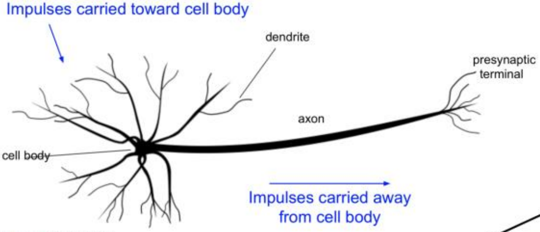
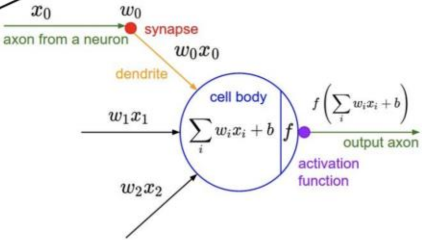
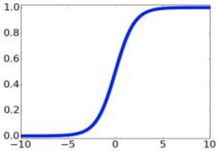
Neural Network Layers

Neural network analogy



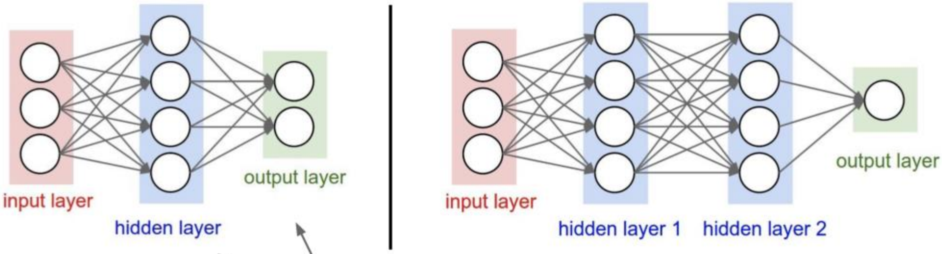




sigmoid activation function

$$\frac{1}{1+e^{-x}}$$

Neural Network Architectures



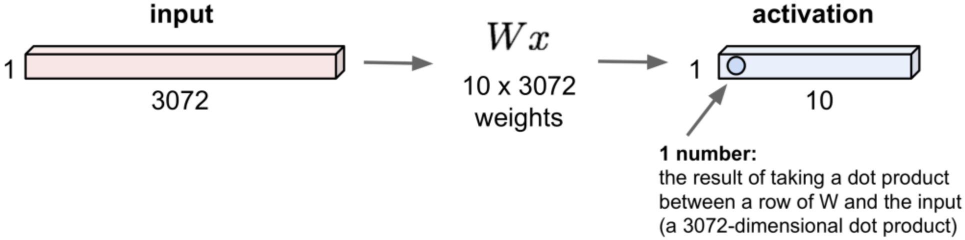
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"Fully-connected" layers

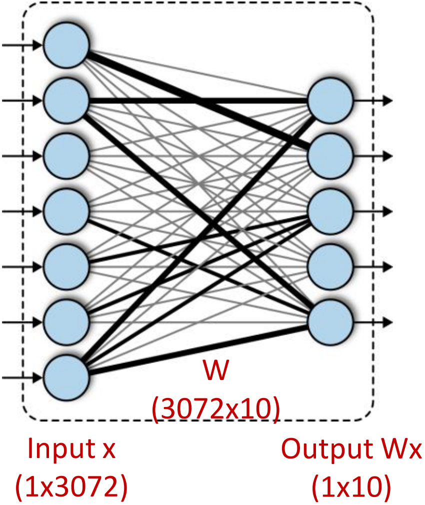
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Fully Connected Layer

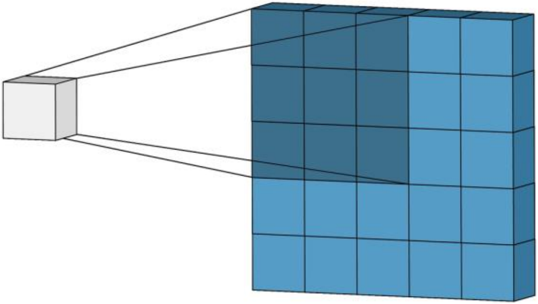
32x32x3 image -> stretch to 3072 x 1



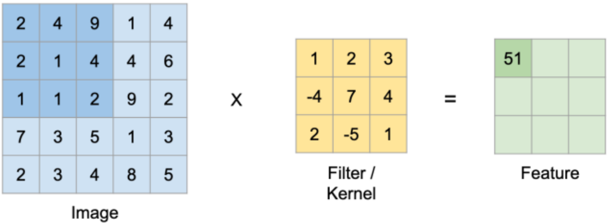
Fully Connected Layer

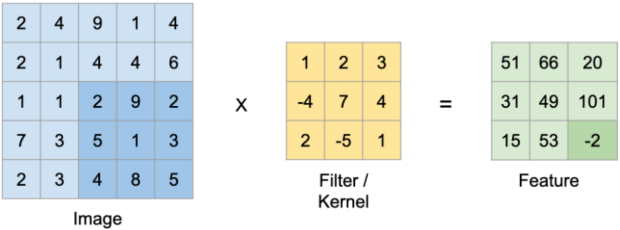


Convolutional layer



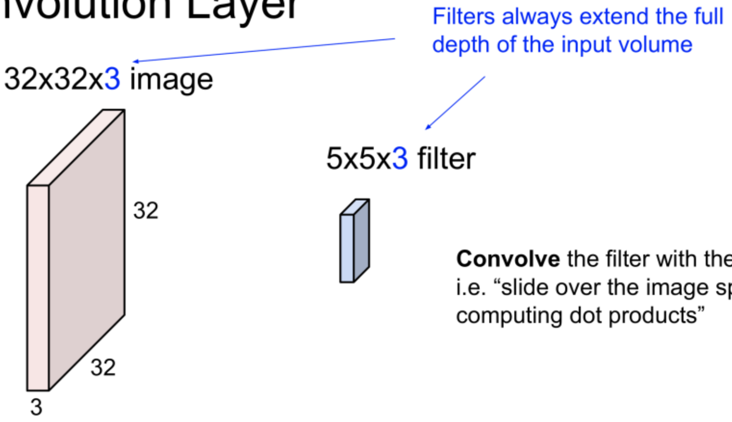
Convolutional layer





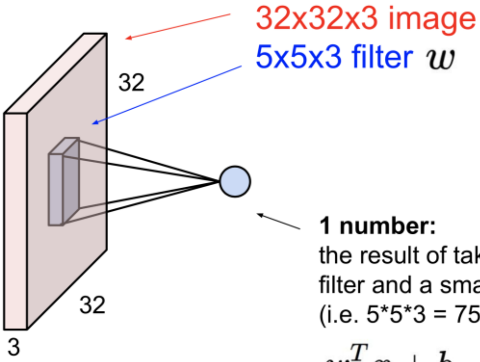
Convolutional layer

Convolution Layer



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

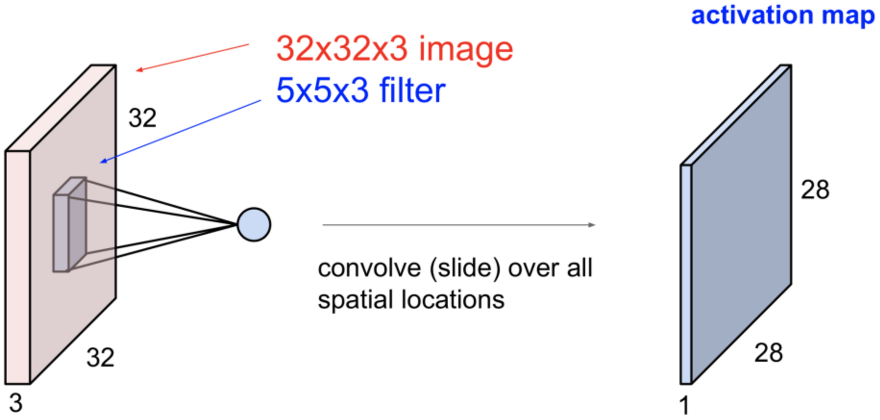
Convolutional layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

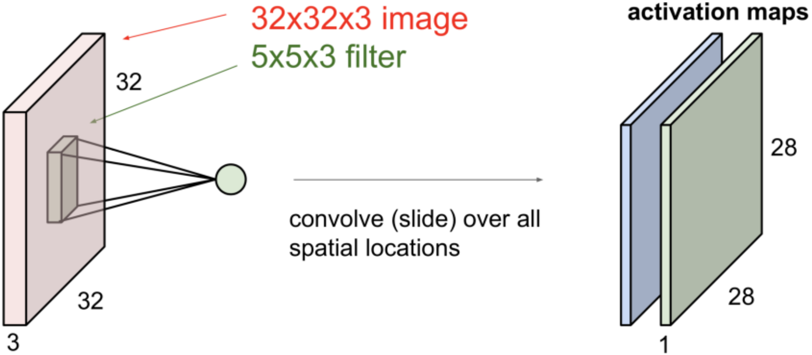
Convolutional layer



Convolutional layer

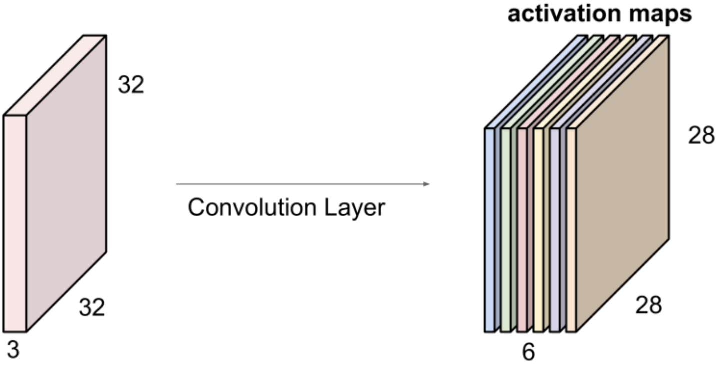
Convolution Layer

consider a second, green filter



Convolutional layer

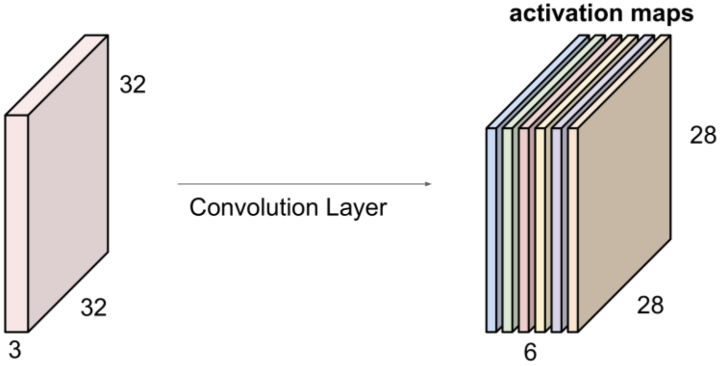
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Convolutional layer

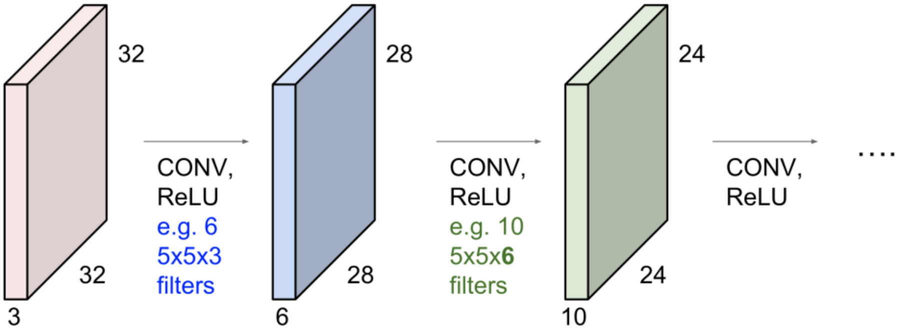
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



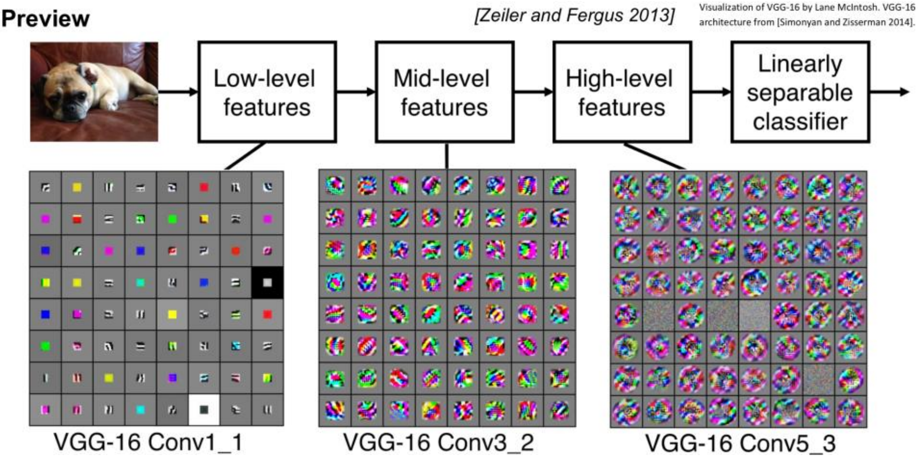
We stack these up to get a "new image" of size 28x28x6!

Convolutional layer

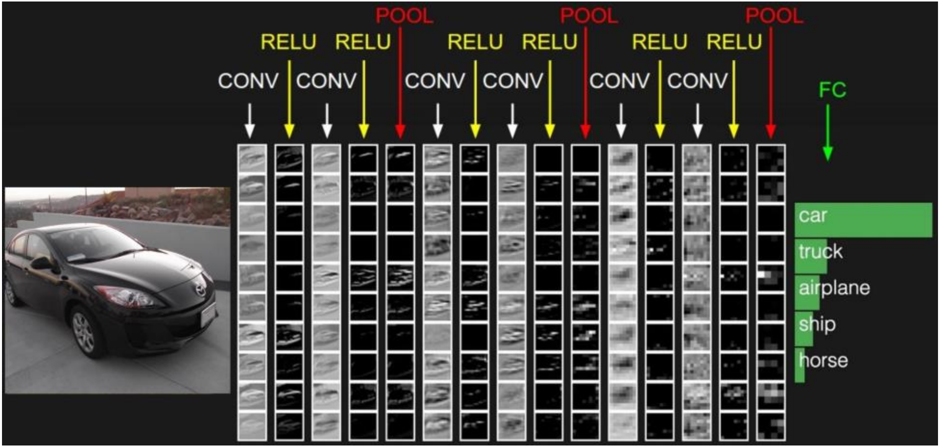
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Convolutional layer

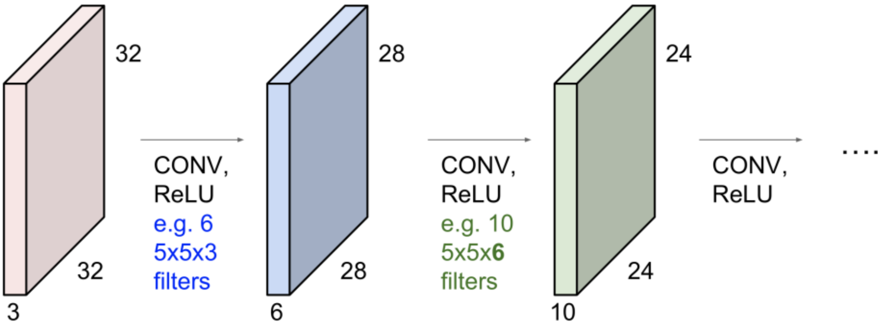


Convolutional layer



Convolutional layer

Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Convolutional layer

0	0	0	0	0	0	0
0	2	4	9	1	4	0
0	2	1	4	4	6	0
0	1	1	2	9	2	0
0	7	3	5	1	3	0
0	2	3	4	8	5	0
0	0	0	0	0	0	0

Image

X

1	2	3
-4	7	4
2	-5	1

Filter / Kernel

=

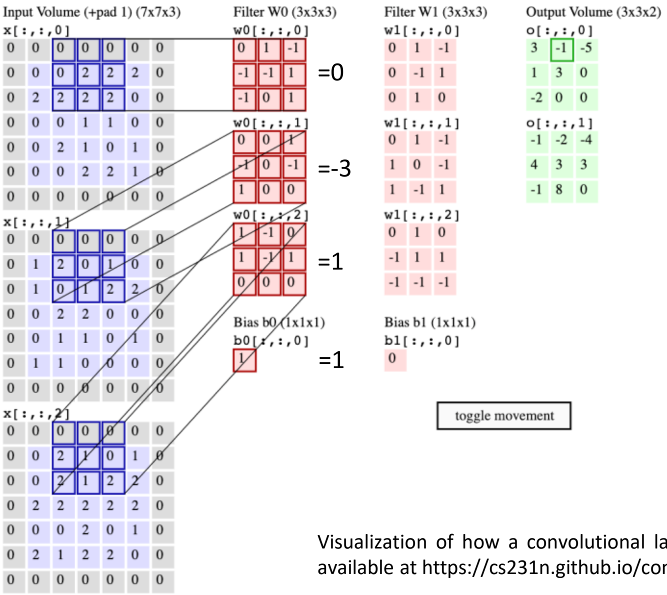
21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Feature

- Input 5x5 with padding 1.
- Filter with kernel size 3x3 and stride 1.

→ Output is 5x5.

Convolutional layer



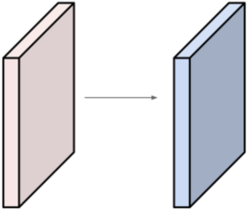
Visualization of how a convolutional layer works. Animation available at https://cs231n.github.io/convolutional-networks/

Convolutional layer

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

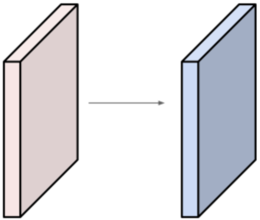
Output volume size: ?



Convolutional layer

Examples time:

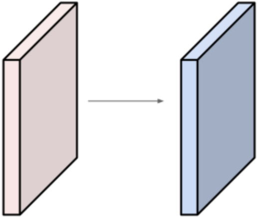
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Convolutional layer

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

Convolutional layer

Summary of the Convolutional layer:

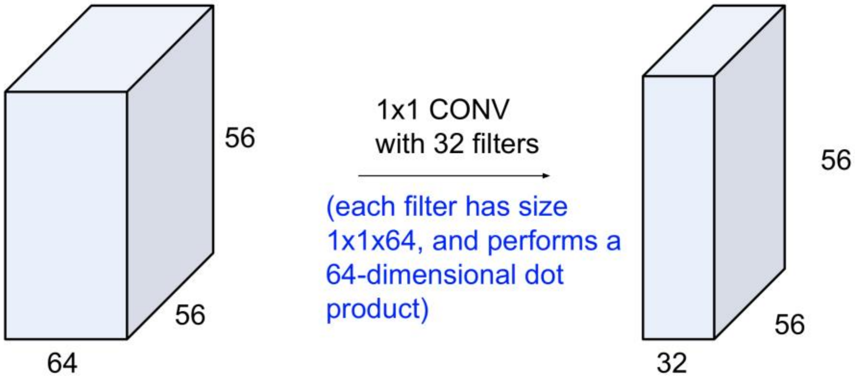
- Accept a volume of size $W_1 \times H_1 \times D_1$
- Require four hyperparameters:
    - Spatial extent F
    - Number of filters K
    - Stride S
    - Zero padding P
- Produce a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 + 2P - F)/S + 1$
    - $H_2 = (H_1 + 2P - F)/S + 1$
    - $D_2 = K$
- With parameter sharing, it introduces $F \times F \times D_1$ weights per filter, for a total of $(F \times F \times D_1) \times K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result performing a valid convolution of the d-th filter over the input volume with a stride S, and then offset by d-th bias.

Convolutional layer

Question: What is the reason for which the Convolutional layer was born? Does it make sense to have a Convolutional layer with kernel size 1x1? Why or why not?

One-by-one convolution layer

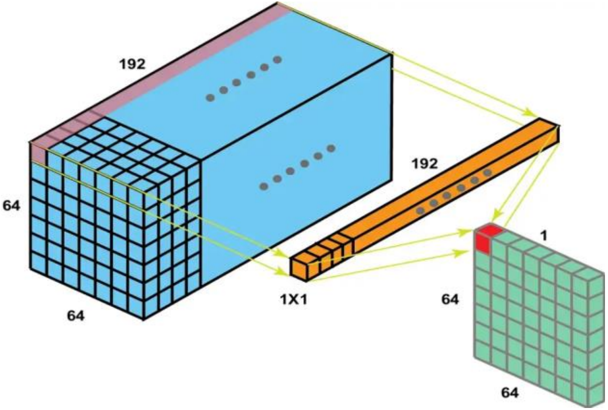
(btw, 1x1 convolution layers make perfect sense)



One-by-one Convolutional layer

One-by-one (1x1) convolutional layer is used for:

- Dimensionality reduction.
- Dimensionality augmentation.



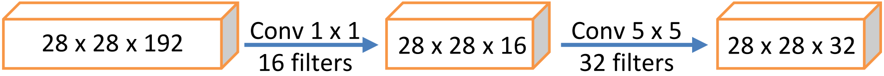
One-by-one Convolutional layer

One-by-one (1x1) convolutional layer is used for:

- **Dimensionality reduction**.
- Dimensionality augmentation.

Number of learnable parameters = ?
Number of operations = ?

Assume that there is **no bias** for the sake of simplicity.



Number of learnable parameters = ?
Number of operations = ?

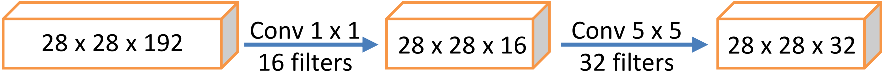
Assume that there is **no bias** for the sake of simplicity.

One-by-one Convolutional layer

One-by-one (1x1) convolutional layer is used for:

- **Dimensionality reduction**.
- Dimensionality augmentation.

Number of learnable parameters = (5 x 5 x 192) x 32 = **153,600** params.
Number of operations = (5 x 5 x 192) x (28 x 28) x 32 = **120,422,400** ops.

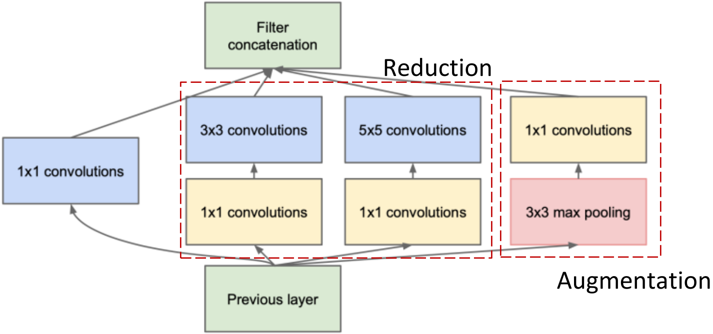


Number of learnable parameters = (1 x 1 x 192) x 16 + (5 x 5 x 16) x 32 = **15,872**.
Number of operations = (1 x 1 x 192) x (28 x 28) x 16 + (5 x 5 x 16) x (28 x28) x 32
= **12,443,648** ops.

One-by-one Convolutional layer

One-by-one (1x1) convolutional layer is used for:

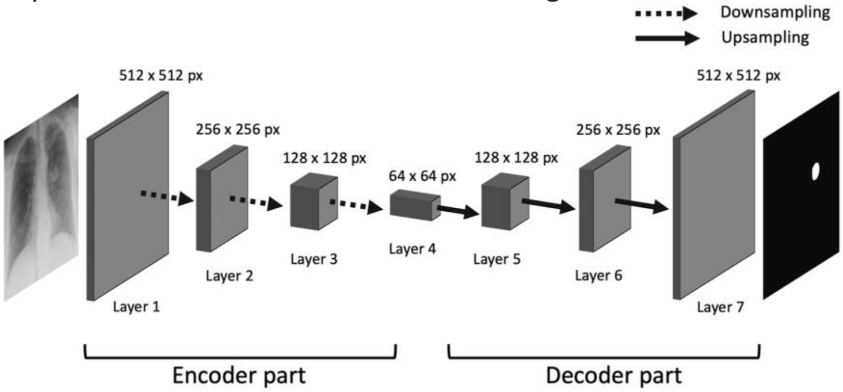
- Dimensionality reduction (more **popular** than augmentation).
- **Dimensionality augmentation:** increase the number of feature maps after pooling, artificially creating *more projections* of the *down-sampled* feature map content.



Inception module with 1x1 conv for dimensionality reduction and augmentation.

Transposed convolutional layer

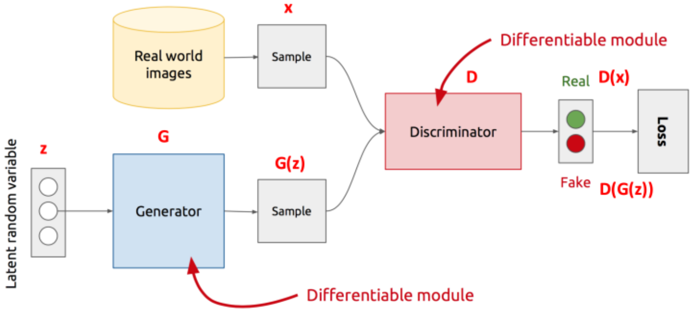
- Convolutional layer: **down-samples** the input.
- Transposed convolutional layer: **up-samples** the spatial dimensions, usually used in **auto-encoders** and **GANs**, or generally any network that must **reconstruct** an image.



Auto encoder network

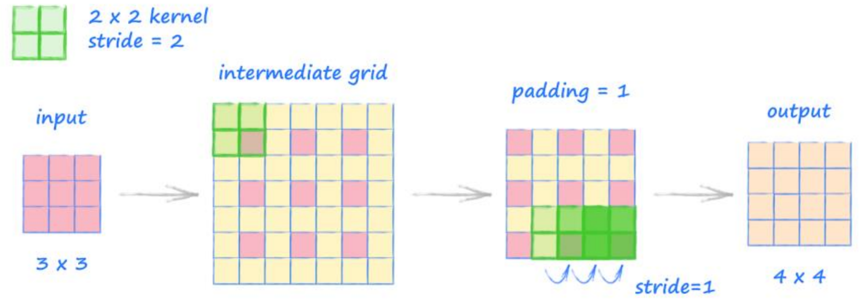
Transposed convolutional layer

- Convolutional layer: **down-samples** the input.
- Transposed convolutional layer: **up-samples** the spatial dimensions, usually used in **auto-encoders** and **GANs**, or generally any network that must **reconstruct** an image.



Generative Adversarial Network (GAN)
Transposed convolutional layers are used in the Generator (G).

Transposed convolutional layer



Input **N** x **N**
Filter size **F x F**

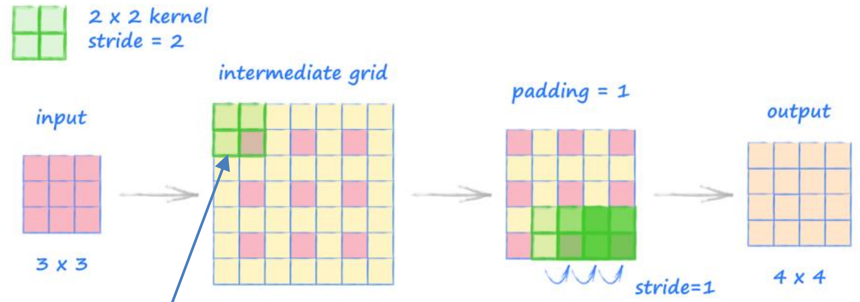
Stride **S** between input cells

Remove padding **P**, then apply standard convolution with S=1

Output

Question: what is the output size of a transposed convolutional layer, given input **N x N**, stride **S** and padding **P**?

Transposed convolutional layer



Input **N** x **N**
Filter size **F x F**

Stride **S** between input cells
($Grid_{stride}$)

Remove padding **P**, then apply standard convolution with S=1
($Grid_{pad}$)

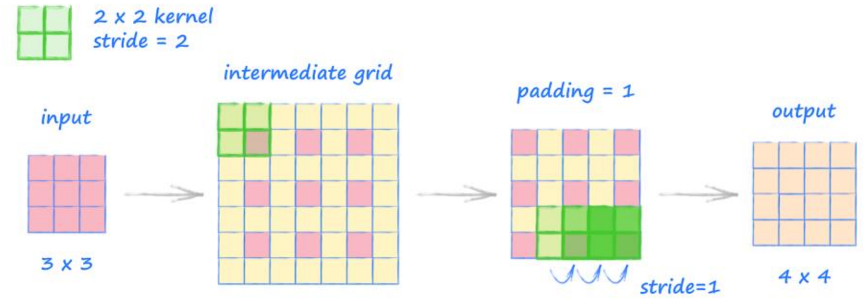
Output

Add the maximum amount of these so that a kernel in the top left covers **one** of the original cells.

$$Grid_{stride} = N + (N-1) \times (S-1) + \overbrace{(F-1)} \times 2$$

$$Grid_{pad} = Grid_{stride} - 2 \times P = N + (N-1) \times (S-1) + (F-1) \times 2 - 2 \times P$$
$$= (N-1)S + 2F - 2P - 1$$

Transposed convolutional layer



Input **N** x **N**
Filter size **F x F**

Stride **S** between input cells ($Grid_{stride}$)

Remove padding **P**, then apply standard convolution with S=1 ($Grid_{pad}$)

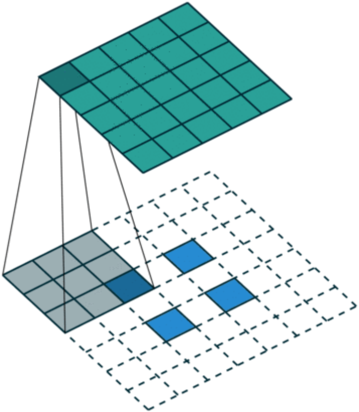
Output

$Output = \frac{Grid_{pad}+2\times P'-F}{S'} + 1 = Grid_{pad} - F + 1$ where P' is always 0 and S' is always 1.

$\boldsymbol{Output} = (N-1)S + 2F - 2P - 1 - F + 1 = \boldsymbol{(N-1)S + F - 2P.}$

Above example: $\text{Output} = (3-1)2 + 2 - 2 = 4$. Hence, the output feature map is 4x4.

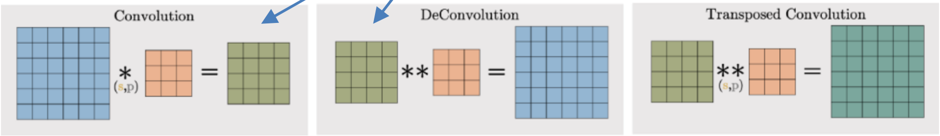
Transposed convolutional layer



Question: In this transposed convolutional layer,
what are input size **N**, stride **S**, filter size **F** and padding **P**?

Deconvolution

Deconvolution is a mathematical operation that reverses the process of a convolutional layer, i.e., get an input through a convolutional layer, then *get its output through a deconvolutional layer*, we get back the **exact same** input.



Deconvolution is not transposed convolution. People often misinterpret “transposed convolution” as “deconvolution”.

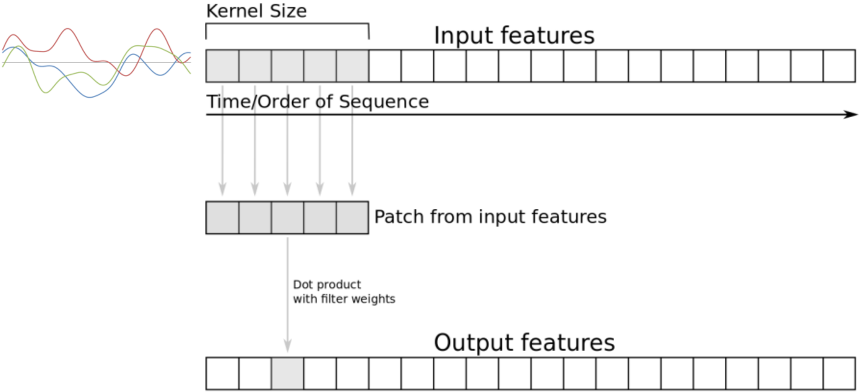
Dimensions of convolutions

- **1D Convolution**: kernel moves in **1 direction**. Input and output data of 1D CNN is 2 dimensional. Mostly used on time-series data.
- **2D Convolution**: kernel moves in **2 directions**. Input and output data of 2D CNN is 3 dimensional. Mostly used on image data.
- **3D Convolution**: kernel moves in **3 directions**. Input and output data of 3D CNN is 4 dimensional. Mostly used on 3D image data (MRI, CT Scans, Video).

1D convolution is not 1x1 convolution.

1D Convolution

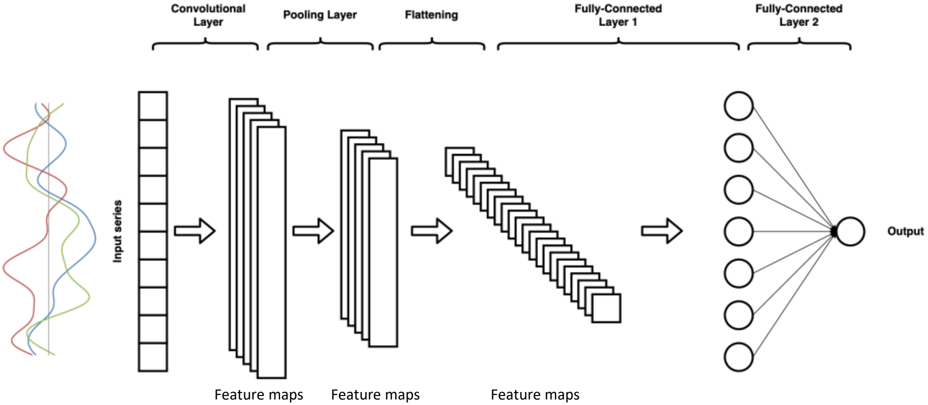
- **1D Convolution**: kernel moves in **1 direction**. Input and output data of 1D CNN is 2 dimensional. Mostly used for **forecasting time-series/sequential data**.



1D convolutional layer operates on time-series/sequential data

1D Convolution

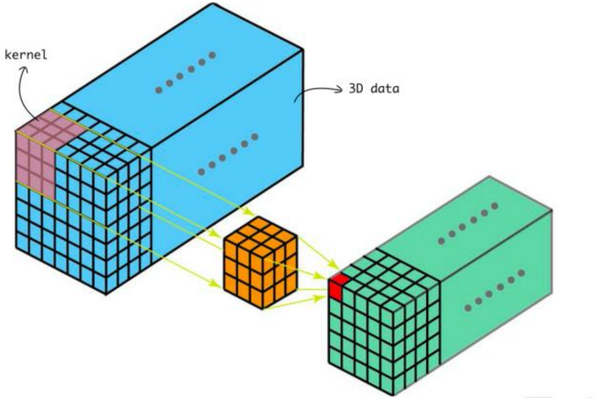
- **1D Convolution**: kernel moves in **1 direction**. Input and output data of 1D CNN is 2 dimensional. Mostly used for **forecasting time-series/sequential data**.



An example of neural network architecture to process time-series/sequential data.

Three-Dimensional (3D) Convolutional layer

- **1D Convolution**: kernel moves in **1 direction**. Mostly used on **time-Series data**.
- **2D Convolution**: kernel moves in **2 directions**. Mostly used on **image data**.
- **3D Convolution**: kernel moves in **3 directions**. Mostly used on **3D image data** (MRI, CT Scans, Video).

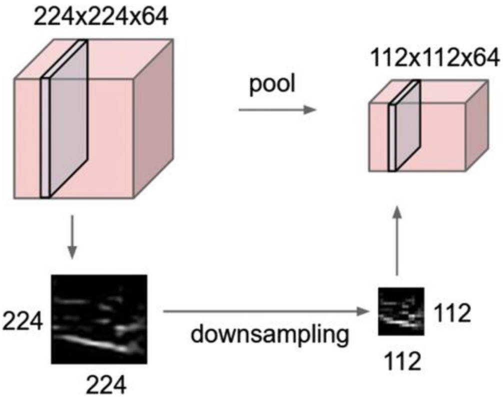


3D convolutional layer operates on 3D image data

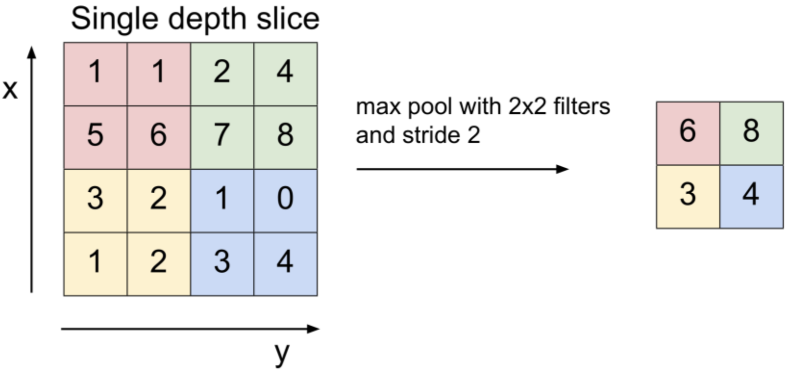
Pooling layer

Pooling layer:

- makes the representations **smaller** and more manageable.
- operates on each activation map independently.



Pooling layer

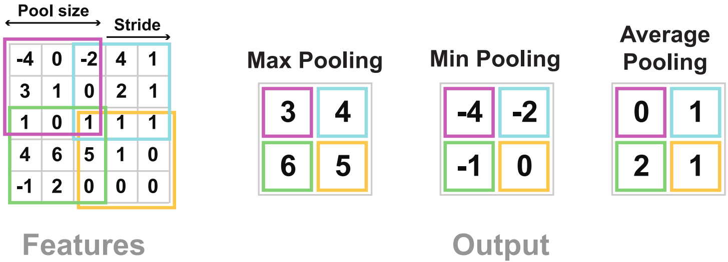


Example of max pooling operation

Pooling layer

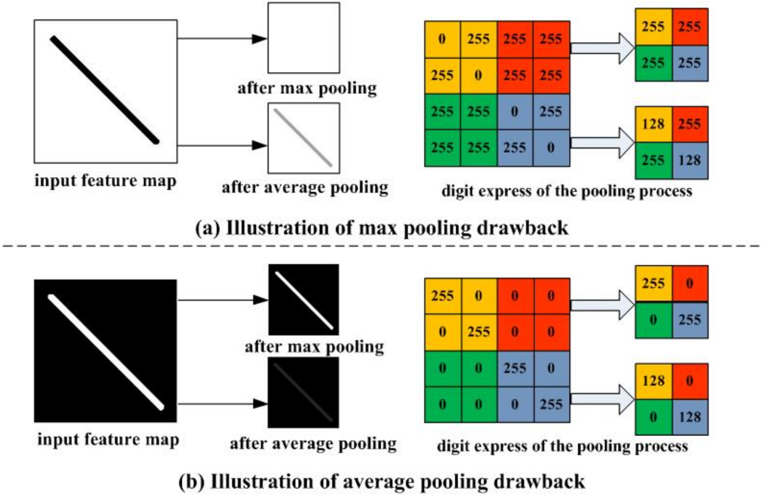
Various pooling layers:

- max pooling
- min pooling
- average pooling
- sum pooling
- …



Example of different types of pooling layers.

Pooling layer



Comparison between max pooling and average pooling.

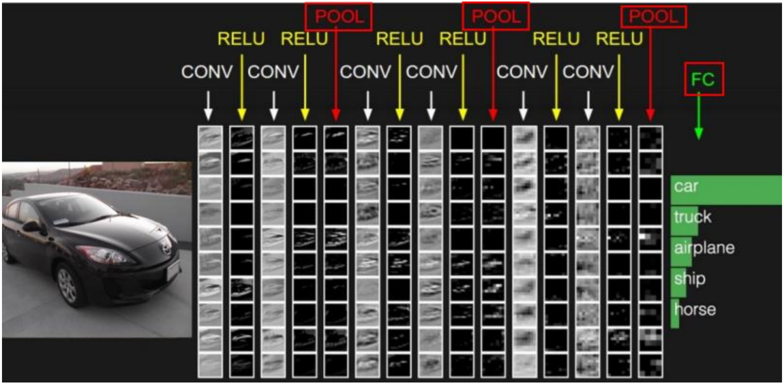
Pooling layer

Summary of the Pooling layer:

- Accept a volume of size $W_1 \times H_1 \times D_1$
- Require three hyperparameters:
  - Spatial extent F
  - Stride S
- Produce a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- It introduces zero parameters since it computes a fixed function of the input.
- Note that it is not common to use zero-padding for Pooling layer.

Fully Connected Layer (FC Layer)

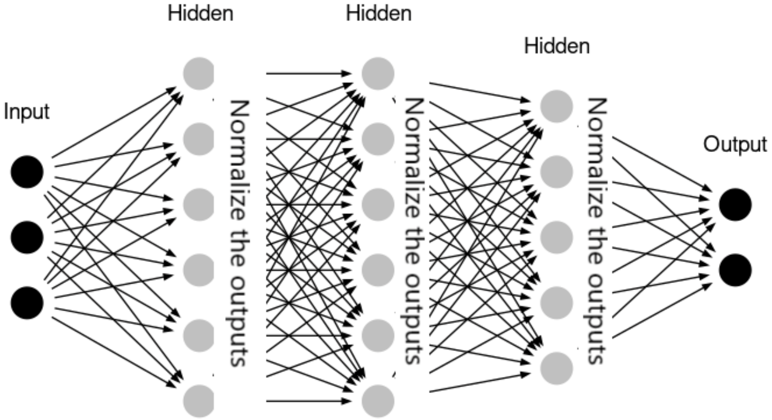
FC layer: contains neurons that connect to the entire input volume.



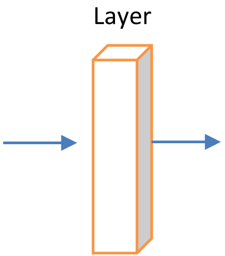
A conventional deep neural architecture for classification problem.

Batch Normalization

Idea: if the input is normalized, why not at all intermediate layers?

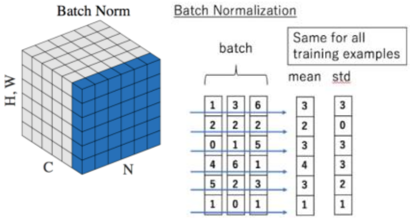


Batch Normalization



$$x \Rightarrow \hat{x} = \frac{x-\mu}{\sigma} \Rightarrow y = \gamma\hat{x} + \beta$$

μ: mean of x in mini-batch.
σ: std of x in mini-batch.
γ: scale, learnable.
β: shift, learnable.



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

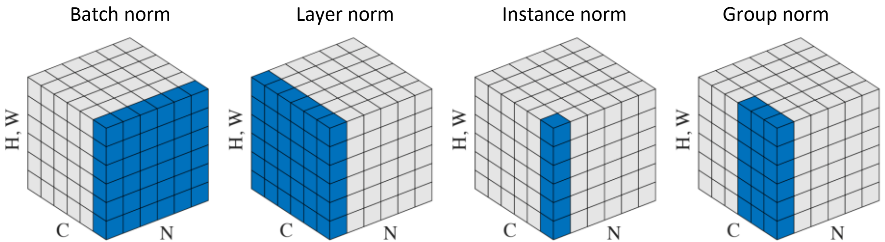
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

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

Normalization methods

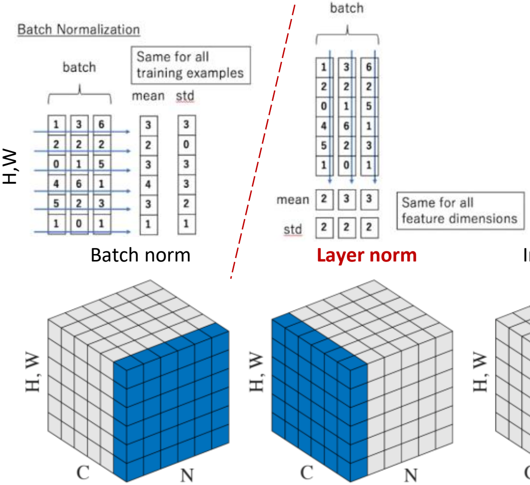
Various normalization layers:

- batch normalization
- layer normalization
- instance normalization
- group normalization



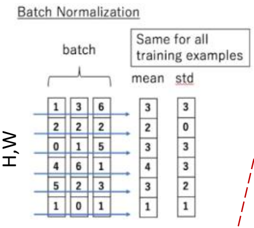
Normalization methods. Each subplot shows a feature map tensor, with **N** as the batch axis, **C** as the channel axis, and (**H**, **W**) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

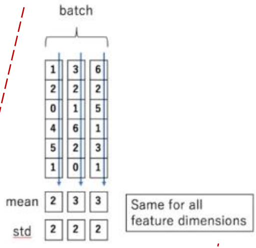
Normalization methods



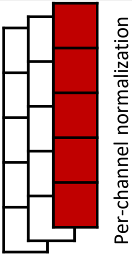
Layer normalization: normalizes the activations of the previous layer for each given data sample independently in a batch. This normalization is often used in RNN to stabilize the network convergence.

Normalization methods

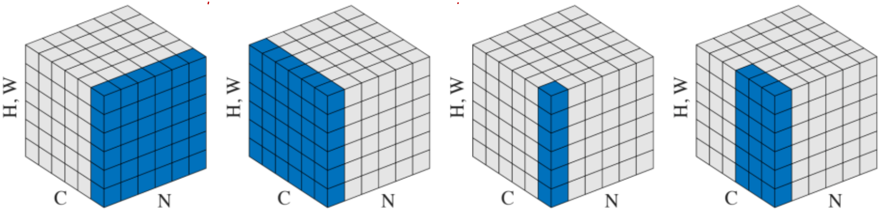




One sample
Channels

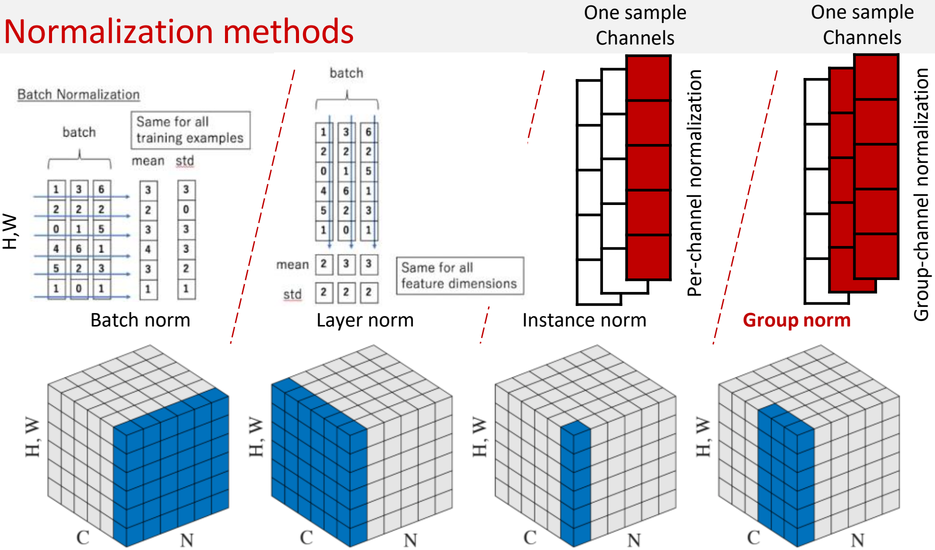


Batch norm | Layer norm | **Instance norm** | Group norm



Instance normalization: normalizes all features of each channel of a given example independently, to remove image-content instance-specific mean and covariance shift across samples in a data batch.

Normalization methods

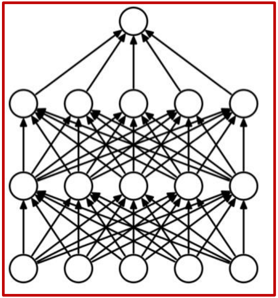


Group normalization: divides channels into groups and normalizes the features within each group. Group norm is a generalization of Instance norm where group size = 1. The motivation is features may span across a group of channels.

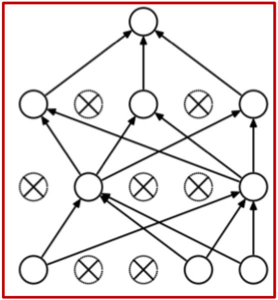
Dropout

What is **dropout**?

How does **dropout** work during **training** and during **testing**?



Standard neural network

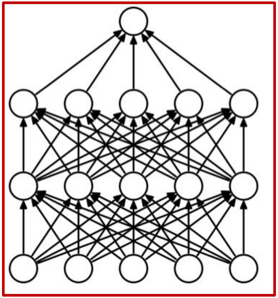


After applying dropout

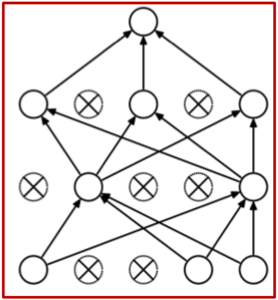
Dropout

What is dropout?

- **Randomly** “drop” the neurons.
- **Prevent overfitting** by reducing co-adaptation of neurons.
- Like training many **random sub-networks**.



Standard neural network



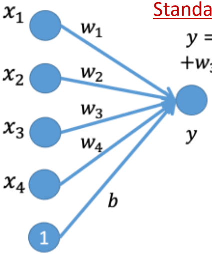
After applying dropout

https://arxiv.org/pdf/1207.0580.pdf

Dropout

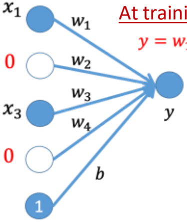
Dropout **during training**:

- For each neuron, randomly set its input to zero with probability p, for example p=0.5.



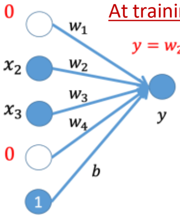
Standard neural network

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$



At training: with dropout

$$y = w_1x_1 + w_3x_3 + b$$



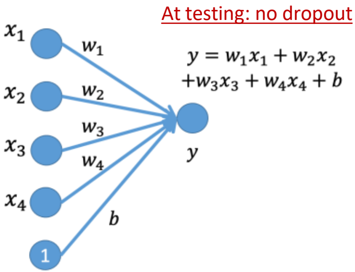
At training: with dropout

$$y = w_2x_2 + w_3x_3 + b$$

Dropout

Dropout **during testing**:

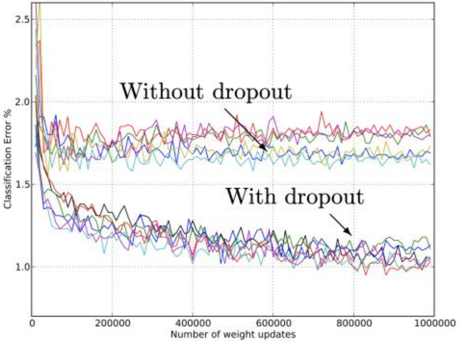
- **No dropout**, i.e., set all neurons with dropout probability p=0.
- Note: Dropouts can be used for neural network inference by dropping during predictions and predicting multiple times to get a distribution.



Dropout

Efficiency of dropout:

- Widely used and **highly effective**.
- Proposed as an alternative to ensemble methods, which is too expensive for neural networks.

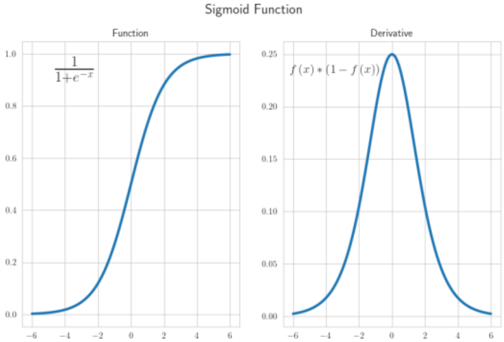


Test error for different architectures with and without dropout.

The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Activation functions - Sigmoid

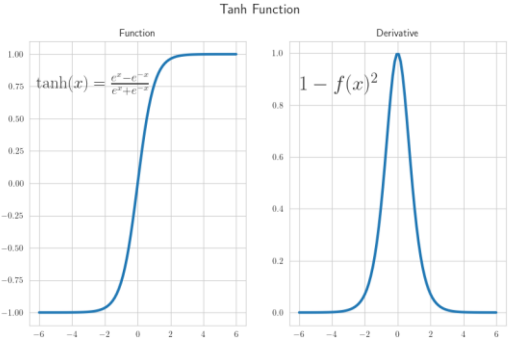


$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x) * (1 - f(x))$$

- Commonly used for models where we want to predict a probability as the function's output is between 0 and 1.
- Easy to cause vanishing gradient.

Activation functions - Tanh

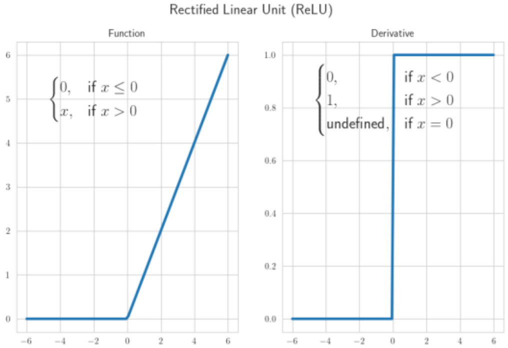


$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f'(x) = 1 - f(x)^2$$

- The range of the tanh function is between -1 and 1.
- Good replacement of sigmoid to avoid vanishing gradient.

Activation functions - ReLU

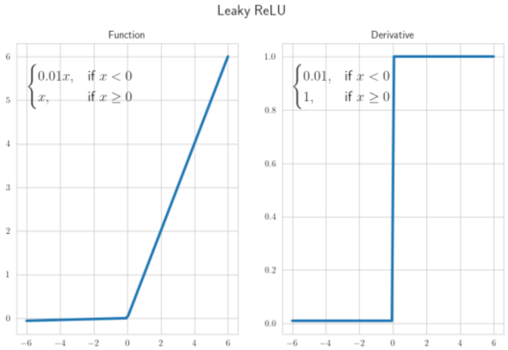


$$f(x) = \begin{cases} 0, \text{if } x \leq 0 \\ x, \text{if } x > 0 \end{cases}$$

$$f'(x) = \begin{cases} 0, \text{if } x < 0 \\ 1, \text{if } x > 0 \\ \text{undefined}, \text{if } x = 0 \end{cases}$$

- Quick and simple calculation. The value range is between 0 and $+\infty$.
- At $x = 0$, the function is not differentiable because its left and right derivative are not equal.
- Negative values become zero which decreases the quality of training process.

Activation functions – Leaky ReLU

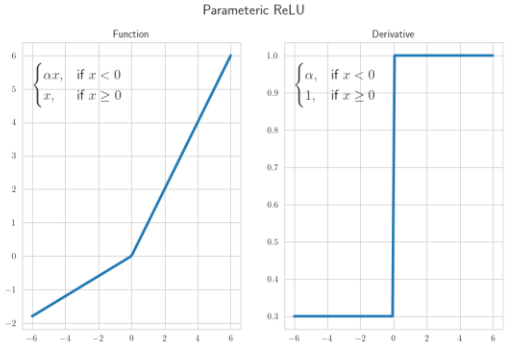


$$f(x) = \begin{cases} 0.01x, \text{if } x \leq 0 \\ x, \text{if } x > 0 \end{cases}$$

$$f'(x) = \begin{cases} 0.01, \text{if } x < 0 \\ 1, \text{if } x > 0 \\ \text{undefined}, \text{if } x = 0 \end{cases}$$

- A variant of ReLU. Quick and simple calculation. The value range is between $\pm$ $-\infty$ and $+\infty$. Usually, the parameter is often 0.01 or 0.1
- At at $x = 0$, the function is not differentiable because its left and right derivative are not equal.

Activation functions - Parametric ReLU

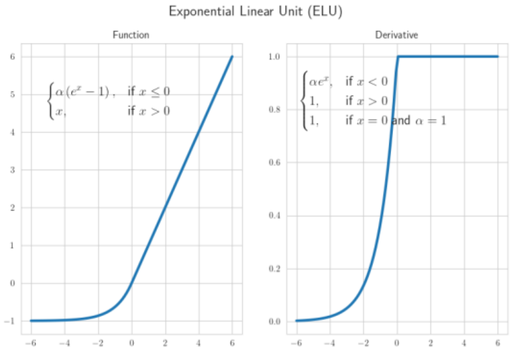


$$f(x) = \begin{cases} \alpha x, \text{if } x \leq 0 \\ x, \text{if } x > 0 \end{cases}$$

$$f'(x) = \begin{cases} \alpha, \text{if } x < 0 \\ 1, \text{if } x > 0 \\ \text{undefined}, \text{if } x = 0 \end{cases}$$

- A variant of Leaky ReLU with parameter α. The value range is between $-\infty$ and $+\infty$.
- At at $x = 0$, the the function is not differentiable because its left and right derivative are not equal.

Activation functions - ELU (Exponential Linear Unit)



$$f(x) = \begin{cases} \alpha(e^x - 1), \text{if } x \leq 0 \\ x, \text{if } x > 0 \end{cases}$$

$$f'(x) = \begin{cases} \alpha e^x, \text{if } x < 0 \\ 1, \text{if } x > 0 \\ 1, \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$$

- Continuous and differentiable at all points, with given condition at $x = 0$ and $\alpha = 1$.
- Using ELU often leads to a lower training time and a higher accuracy as compared to ReLU and its variants.

Activation functions - Others

Built-in activation functions in Tensorflow:

`elu(...)` : Exponential Linear Unit.

`exponential(...)` : Exponential activation function.

`gelu(...)` : Applies the Gaussian error linear unit (GELU) activation function.

`get(...)` : Returns function.

`hard_sigmoid(...)` : Hard sigmoid activation function.

`linear(...)` : Linear activation function (pass-through).

`relu(...)` : Applies the rectified linear unit activation function.

`selu(...)` : Scaled Exponential Linear Unit (SELU).

`serialize(...)` : Returns the string identifier of an activation function.

`sigmoid(...)` : Sigmoid activation function, `sigmoid(x) = 1 / (1 + exp(-x))`.

`softmax(...)` : Softmax converts a vector of values to a probability distribution.

`softplus(...)` : Softplus activation function, `softplus(x) = log(exp(x) + 1)`.

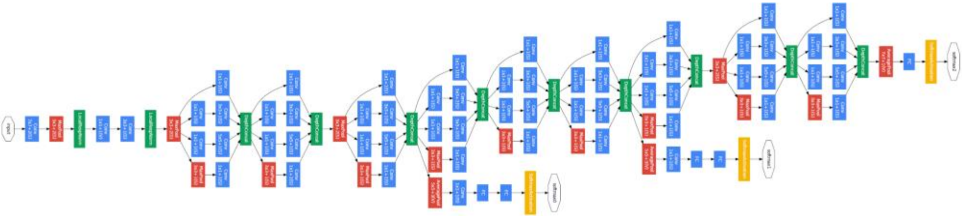
`softsign(...)` : Softsign activation function, `softsign(x) = x / (abs(x) + 1)`.

`swish(...)` : Swish activation function, `swish(x) = x * sigmoid(x)`.

`tanh(...)` : Hyperbolic tangent activation function.

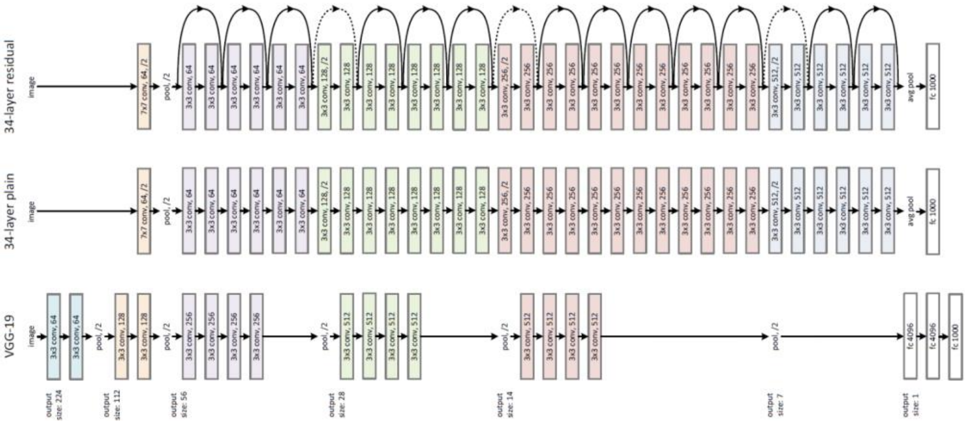
Reference: https://www.tensorflow.org/api_docs/python/tf/keras/activations

Neural Network Architectures



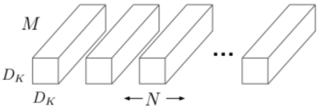
GoogLe Net (Inception v1)

Neural Network Architectures

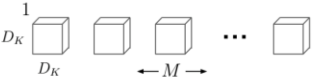


ResNet

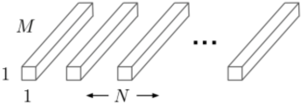
Neural Network Architectures



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

MobileNet v1

What if building N-layer Neural Network with N is large?

Summary

Fully connected layer.

Convolutional layer.

- 1x1 convolutional layer
- 1D, 2D, 3D convolutional layer.
- Transposed convolutional layer.
- Deconvolutional layer.

Normalization layer

- Batch normalization.
- Layer normalization.
- Instance normalization.
- Group normalization.

Pooling layer.

Dropout layer.

Activation functions.

Some simple Neural Network Architectures.

Q&A

Thank you