CNN - Convolutional neural networks

(Feature map, Convolution, Pooling, Padding)

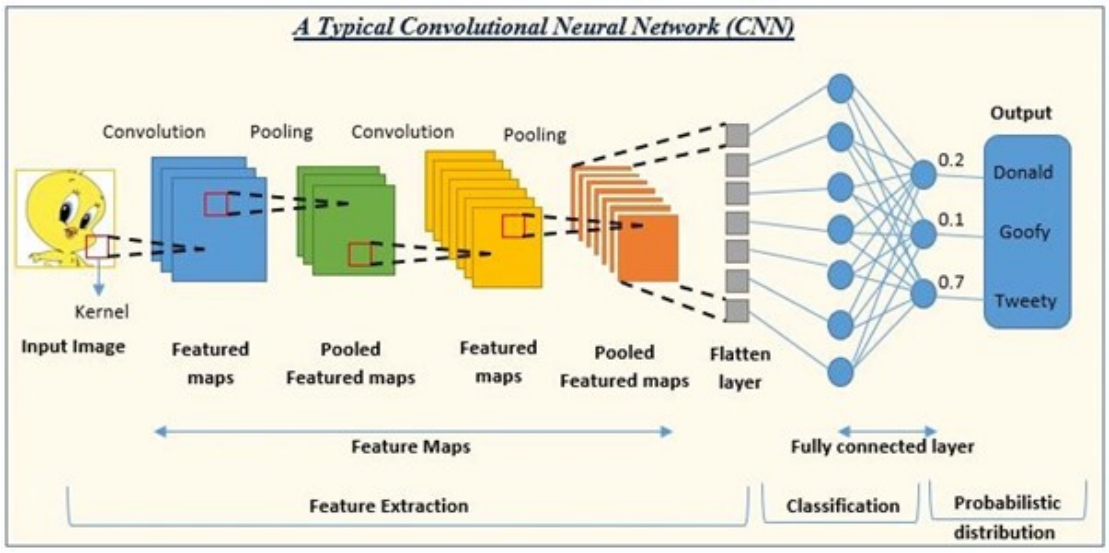
Overview

Common usage: Image classification, object detection (classification + localization), voice detection (related to signal)

What is the purpose of using CNN?

Extract features from input data (feature map)

Basic CNN architecture:



Forward propagation:
Input layer → Convolution layer → Activation function → Pooling layer → Fully connected layer → Output layer
(输入层 → 卷积层 → 激活函数 → 池化层 → 全连接层 → 输出层)

1. Input layer

Input: Image/video/signal

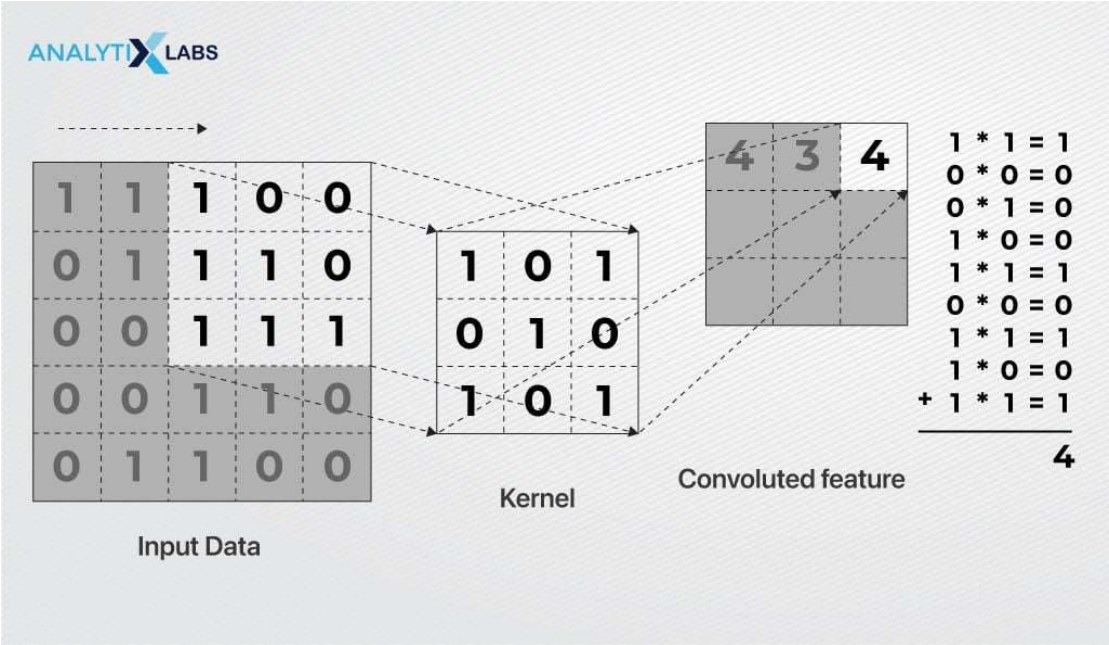
For image, input dimension = $[H_{in}, W_{in}, C_{in}]$ (Height, Width, No. of channels)

For RGB images: $C_{in} = 3$

Input preprocessing: 1. Resize image (Why?:不是卷积层的要求，而是全连接层的要求). 2. Normalization $[0,255] \rightarrow [0,1]$ or $[-1,1]$ (Why?: Alleviate vanishing gradient problem (slow learning) and exploding gradient problem) & Standardization.

2. Convolution layer

Feature maps (activation maps) represent the strength of the signal of specific features at different spatial locations in the input data. High values in a feature map indicate the stronger signal of the detected feature (e.g. edges, texture, color) at that region.



Computation of feature map: weighted sum of input region (weights are from values of kernels). Mathematically it is cross-correlation. Real convolution has horizontal and vertical flip of kernel before computation.

→ 1 convolution = feature extraction (×) & information 混合 (+)

Feature extraction (×) can be regarded as pattern matching of the kernel to each region of the input (e.g. the above kernel can match the pattern "X"). As the kernel is learnable parameters, it means that the feature that the kernel would learn is determined by the model training.

Kernel: dimension = $C_{out} \times [K_h, K_w, C_{in}]$. Usually $K_h = K_w$

no. of kernels × [height, width, no. of input channel]

Feature map size after convolution

Kernel Size (核大小): Width x Height，Commonly 1x1，3x3，5x5，7x7……(Why usually use odd numbers?)

- Symmetry in padding (e.g. stride = 1, if we want $n_{out} = n_{in}$ what is p in terms of k?)
- Odd number size kernel has a unique center. In convolution, this center element aligns exactly over one input pixel, i.e. there is an equal number of

pixels on all sides around the input pixel. In view of the receptive field of an output pixel, it is more consistent for feature extraction.

Stride (步长): Number of pixels per slide

Padding (填充): To control the size of the output feature map and prevent loss of edge information

Output map size: N=(W-F+2P)/S+1

N: Output Size (输出大小)

W: Input Size (输入大小)

F: Kernel Size (卷积核大小)

P: Padding (填充值的大小)

S: Stride (步长大小)

Example

With input size 5x5, kernel=3 (x3), stride=1, padding=0

Output size is (5-3+2*0)/(1) + 1 = 3 both at width and height, which is size 3x3.

Dimensions of the convolved feature map

Multi-channel image + 1 multi-channel convolution kernel:

Number of kernel channels = Number of input channels

Each pixel value is convolved with the corresponding kernel channel value, generating an output convolution result. The three convolution results are summed at corresponding positions to obtain the final convolution result (single channel).

Multi-channel image + m multi-channel convolution kernels:

The output result has dimensions [C, H, W] and contains m channels.

C: Number of convolutional kernels

H: Height of feature map in each channel

W: Width of feature map in each channel

For an input RGB image, a 3x3 convolutional kernel indicates width 3, height 3, and 3 channels (omitted).

A 3x3x16 convolutional kernel indicates 16 3x3 convolutional kernels.

Padding

For each element in the feature map, it corresponds to one of the regions in the input, and the value in the feature map shows if that region shows a strong signal of the feature.

Why is the feature map dimension smaller in the above image?

To fit the kernel completely in the image, elements on the edge of the image cannot be the center of the kernel in convolution in the above case, so the center of the kernel can only be put inside.

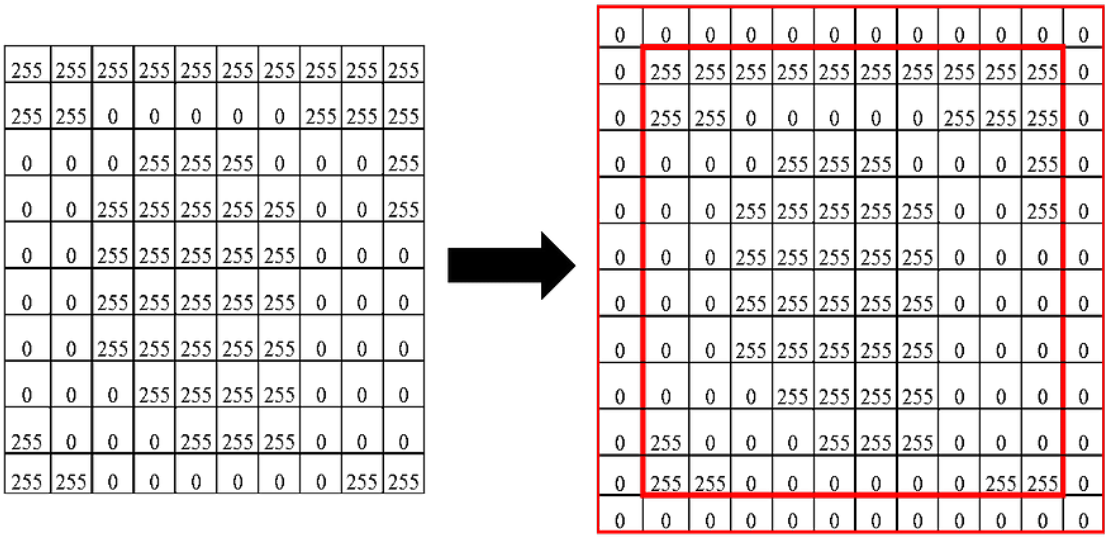
Problem: The information of the regions near the edge is lost because of the above reason.

Solution to the above problem:

Purpose of Padding: make use of the information near the edge of the image, so that the kernel can be put there, while controlling the dimension of output feature map

Zero padding:

Involves adding rows or columns filled with zeros to the border of the original array or image.



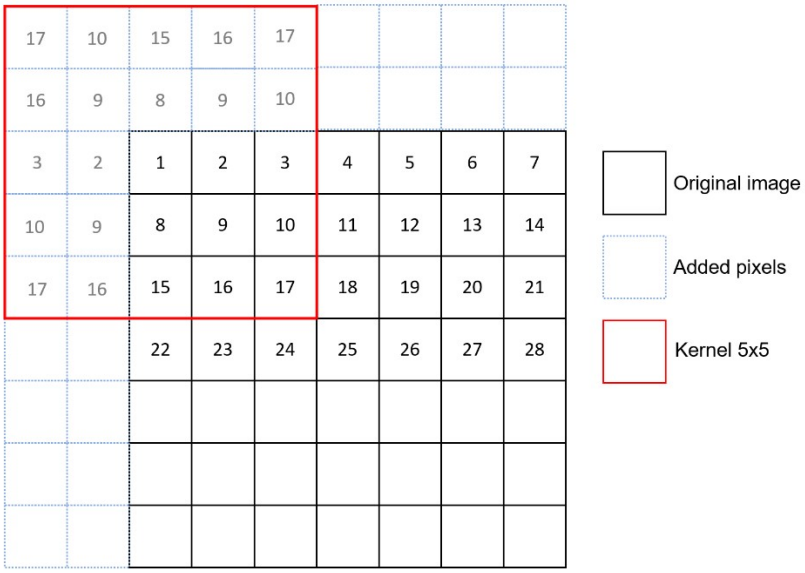
10x10的影像

因為kernel map為3x3，所以在圖的最外圈上下左右各加一行一列都為0的值，讓圖變成12x12

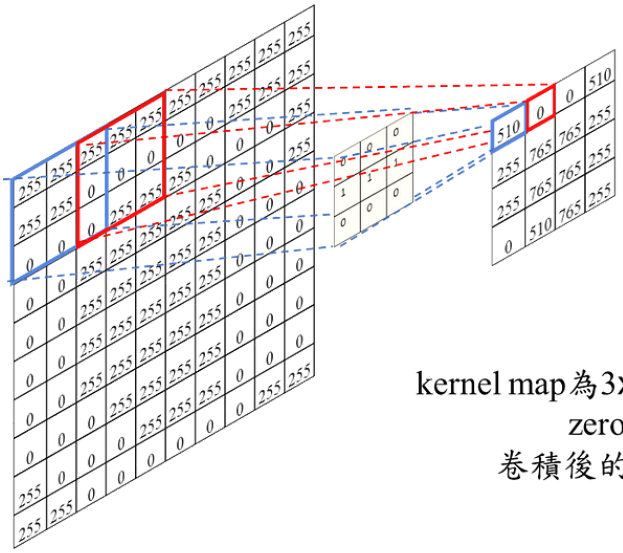
Reflection padding (mirror padding):

The pixels at the edges of the image are mirrored to create a boundary of reflected pixels.

Image of reflect padding: (commonly used in medical images)



Stride: number of blocks the kernel move, default=1 (higher stride, smaller feature map). Below is an example with stride=2:



10 x 10的影像

kernel map為3x3，stride=2，不做 zero-padding
卷積後的圖大小為4x4

Convolution Kernels

Receptive Field and Equivalence effects in kernel sizes

Receptive field: The area size in the input image corresponding to a specific pixel in an output feature map of a given layer, i.e., the "perceptual range" of the input data.

> Smaller kernel: fewer parameters, capture more local features (narrower receptive field)
>
> Larger kernel: more parameters, capture more global features (broader receptive field)

The effect of a single 5×5 convolution kernel is equivalent to two 3×3 kernels (with identical receptive fields for the output feature map).

$$RF_i = S_i(RF_{i+1}) + K_i$$

Why do practical networks commonly use two 3×3 kernels instead of a single 5×5 kernel? Because they require fewer parameters and less computational effort.

Reference:感受野以及与卷积核运算比较——低调学习 CNN(2) 卷积核与感受野-CSDN 博客

Why are so many convolutional layers used?

To gradually expand the receptive field for each feature map in order to understand a larger region of information in the input, including more non-linear information from the input.

Advantage of using 2 3x3 kernels:

Fewer computations (FLOPs, MACCs), Hierarchical feature learning (able to learn more nonlinear relationship due to using more activation function)

The equation of calculating Floating Point Operations (FLOPs) in a convolution layer:

$$\text{FLOPs} = (K_h \times K_w \times C_{in} + K_h \times K_w \times C_{in} - 1 + 1) \times H_{out} \times W_{out} \times C_{out}$$

考慮 bias

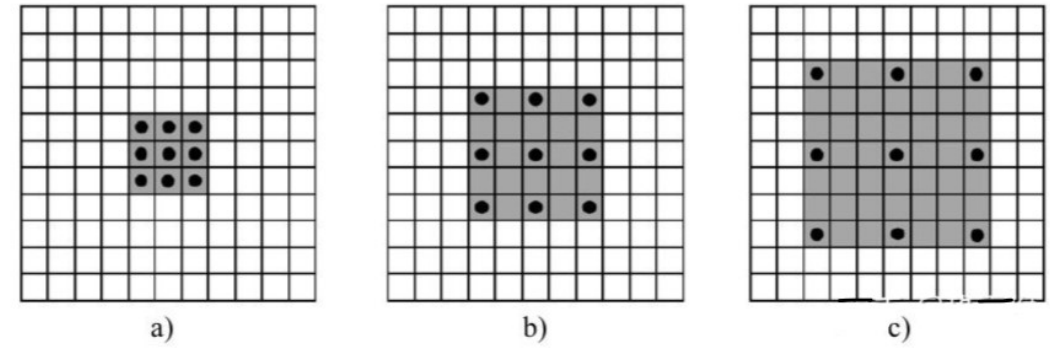
$$= 2 \times K_h \times K_w \times C_{in} \times H_{out} \times W_{out} \times C_{out}$$

https://medium.com/ching-i/cnn-parameters-flops-macs-cio-%E8%A8%88%E7%AE%97-9575d61765cc

Dilated convolution (空洞卷积)

Dilation was originally proposed to address image segmentation problems. Common image segmentation algorithms typically employ pooling layers and convolutional layers to increase the receptive field while simultaneously reducing the feature map resolution.

Subsequently, upsampling is used to restore the image size. The process of reducing and then enlarging feature maps results in accuracy loss. Therefore, an operation is needed that can increase the receptive field while maintaining the feature map size unchanged, thereby replacing the downsampling and upsampling operations.



a) is a standard convolution process (dilation rate = 1), with a receptive field of 3 after convolution.

b) is a dilated convolution with dilation rate = 2, with a receptive field of 5 after convolution.

c) is a dilated convolution with dilation rate = 3, with a receptive field of 7 after convolution.

Equivalent Convolution Kernel Size:

$$k' = k + (k-1) \times (d+1)$$

The receptive field calculation formula for the current layer is as follows, where RF_{i+1} represents the receptive field of the current layer, RF_i represents the receptive field of the previous layer, k′ represents the size of the convolutional kernel.

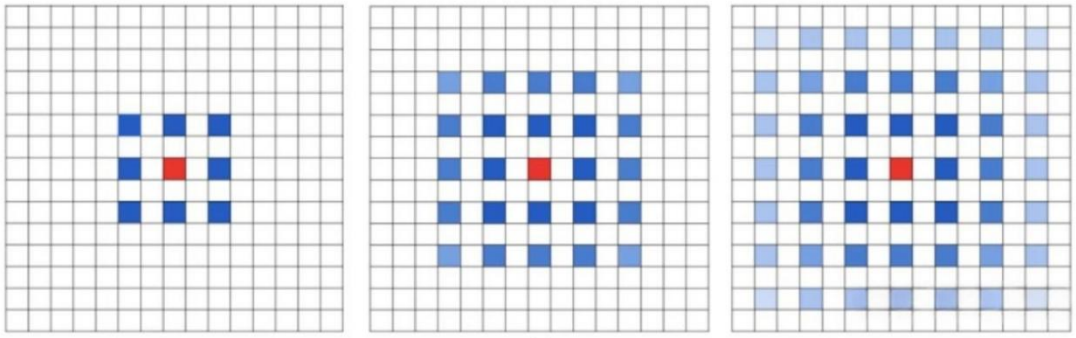
$$RF_{i+1} = RF_i + (k' - 1) \times S_i$$

Where S_i is the product of the step sizes of all previous layers (excluding current layer):

$$S_i = \Pi_{i=1}^{i} Stride_i$$

The Gridding Effect:

If we simply stack multiple 3x3 kernels with a dilation rate of 2, this issue will arise.



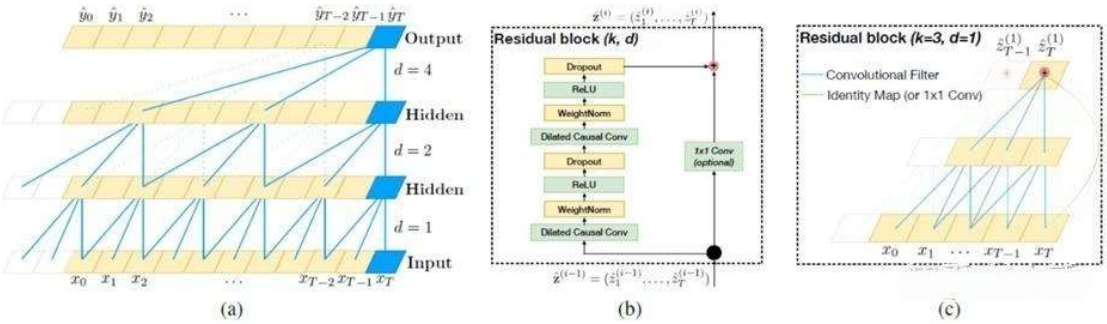
Due to the grid-like nature of dilated convolution computations, the convolutional results obtained at a given layer originate from independent sets of the preceding layer, lacking mutual dependencies.

Consequently, there is no correlation among the convolutional results within that layer, resulting in the loss of local information. This proves detrimental for pixel-level dense prediction tasks.

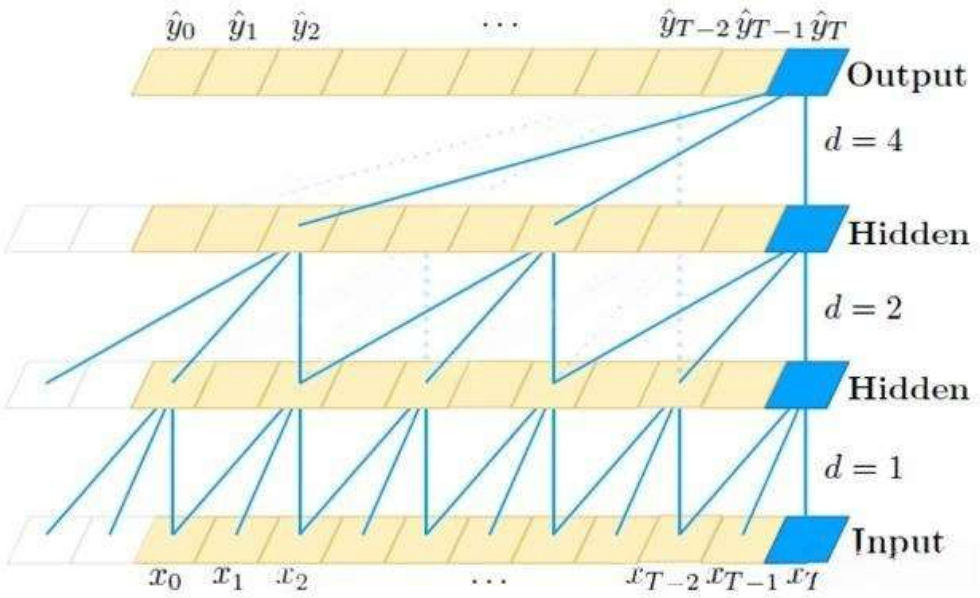
Reference: https://zhuanlan.zhihu.com/p/113285797

Temporal Convolutional Networks (TCN)

https://arxiv.org/pdf/1608.08242

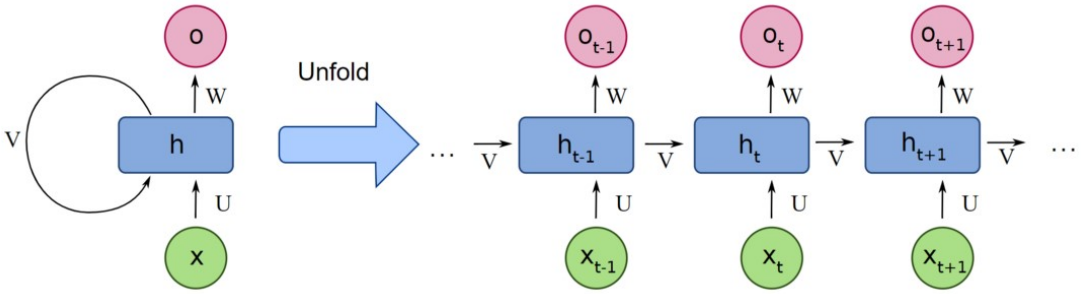


Traditional convolutional operations operate within local receptive fields, leading to challenges for original TCNs in capturing long-term dependencies within lengthy sequence data. However, by employing techniques such as deep TCNs and dilated convolutions, TCNs can expand their receptive fields and increase the distance over which information is propagated, thereby more effectively capturing and modeling long-term dependencies.



Compared to traditional recurrent neural networks (RNNs), TCNs employ convolutional operations, enabling efficient parallel computation. This grants TCNs advantages in training and inference on large-scale datasets, allowing them to better leverage the parallel computing capabilities of modern hardware.

Reference: https://zhuanlan.zhihu.com/p/23378168884



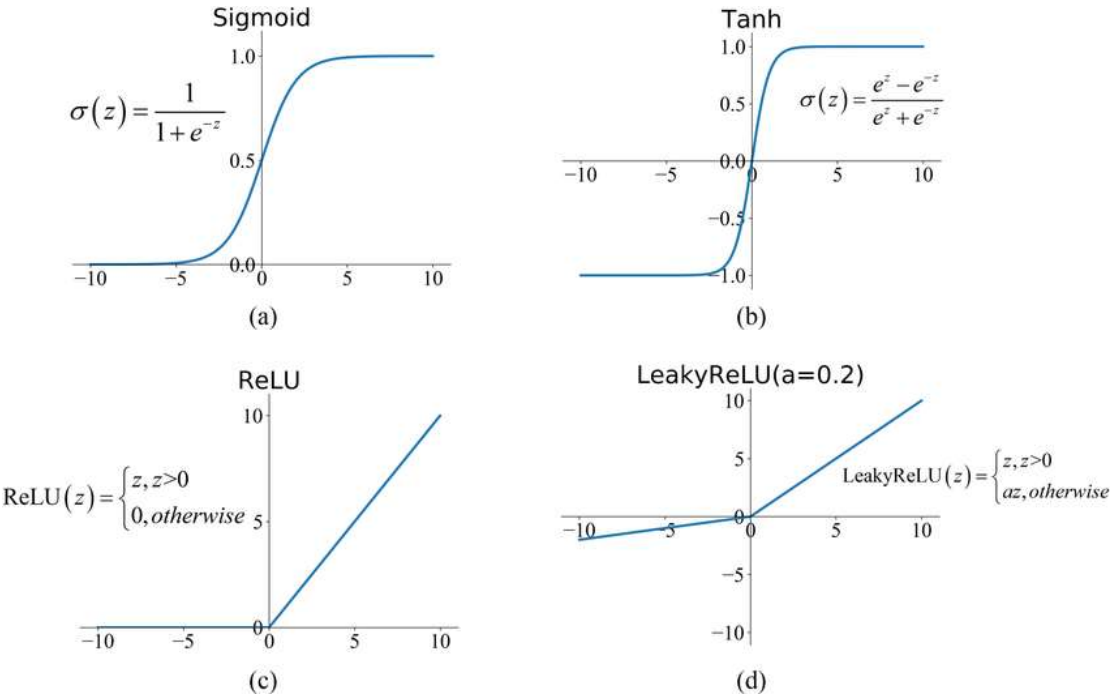
A typical RNN network structure

https://medium.com/@saba99/recurrent-neural-network-rnn-a56cef50d843

3. Activation function

Introduce nonlinearity to the model. (without it, the model only explains linearity as convolution is a linear processing).

Why nonlinear? (Real life example: dosage and treatment effect)



https://www.researchgate.net/figure/Commonly-used-activation-functions-a-Sigmoid-b-Tanh-c-ReLU-and-d-LReLU_fig3_335845675

4. Pooling layer:

Reduce the feature map spatial dimension and try to avoid overfitting

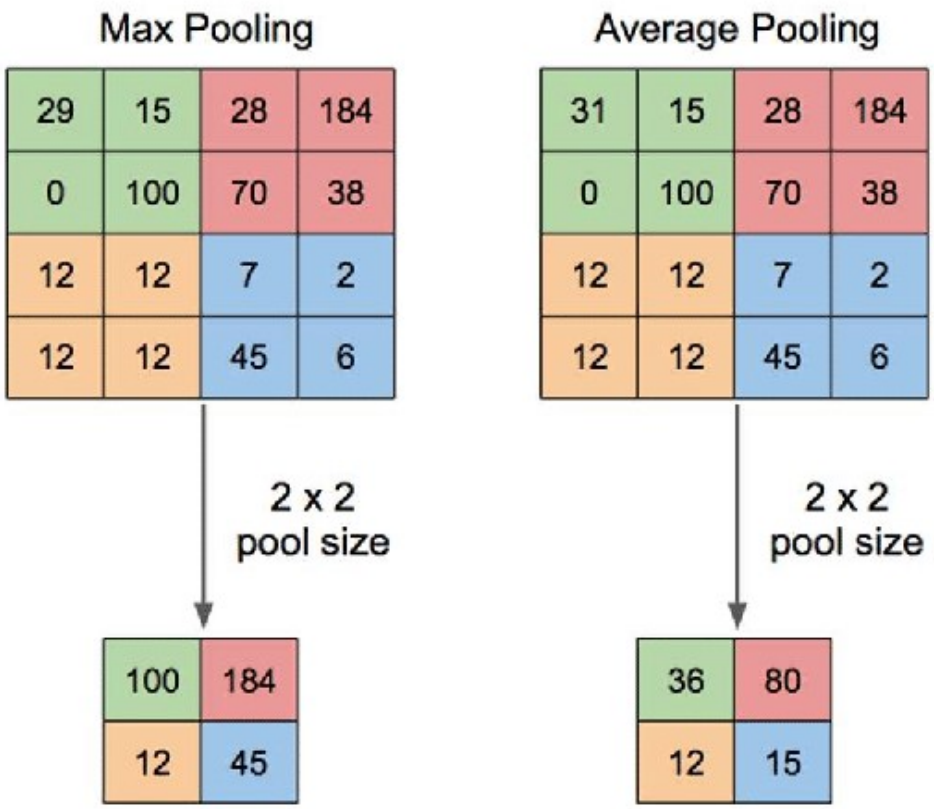
Pooling layers help mitigate overfitting by introducing a form of **spatial invariance**. By summarizing the presence of features over larger regions, pooling layers make the network **less sensitive to the exact position of features** within the input image. This invariance is beneficial for tasks such as image recognition, where the exact location of features (e.g., edges, textures) may vary across different images.

Moreover, the reduction in spatial dimensions achieved by pooling layers leads to a decrease in the number of parameters in the subsequent fully connected layers. This **reduction in parameters** helps to **prevent the model from becoming overly complex**, thereby reducing the risk of overfitting.

But there is information loss due to smaller size of feature map (not good for tiny object detection). The information related to the location of the region is lost as we cannot tell from the output the spatial location of the pixel this value corresponds to in the input.

Max Pooling selects the maximum element from the region of the feature map as the output.

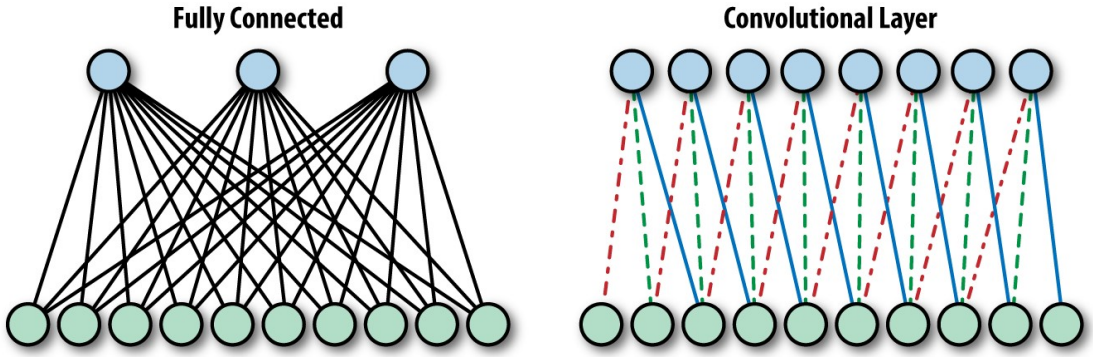
Average pooling computes the average of the elements present in the region of feature map.



Max pooling is better than average pooling in **capturing sharp features** (keep the strongest signal within the region).

Average pooling can give a **smoother feature map**, and provide a more generalized representation of the input.

5. Fully-connected layer



https://www.oreilly.com/library/view/learning-tensorflow/9781491978504/ch04.html

Blue neurons are the input, while green neurons are the output.

In a **fully connected layer** (left), each unit is **connected to all units** of the previous layers. In a **convolutional layer** (right), each unit is **connected to a constant number of units** in a local region of the previous layer. Furthermore, in a **convolutional layer**, the units all **share the weights** for these connections, as indicated by the shared line

types.

Dimension of FC layer is chosen based on the task requirement

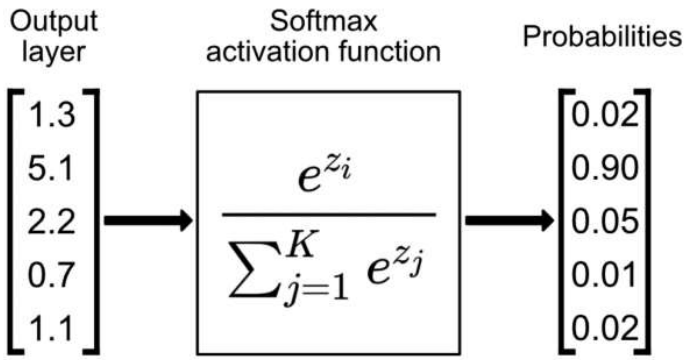
e.g. the task is 5-class classification, we need 5 neurons as the output.

Because every neuron is interconnected in FC layer, the output is more capable to capture more information from the feature map than using other layers.

Problem of FC layer: high computation cost, information related to the location of the pixel is lost (see the comparisons between YOLOv1 and YOLOv2)

6. Output layer

Classification: softmax function (values → probability distribution). Choose the top 1 class as the prediction (i.e. the class with the highest probability).



https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60/

Problem of softmax function:

- e.g. if the output layer is [0,0,0,0,0.00001], with values near 0 representing lower confidence of classifying the image as the class, we know that this image should not be classified as any of the class here, but softmax function would amplify the minor differences in value between class 5 and other classes, and pick class 5 as the output class.
- e.g. an image of apple. This belongs to both the apple class and fruit class. If we want multiclass classification, softmax function cannot be directly used across classes as this function would assign the image to the class with the highest probability weighted on all classes, which is not reasonable.

Regression: usually 1 value (e.g. prediction of housing price), depending on the task (e.g. bounding box location: x,y,w,h 4 values)

How do we know the model perform well or not?

Precision, Recall, mAP (see the part in evaluation metrics)

How does the model learn from training dataset?

Loss function: measure the difference (error) between ground truth and prediction

e.g. MSE loss, MAE loss, BCE loss

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$L_{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$L_{BCE} = \frac{1}{n}\sum_{i=1}^{n}(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i))$$

n: number of observations

y_i: ground truth

$\hat{y}_i$: prediction

Goal of training: minimize the loss function by updating the model weights

Method of updating the model weight: calculate the gradient of the loss function with respect to the weights in the network (**backpropagation**)

Forward propagation and backward propagation

Forward propagation refers to the aforementioned process.

Backpropagation compares the training results with the ground truth (calculating the loss), updates the parameters, and minimizes the loss.

(Loss 和 Optimizer 实际中很难有优化空间，学会使用即可)

In practice, training typically does not process the entire dataset at once. Instead, the data is divided into multiple batches. Each optimization iteration operates on a single batch. Completing one batch constitutes one update, and completing all batches constitutes one epoch.

1. Batch

Batch size refers to the number of records in a batch.

Smaller batch size leads to More frequent updates, and result in unstable update step.

Larger batch size leads to Less updates, but steps are more precise, making it less prone to moving in the wrong direction.

Full batch: If N represents the entire dataset and the batch size equals N, this is referred to as a full batch.

Batch size influence on training

1. Which takes less time to train? Small batch or large batch?

Large batches are more time-efficient.

Reason: Large batches can save computation time through parallel processing, and a larger batch size means fewer batches are needed, resulting in fewer updates and shorter epoch durations.

2. Which yields better training results: small batch or large batch?

Small batch.

Reason: Small batch movements are relatively noisy, and this characteristic makes it less likely to get stuck when gradient values are small.

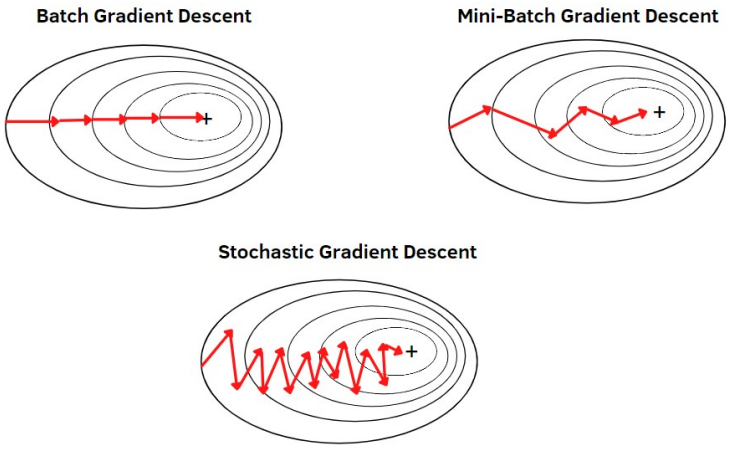
Another possible reason is that some believe large batches may lead to worse sharp minima during gradient descent, while small batches tend to guide toward better flat minima.

Backpropagation strategies

Optimizers: e.g. Stochastic gradient descent (SGD), Momentum, AdaGrad, Adam (not much to be modified by us as they are more in maths)

SGD: update model weights based on one training sample

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$



Where W is weight, L is loss function， η is learning rate， $\partial L/\partial W$ is the Gradient (Derivative) of the Loss Function with Respect to Parameters.

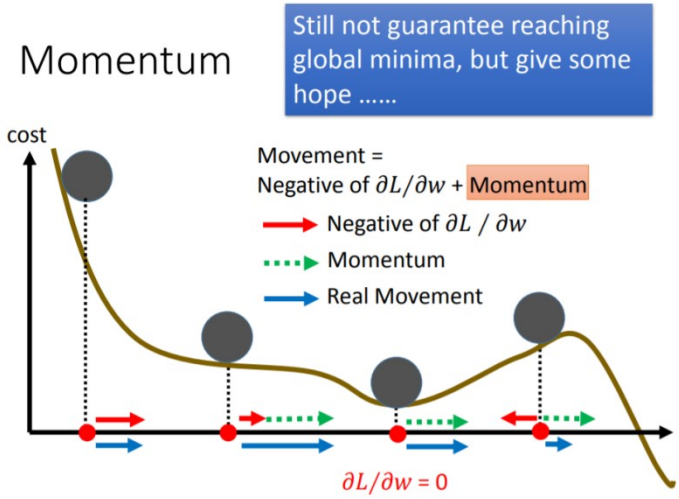
Advantage: low computation cost for each update

Problem: may get stuck in local minima, slow convergence, oscillations (because the gradient calculated in each step can vary a lot)

Momentum: This optimizer simulates the concept of physical momentum: learning speeds up along the same directional dimension and slows down when the direction changes.

$$V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + V_t$$



Here an additional parameter Vt is introduced, which can be conceptualized as a "velocity vector." It relates to the previous update: if the gradient direction from the last update matches the current one, $|Vt|$ (velocity) will increase progressively (indicating gradient amplification). causing the update gradient for the W (weight)to accelerate. If the directions differ, $|Vt|$ becomes smaller than the previous value (indicating a weaker gradient), and the update gradient for W slows down. β can be thought of as air resistance or ground friction, typically set to 0.9.

Advantage: faster convergence as it accelerates along constantly downhill dimensions, reduce oscillation, more hope to escape from local minima and saddle point

Problem: can overshoot around the minimum point

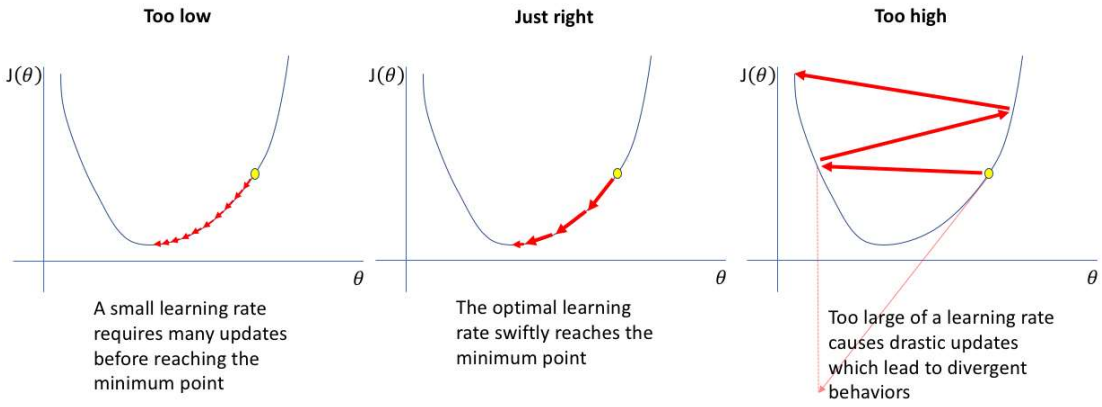
AdaGrad: Adjust the learning rate $\boldsymbol{\eta}$ based on the gradient (SGD and Momentum employ a fixed learning rate throughout the entire training process).

$$W \leftarrow W - \eta \frac{1}{\sqrt{n+\epsilon}} \frac{\partial L}{\partial W}$$

$$n = \sum_{r=1}^{t} \left(\frac{\partial L_r}{\partial W_r}\right)^2$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{\sum_{r=1}^{t} \left(\frac{\partial L_r}{\partial W_r}\right)^2 + \epsilon}} \frac{\partial L}{\partial W}$$

We do not know the best value of learning rate before training. What would happen if we manually set the learning rate too low or too high?



Advantage: No need manual tuning of learning rate (the model can decide how fast/slow it learns from the dataset), works well with features that appear infrequently, as it can assign higher learning rates to them.

Problem: Not ideal for non-sparse, long-running tasks.

Adam (most commonly used optimizer among all 4):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L_t}{\partial W_t} \qquad \hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$v_t = \beta_1 v_{t-1} + (1-\beta_2)(\frac{\partial L_t}{\partial W_t})^2 \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

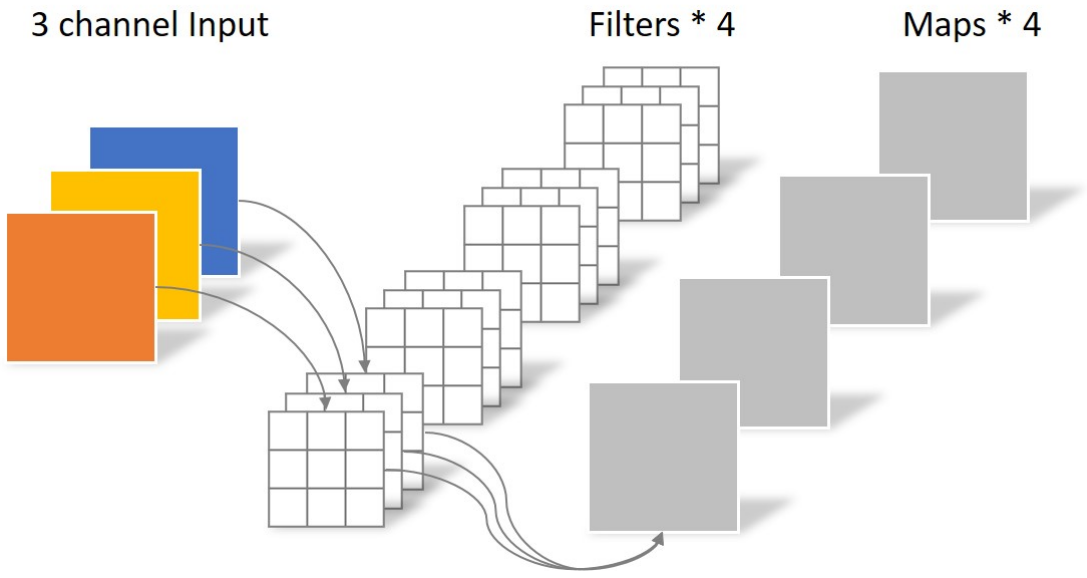
$$W \leftarrow W - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Adam retains Momentum's adjustment of gradient velocity based on past gradient directions and Adam's adjustment of the learning rate based on the squared values of past gradients. Additionally, Adam incorporates parameter “bias correction,” ensuring that the learning rate remains within a defined range for each iteration. This results in a smoother update of parameters.

Depthwise Separable Convolution (DW Conv)

Problem of traditional CNN: Too complex (high FLOPs)

Traditional CNN: each kernel convolves with the input across every channel. Number of output channels is the number of kernels.

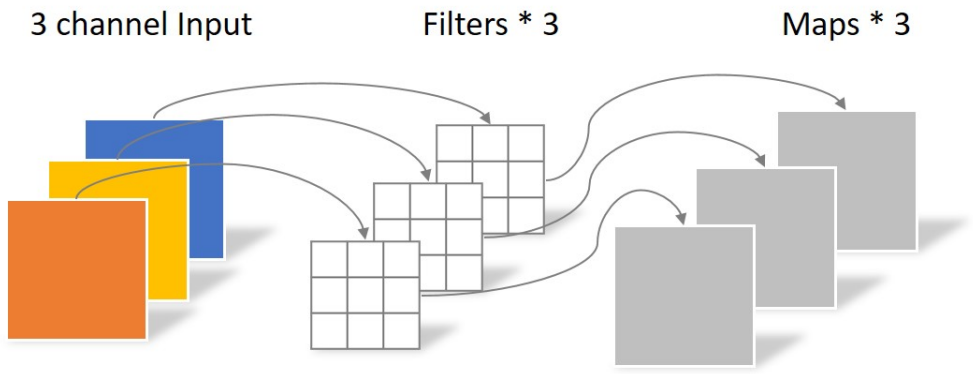


https://blog.csdn.net/BigDream123/article/details/115111062

Recap: 1 convolution = feature extraction (×) & information 混合 (+). Can we separate these two steps?

Depthwise convolution: convolve each input channel independently with its own

kernel, no mixing across channels here. Number of output channels is the number of input channels.

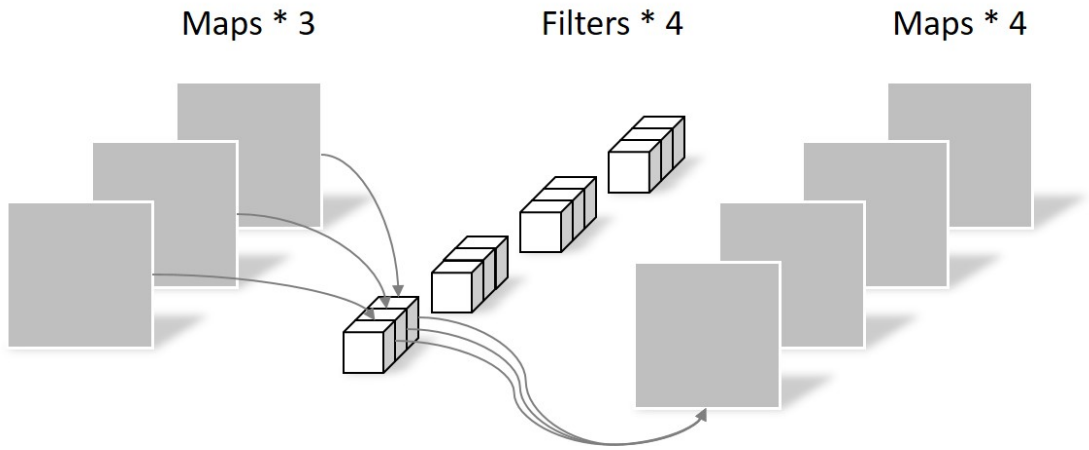


https://blog.csdn.net/BigDream123/article/details/115111062

Pointwise convolution:

Combines information from all maps on the same spatial location.

Kernel dimension: $1 \times 1 \times$ depth $\times$ no. of kernels



https://blog.csdn.net/BigDream123/article/details/115111062

Practice: Compare the FLOPs of traditional convolution layer and depthwise convolution with pointwise (Kernels layer.

Advantage of DW Conv compared with traditional conv: fewer computation

Problem: Less capable to capture information across image channels. Only using 1x1 kernel in pointwise convolution to learn information across channels, while traditional conv uses larger kernels to learn across channel information.

Note: Fewer computations do not imply more efficient computing:

e.g. With GPU, parallel computation can handle arithmetic operations simultaneously.

For traditional convolution, as the computation of each element in output feature map only depends on its input region, traditional CNN can utilize parallel computing, more efficient. But pointwise convolution in DWConv requires the output from depthwise convolution, there is a bottleneck if using parallel computation, thus not as efficient as training traditional CNN.

MobileNetv1 v2 v3: https://blog.csdn.net/weixin_37737254/article/details/114325409