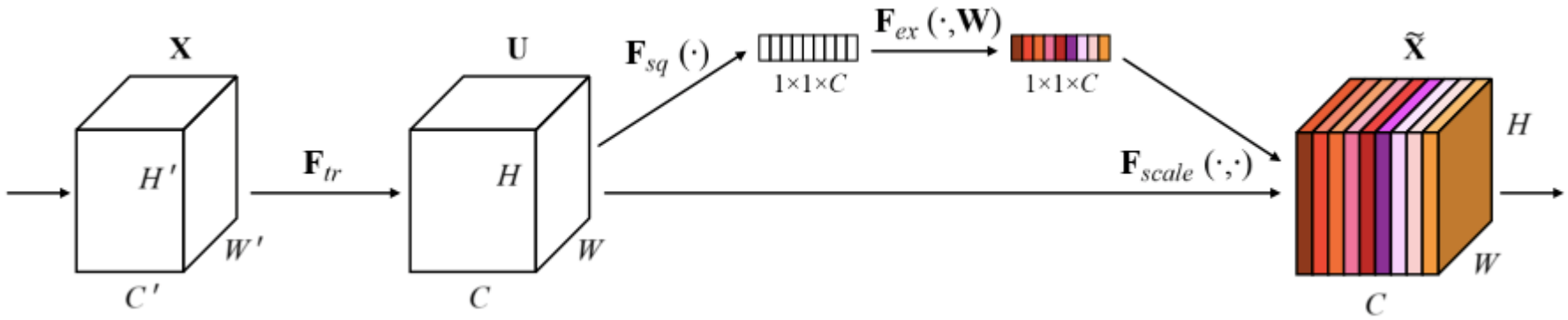
Sandler M, Howard A, Zhu M, et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. CVPR, 2018.

- **Light CNN**
  - Shufflenet V2
  - Practical Guidelines for Efficient Network Design
    - Equal channel width minimizes MAC（memory access cost)
    - Excessive group convolution increases MAC
    - Network fragmentation reduces degree of parallelism
    - Element-wise operations are non-negligible



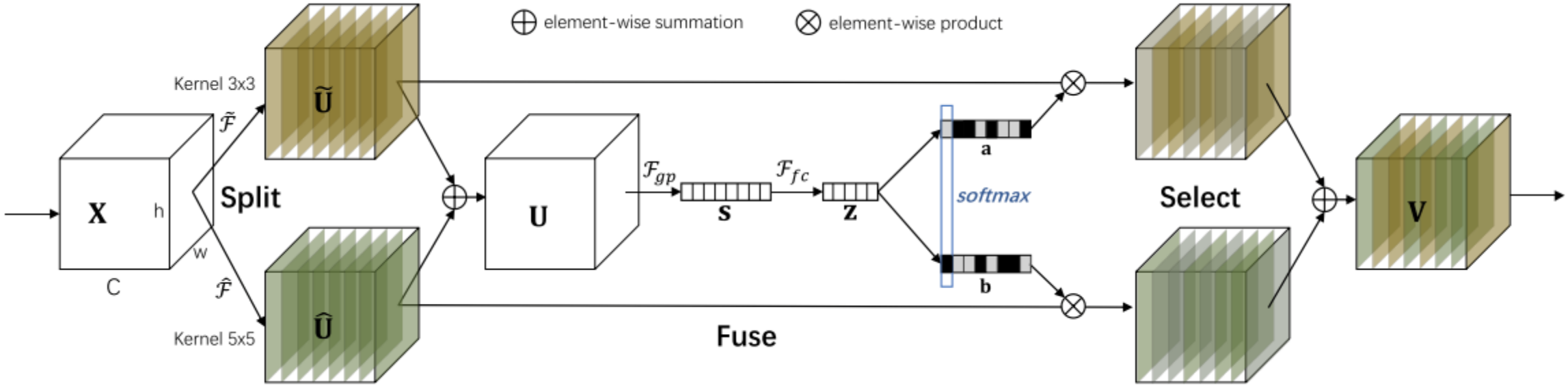
Ma N, Zhang X, Zheng H, et al. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. ECCV, 2018.

- **Data Dependent Network**
  - SENet
    - Reweight channels according to the input, or Channel Attention
    - Capture global context information via GAP



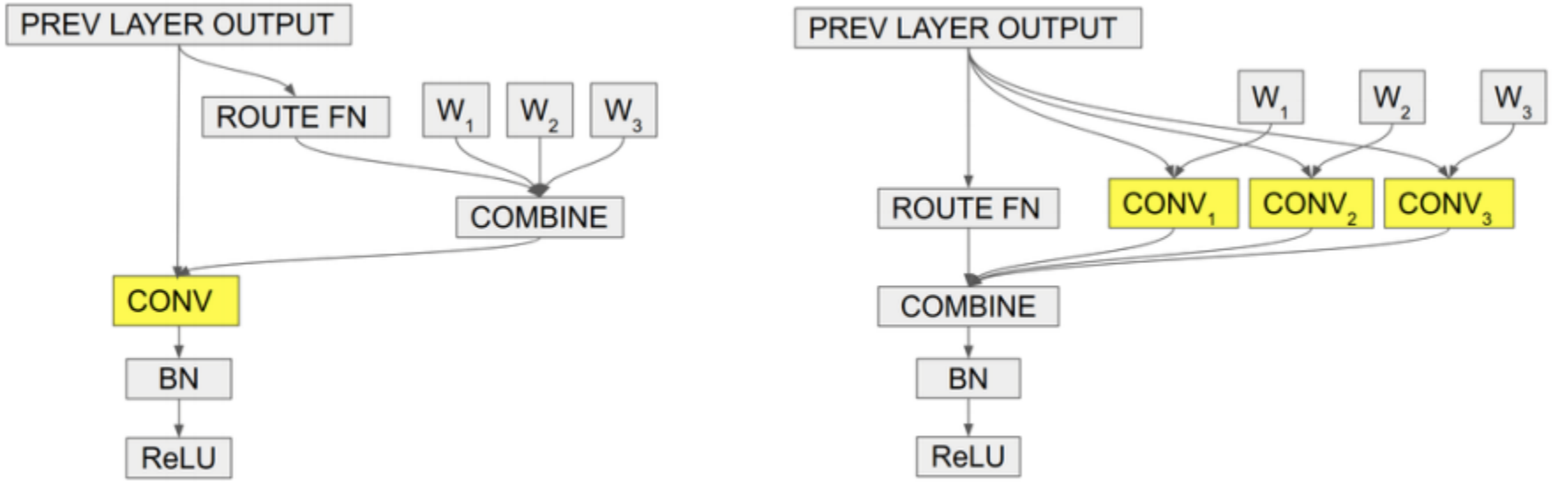
Hu J, Shen L, Sun G. Squeeze-and-Excitation Networks. CVPR, 2018.

- **Data Dependent Network**
  - SKNet
    - Switch between 5x5 conv and 3x3 conv according to the input



Li X, Wang W, Hu X, et al. Selective Kernel Networks. CVPR, 2019.

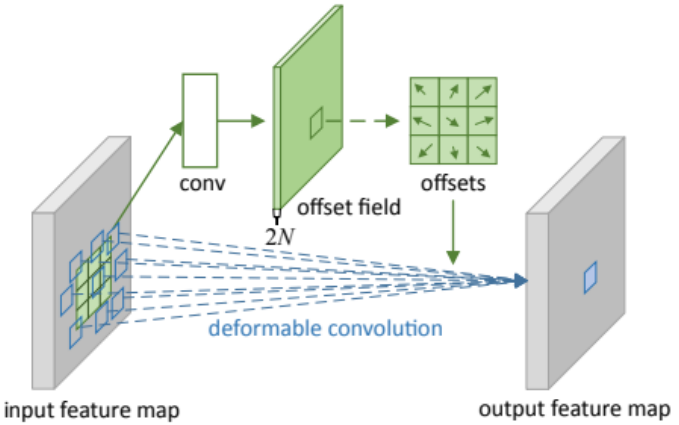
- **Data Dependent Network**
  - CondConv
    - Generate CNN weight according to the input

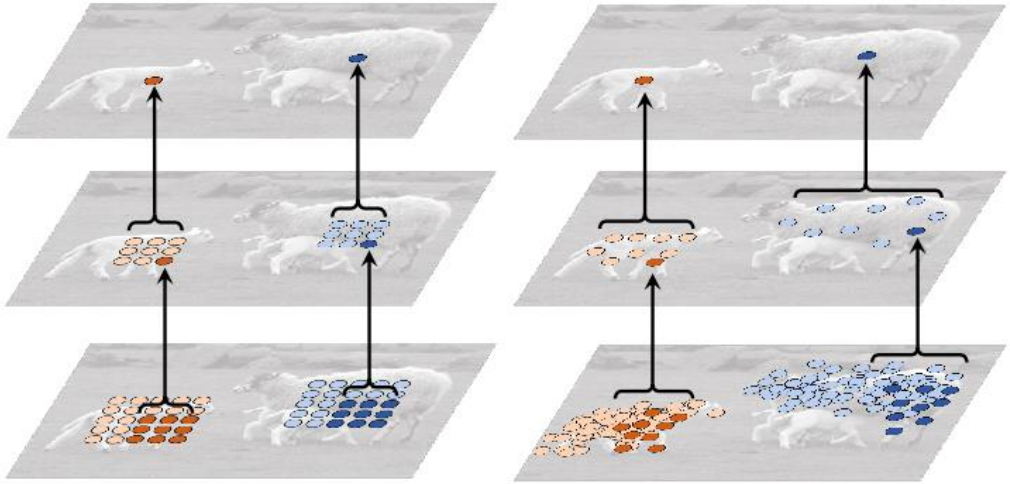


(a) CondConv: $(\alpha_1 W_1 + \ldots + \alpha_n W_n) * x$ (b) Mixture of Experts: $\alpha_1 (W_1 * x) + \ldots + \alpha_n (W_n * x)$

Yang B, Bender G, Le Q, et al. CondConv: Conditionally Parameterized Convolutions for Efficient Inference. NIPS, 2019.

- **Data Dependent Network**
  - Deformable Convolution
    - Change the pixel-to-weight relation according to the input
    - Enlarge the receptive field
    - Helpful for the resistance of object deformation





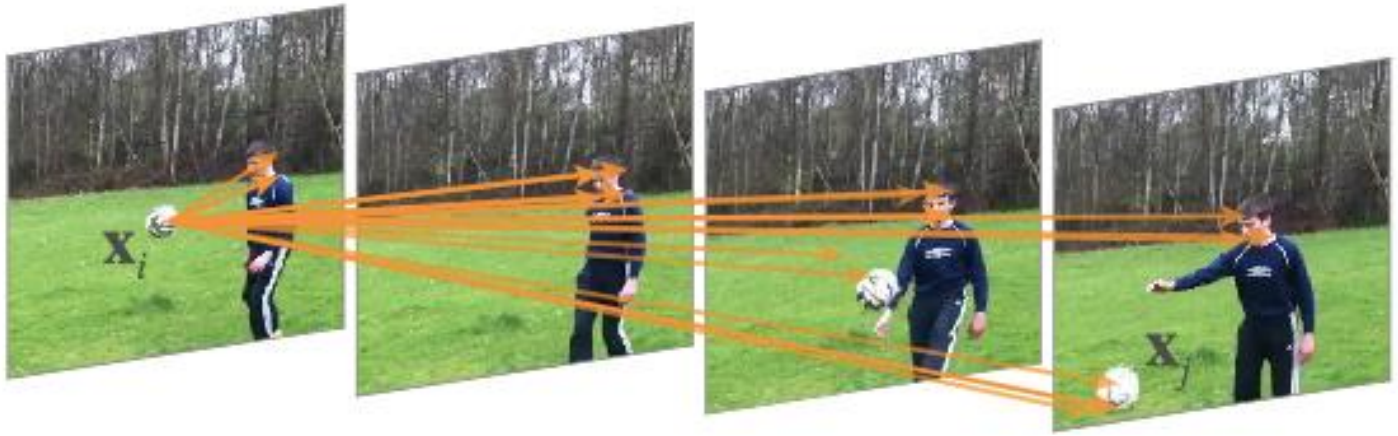
(a) standard convolution (b) deformable convolution

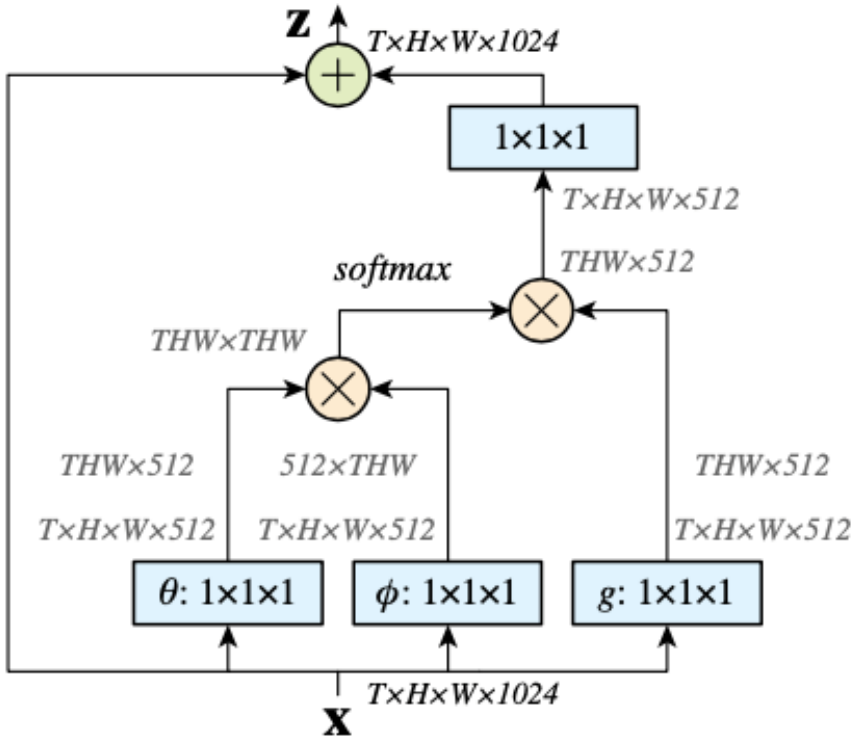
Dai J, Qi H, Xiong Y, et al. Deformable Convolutional Networks. ICCV, 2017.

- **Data Dependent Network**
  - Deformable Convolution

Dai J, Qi H, Xiong Y, et al. Deformable Convolutional Networks. ICCV, 2017.

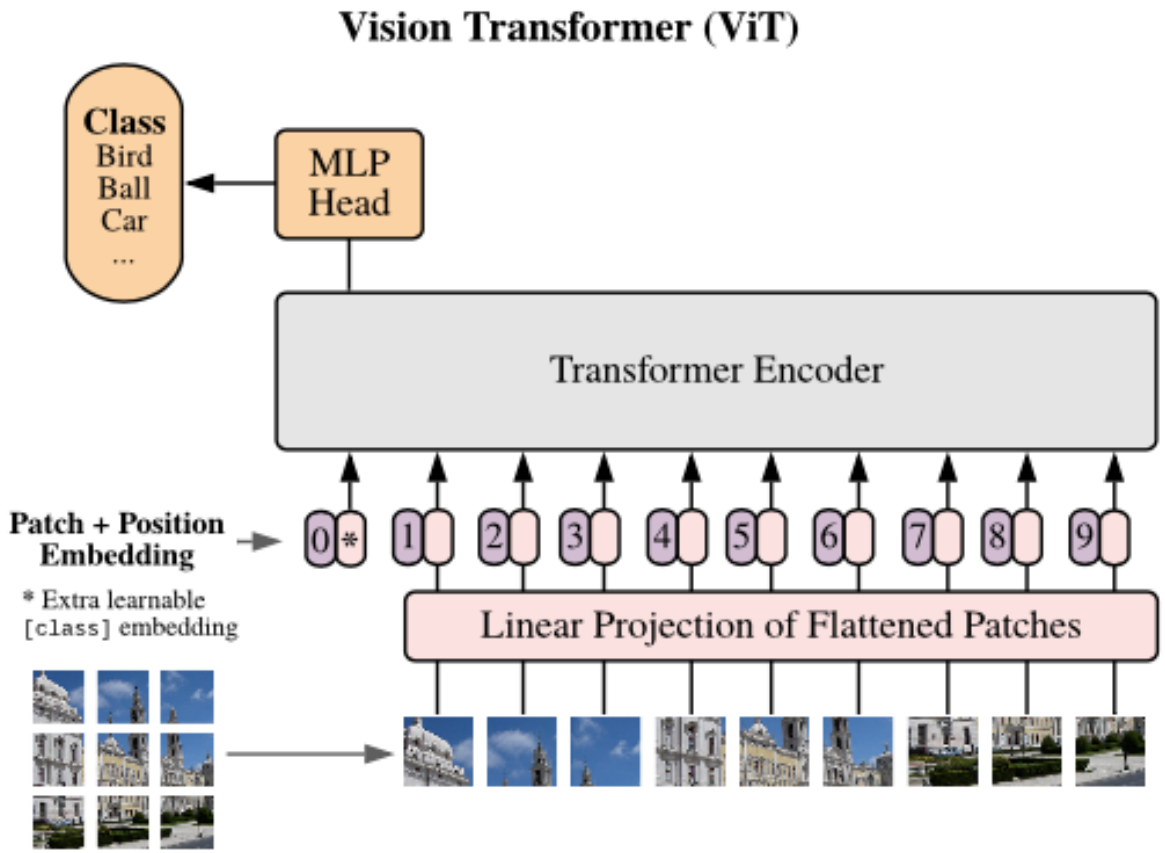
- **Attention Mechanism in CNN**
  - Non-local Neural Network
    - Capture pixel-to-pixel dependency information





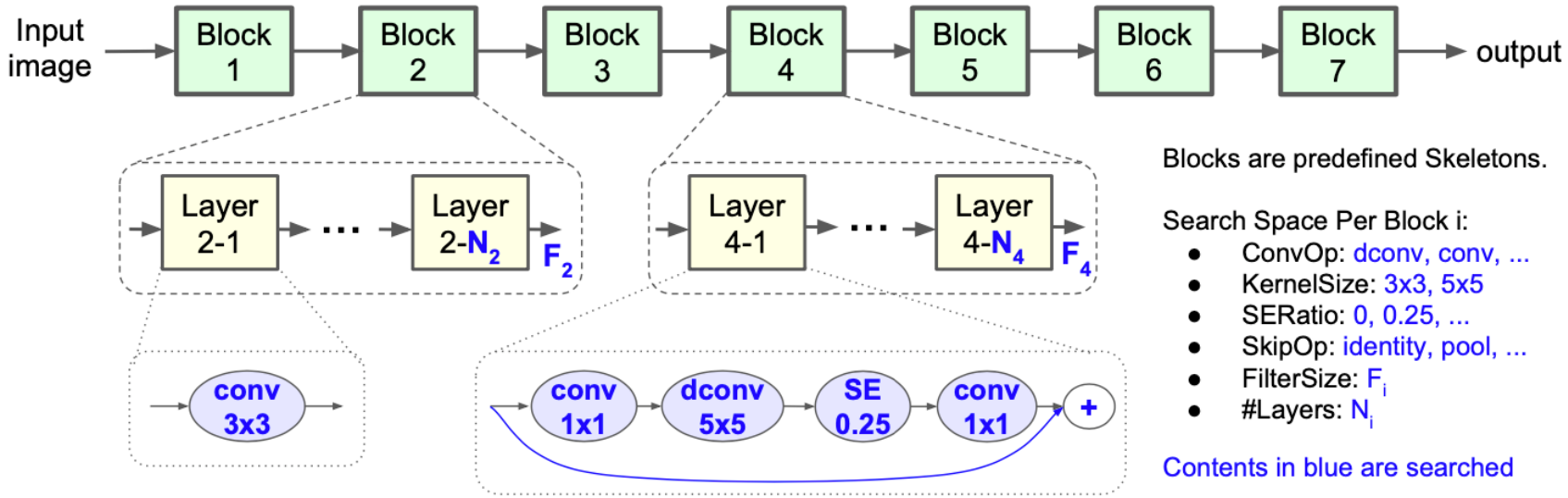
Wang X, Girshick R, Gupta A, et al. Non-local Neural Networks. CVPR, 2018.

- **Attention Mechanism in CNN**
  - Vision Transformer
    - Applying a standard Transformer directly to images
    - See the image as a sequence of patch
    - Require pretrain on large-scale dataset



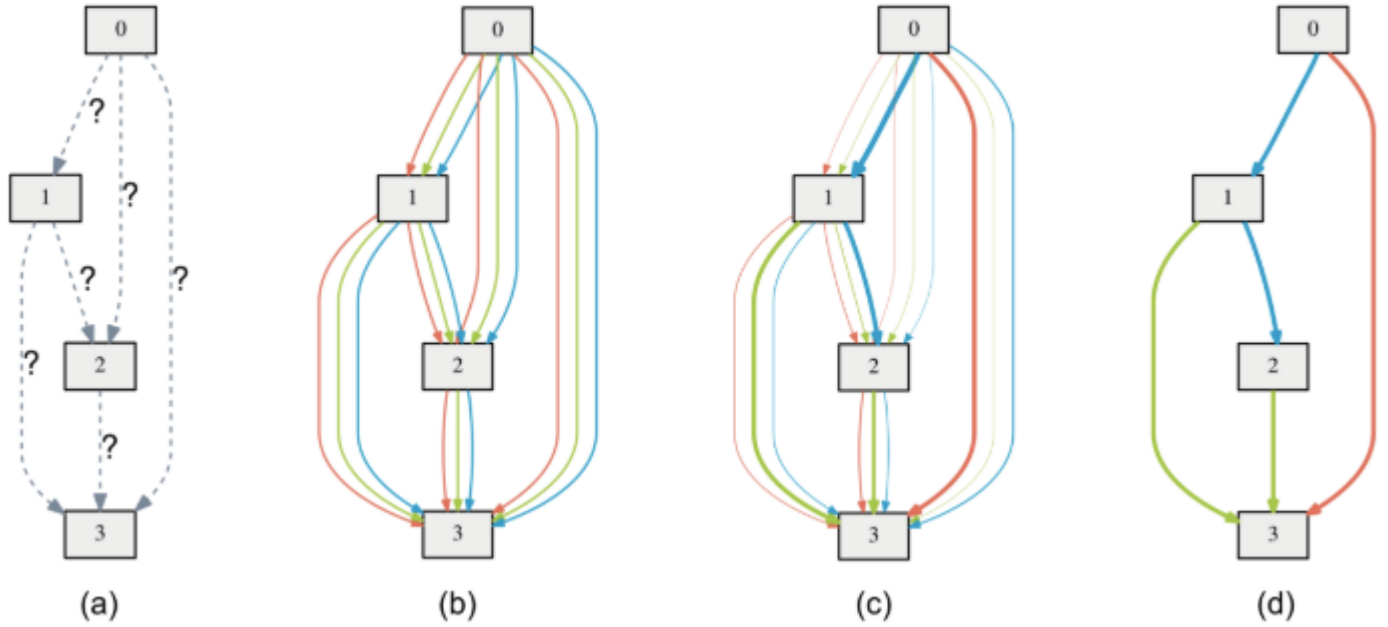
Dosovitskiy A, Beyer L, Kolesnikov A, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ICLR, 2021.

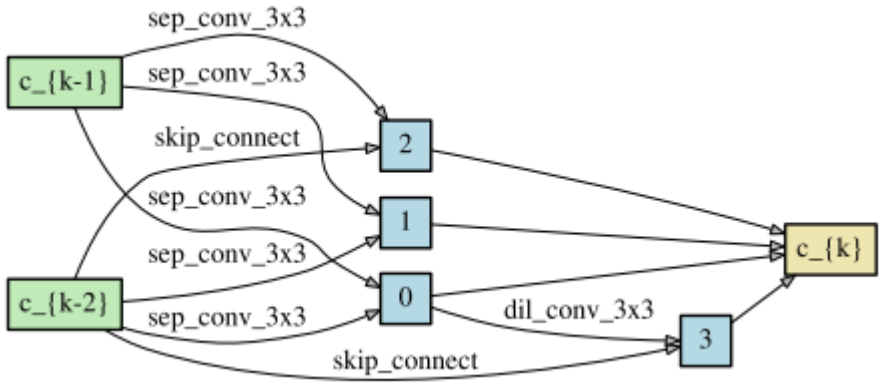
- **Network Architecture Search**
  - MNasNet
    - Search based on Reinforcement Learning
    - Show the superiority of NAS



Tan M, Chen B, Pang R, et al. MnasNet: Platform-Aware Neural Architecture Search for Mobile. CVPR 2019.

- **Network Architecture Search**
  - DARTS
    - General search space
    - Share weight between sub-networks
    - Differentiable search process





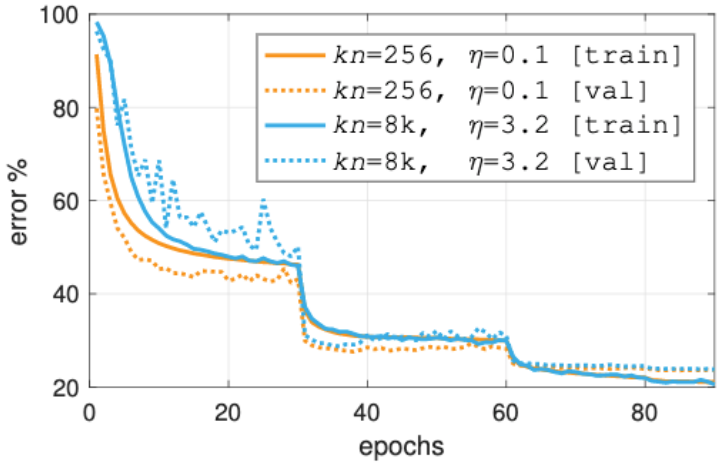
Liu H, Simonyan K, Yang Y. DARTS: Differentiable Architecture Search. ICLR, 2019.

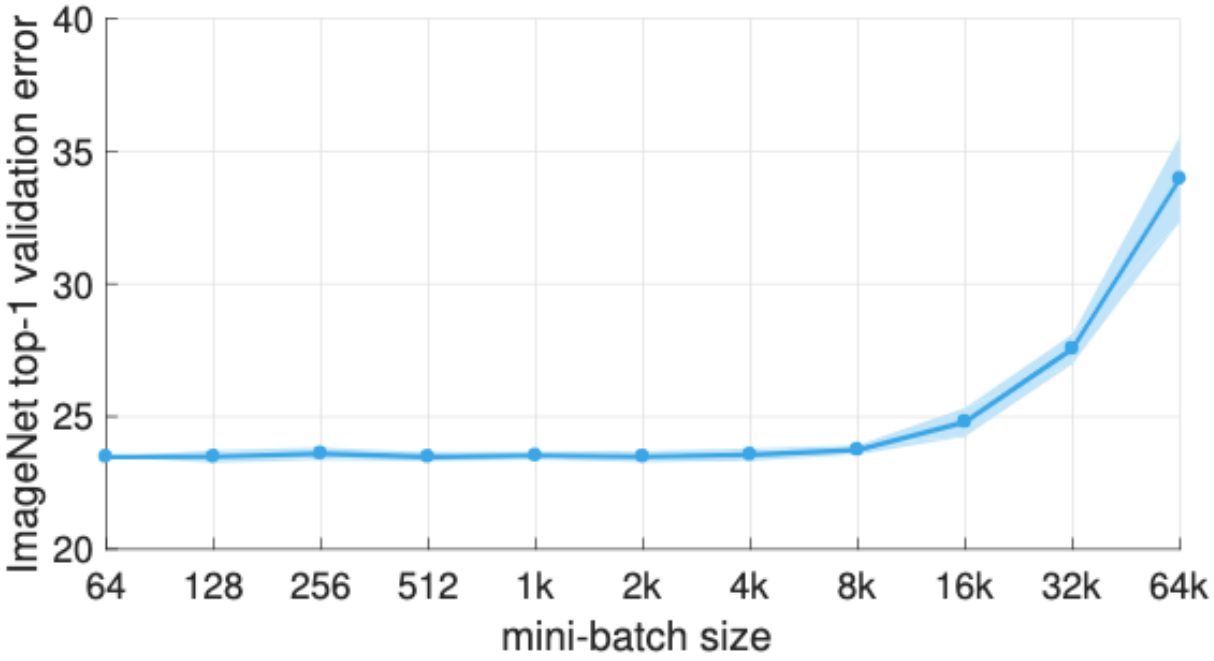
Outline

Part 1	Introduction to CNN
Part 2	The Progress of CNN
Part 3	Analysis

Analysis

- Large batch training for CNN
  - Learning rate linear scale up
  - Learning rate warmup

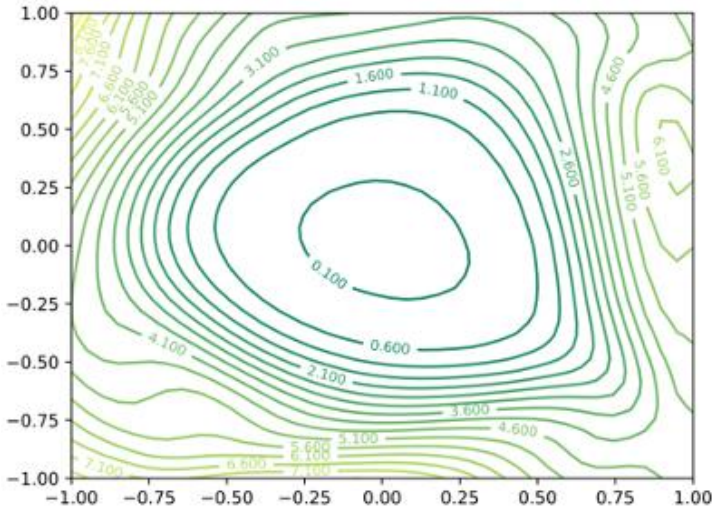




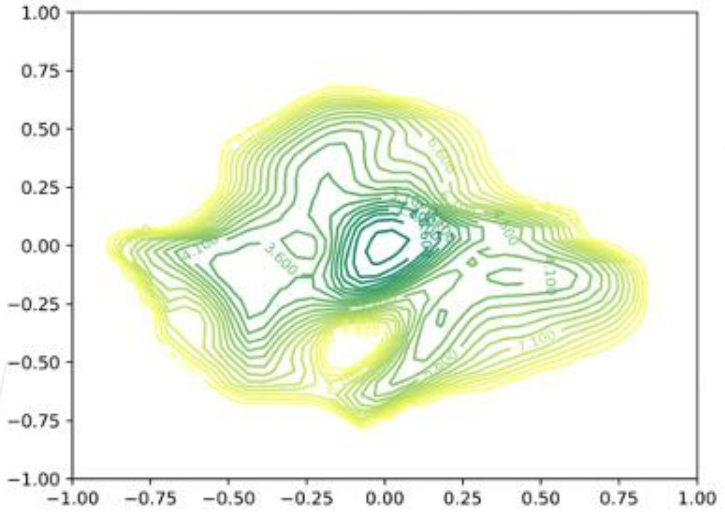
Priya Goyal, Piotr Dollar, Ross Girshick, Pieter Noordhuis. Accurate, Large Minibatch SGD:Training ImageNet in 1 Hour

Analysis

- Optimization View
  - Large batch training tends to converge to a sharper minima



Loss landscape with batch size 128 on Cifar10



Loss landscape with batch size 50000 on Cifar10

Nitish Shirish Keskar, Dheevatsa Mudigere et al. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima
Hao Li, Zheng Xu, Gavin Taylor et al. Visualizing the Loss Landscape of Neural Nets