# Introduction to Deep Learning

## Deep Learning and Convolution

Alexandre Allauzen

ESPCI PARIS | PSL

Dauphine UNIVERSITÉ PARIS | PSL

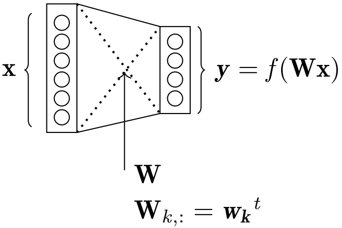
MILES Machine Intelligence and Learning Systems

Jan. 2025

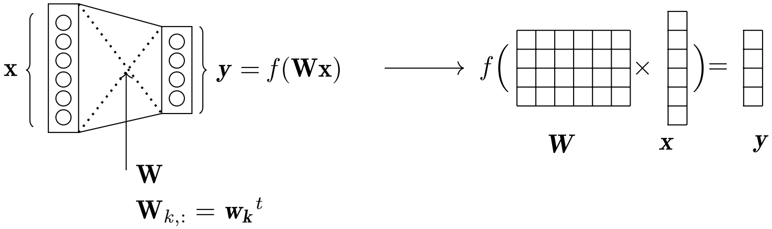
# Outline

# Outline

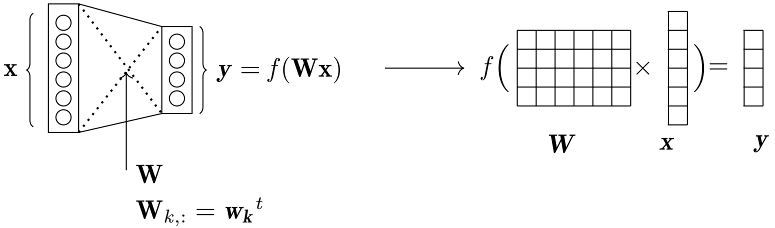
# Two layers fully connected: a linear separation



# Two layers fully connected: a linear separation



# Two layers fully connected: a linear separation




Activation $f$:

- $f$ is usually a non-linear function
- $f$ is a component wise function
- tanh, sigmoid, relu, ...

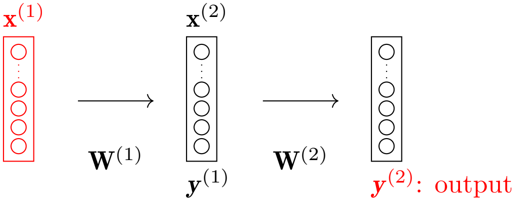
*e.g* the softmax function:

Dimensions:

- $\mathbf{x} : D \times 1$
- $\boldsymbol{W} : M \times D$
- $\boldsymbol{y} : (M \times \cancel{D) \times (D} \times 1) = M \times 1$

$$y_k = P(c = k|\mathbf{x}) = \frac{e^{\boldsymbol{w_k}^t \mathbf{x}}}{\sum_{k'} e^{\boldsymbol{w_{k'}}^t \mathbf{x}}} = \frac{e^{\mathbf{W}_{k,:}\mathbf{x}}}{\sum_{k'} e^{\mathbf{W}_{k',:}\mathbf{x}}}$$

# From linear to non-linear case




$$\boldsymbol{\theta} = (\mathbf{W}^{(1)}, \mathbf{W}^{(2)})$$

Trained by back-propagation of the gradient

## Universal approximation theorem

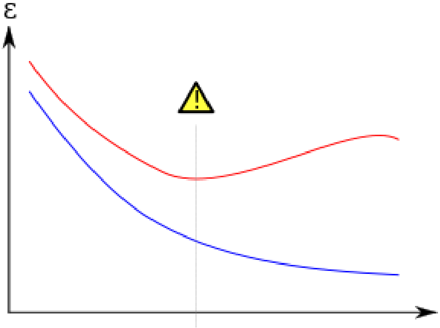
> *a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function. (...)*

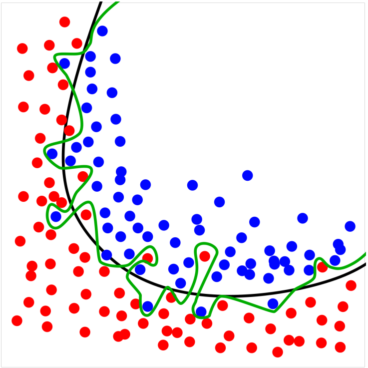
(Cybenko1989)

However, it does not touch upon the algorithmic learnability of those parameters.

# Overfitting

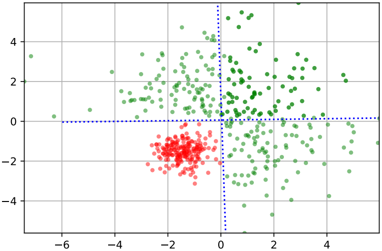
## The danger of the over-parametrization

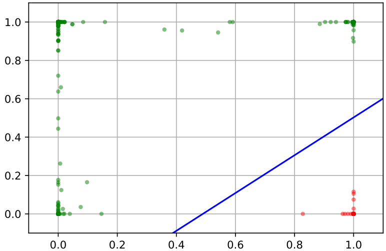


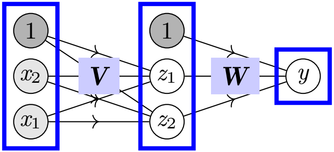



Source: Wikipedia

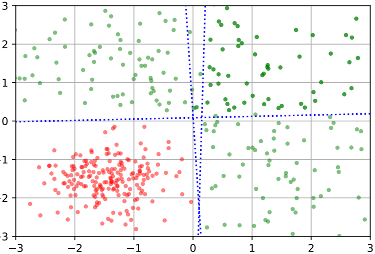
# A first example

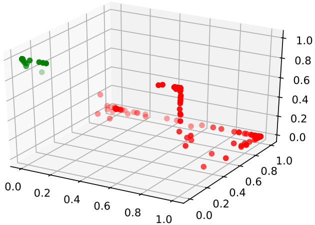


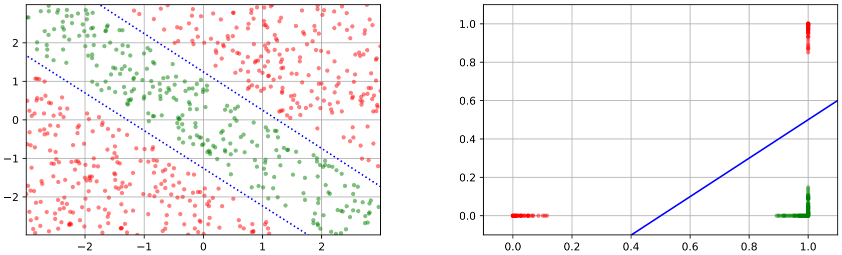



# A first example, with 3 hidden units

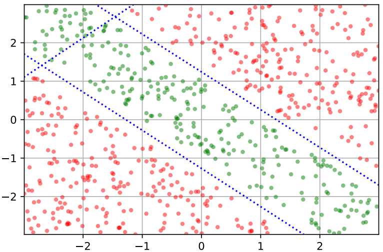


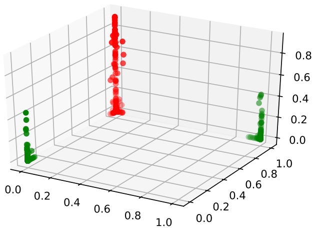



# The xor-like example

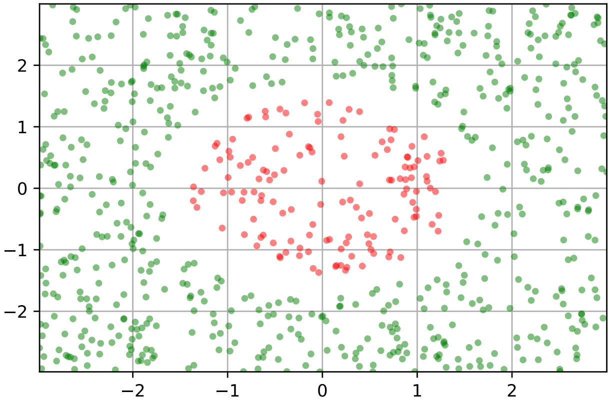


# The xor-like example, with 3 hidden units

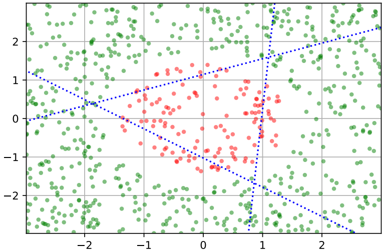


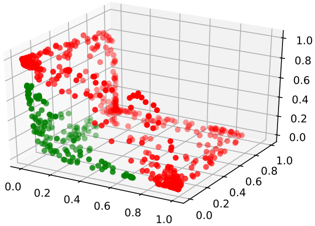



# A more difficult example

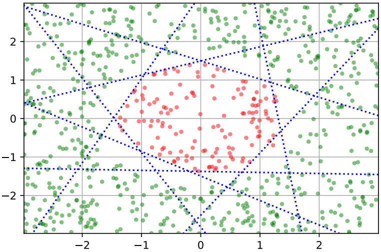


# A more difficult example, with 3 hidden units



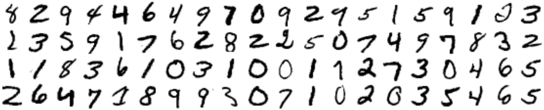
# A more difficult example, with 8 hidden units



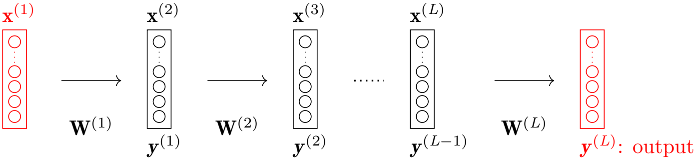
# Outline

# Experimental observations (MNIST task) - 1

## The MNIST database



## Comparison of different depth for feed-forward architecture

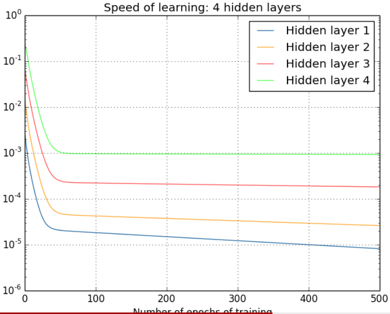



- Hidden layers have a sigmoid activation function.
- The output layer is a softmax.

# Experimental observations (MNIST task) - 2

## Varying the depth

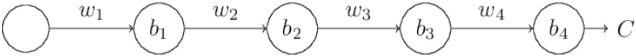
- Without hidden layer: $\approx 88\%$ accuracy
- 1 hidden layer (30): $\approx 96.5\%$ accuracy
- 2 hidden layers (30): $\approx 96.9\%$ accuracy
- 3 hidden layers (30): $\approx 96.5\%$ accuracy
- 4 hidden layers (30): $\approx 96.5\%$ accuracy




(From `http://neuralnetworksanddeeplearning.com/chap5.html`)

# Intuitive explanation

Let consider the simplest deep neural network, with just a single neuron in each layer.




$w_i, b_i$ are resp. the weight and bias of neuron $i$ and $C$ some cost function.

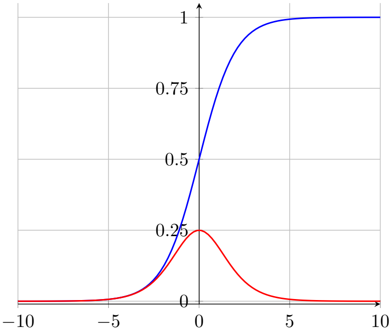
Compute the gradient of $C$ $w.r.t$ the bias $b_1$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y_4} \times \frac{\partial y_4}{\partial a_4} \times \frac{\partial a_4}{\partial y_3} \times \frac{\partial y_3}{\partial a_3} \times \frac{\partial a_3}{\partial y_2} \times \frac{\partial y_2}{\partial a_2} \times \frac{\partial a_2}{\partial y_1} \times \frac{\partial y_1}{\partial a_1} \times \frac{\partial a_1}{\partial b_1} \quad (1)$$

$$= \frac{\partial C}{\partial y_4} \times \sigma'(a_4) \times w_4 \times \sigma'(a_3) \times w_3 \times \sigma'(a_2) \times w_2 \times \sigma'(a_1) \quad (2)$$

# Intuitive explanation - 2

The derivative of the activation function: $\sigma'$




$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

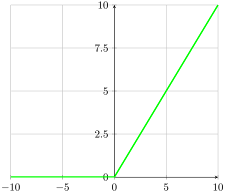
But weights are initialize around 0.

**The different layers in our deep network are learning at vastly different speeds:**

- when later layers in the network are learning well,
- early layers often get stuck during training, learning almost nothing at all.

# A first Solution

## Change the activation function (Rectified Linear Unit or ReLU)




- Avoid the vanishing gradient
- Some units can "die"

See (Glorot et al.2011) for more details

## Variants

- Leaky ReLU (Maas et al.2013)
- Soft-plus $log(1 + e^x)$

And many more, see `https://pytorch.org/docs/stable/nn.html`

## More details

See (Hochreiter et al.2001; Glorot and Bengio2010; LeCun et al.2012)

# A question

Why adding a layer can lower the performance ?

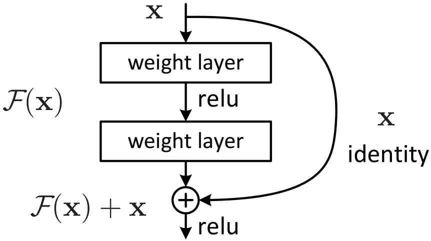
- Overfitting ? and what about the identity
- Vanishing gradient ? and with the *Relu* ?

## Residual block

From (He et al.2016)

- Add a skip connection
- The model learn the "residual"

$$\mathbf{y} = \mathbf{x} + \mathcal{F}(\mathbf{x})$$




A simple version of highway networks (Srivastava et al.2015)

# Residual block

### Forward

$$\mathbf{y} = \mathbf{x} + \mathcal{F}(\mathbf{x})$$

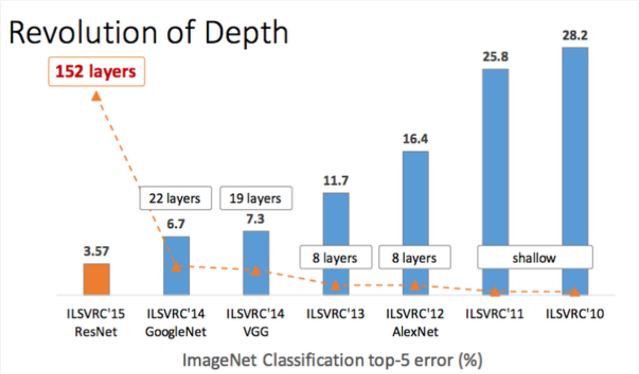
### Backward

Assume a residual block for the layer $l$ in the network. Training requires:

- $\frac{\partial l}{\partial \mathbf{W}^{(l)}}$ for the update of the layer
- $\frac{\partial l}{\partial \mathbf{x}^{(l)}}$ for the backpropagation

$$\begin{aligned}\frac{\partial l}{\partial \mathbf{x}^{(l)}} &= \frac{\partial l}{\partial \mathbf{y}^{(l)}} \times \frac{\partial \mathbf{y}^{(l)}}{\partial \mathbf{x}^{(l)}} \\ &= \frac{\partial l}{\partial \mathbf{y}^{(l)}} \times (1 + \frac{\partial \mathcal{F}(\mathbf{x}^{(l)})}{\partial \mathbf{x}^{(l)}})\end{aligned}$$

# ResNet in action




http://kaiminghe.com/

# Outline

# Regularization $l^2$ or gaussian prior or weight decay

The basic way:

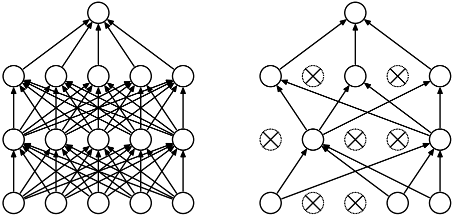
$$\mathcal{L}(\boldsymbol{\theta};\mathcal{D}) = \sum_{i=1}^{N} l(\boldsymbol{\theta}, \mathbf{x}_{(i)}, c_{(i)}) + \frac{\lambda}{2}||\boldsymbol{\theta}||^2$$

- The second term is the regularization term.
- Each parameter has a gaussian prior : $\mathcal{N}(0, 1/\lambda)$.
- $\lambda$ is a hyperparameter.
- The update has the form:

$$\boldsymbol{\theta} = (1 - \eta_t\lambda)\boldsymbol{\theta} - \eta_t\nabla_{\boldsymbol{\theta}}$$

# Dropout

## A new regularization scheme (Srivastava and Salakhutdinov2014)



(a) Standard Neural Net (b) After applying dropout.

- For each training example: randomly turn-off the neurons of hidden units (with $p = 0.5$)
- At test time, use each neuron scaled down by $p$

- Dropout serves to separate effects from strongly correlated features and
- prevents co-adaptation between units
- It can be seen as averaging different models that share parameters.
- It acts as a powerful regularization scheme.

# Dropout - implementation

The layer should keep:

- $\mathbf{W}^{(l)}$: the parameters
- $f^{(l)}$: its activation function
- $\mathbf{x}^{(l)}$: its input
- $\boldsymbol{a}^{(l)}$: its pre-activation associated to the input
- $\boldsymbol{\delta}^{(l)}$: for the update and the back-propagation to the layer $l-1$
- $\boldsymbol{m}^{(l)}$: the dropout mask, to be applied on $\mathbf{x}^{(l)}$

**Forward pass**

For $l = 1$ to $(L-1)$

- Compute $\boldsymbol{y}^{(l)} = f^{(l)}(\mathbf{W}^{(l)}\mathbf{x}^{(l)})$
- $\mathbf{x}^{(l+1)} = \boldsymbol{y}^{(l)} = \boldsymbol{y}^{(l)} \circ \boldsymbol{m}^{(l)}$

$\boldsymbol{y}^{(L)} = f^{(L)}(\mathbf{W}^{(L)}\mathbf{x}^{(L)})$

# Internal Covariate Shift

### After init.: training

For a layer $l$,

- The update of $\mathbf{W}^{(l)}$ depends on the incoming gradient (from the loss)
- And a mini-batch of $\mathbf{X}^{(l)} = (\mathbf{x}^{(l)})$, but the statistics $\mathbf{X}^{(l)}$ depends on $\mathbf{W}^{(l-1)}$
- During the back-propagation, you then update $\mathbf{W}^{(l-1)}$.
- During the next forward-backward pass, the statistics of $\mathbf{X}^{(l)}$ have changed !
- It could have the same effect as a bad init. !

### Sketch of solution

- Stabilize the distribution of each $\mathbf{X}^{(l)}$.
- By learning a layer that make each dimension zero-mean unit-variance.

# Batch normalization

Assume a mini batch of dim. $(B, D)$

### Compute statistics and normalization

$$\mu_j = \frac{1}{B}\sum_{i=1}^{B} x_{i,j} \quad \text{and } \sigma_j^2 = \frac{1}{B}\sum_{i=1}^{B}(x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Record also running average and variance for the test time.

### Shift and scale

$$x_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

With $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are the learnable vectors of the batch-norm layer.

See (Ioffe and Szegedy2015)

# How to apply batch norm

## Where to insert the batch-norm

$$\mathbf{W}_1 \to BN_1 \to ReLU \to \mathbf{W}_2 \to BN_2 \to ReLU \to \cdots$$

but this one is better:

$$\mathbf{W}_1 \to ReLU \to BN_1 \to \mathbf{W}_2 \to ReLU \to BN_2 \to \cdots$$

## Drawbacks

- The batch size must be large enough to capture overall statistics, which is sometimes impossible if you are working with large models.
- The concept of a batch is not always present, or it may change from time to time.
- Not the same be behaviour in train and test

# Layer Norm

Independant of the batch size with no difference between train and test.

**Compute statistics and normalization per dimension**

$$\mu_i = \frac{1}{D}\sum_{j=1}^{D} x_{i,j} \quad \text{and } \sigma_i^2 = \frac{1}{D}\sum_{j=1}^{D}(x_{i,j} - \mu_i)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$$x_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

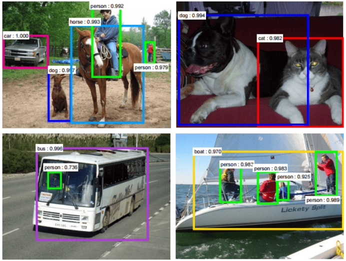
With $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are the learnable vectors of the batch-norm layer.

Works well also with more advanced architectures (recurrent and transformers)

See (Ba et al.2016)

# Outline

# Convolution for image processing




# Image classification

An image = (2D array of values) $\times$ (number of channels)

### 2D array

The spatial structure:

- A 2D real space
- With distance

### Channels

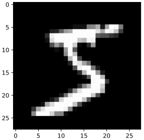
- For image : R,G,B
- In fluid mechanics: pression, velocity, ...
- In general: different measures on the same spatial domain

### Sources

Many figures and examples are inspired by, or extracted from the course of the Stanford course of Fei-Fei Li.

`http://cs231n.stanford.edu/`

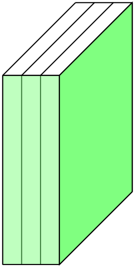
# Image classification: the flatten way




$\rightarrow$

$\mathbf{x} \in \mathbb{R}^{784} \longrightarrow c \in \{0, 1, 2, \ldots, 9\}$

Another example with a color image (3 channels) of $32 \times 32$ pixels



$\rightarrow$

$\mathbf{x} \in \mathbb{R}^{3072} \longrightarrow c \in \{0, 1, 2, \ldots, 9\}$

# Feed-forward NNet for image classification

## Question ?

- Assume a NNet for images of size $256 \times 256$ in color,
- with one hidden layer of 1000 and an output layer for 100 classes

$\rightarrow$ How many parameters do we have ?

$\rightarrow$ And for a larger image like a $16:9$ resolution $(1920 \times 1080)$ ?

## Intuition for convolution

- Look at a small region (window)
- Apply a (small) filter
- Translation invariance by sliding the window along the image
- Parameters sharing

# Convolution 2D - the first step

## Extract a frame, or a window, and apply a “filter”

The input image ($L = 6$, $C = 6$):
and the window

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

Apply the filter:
kernel size of $ks = (3, 3)$

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

### The output value (output channel)

$$\text{First value: } h_{1,1} = \sum_{i,j} w_{i,j} \times x_{i,j}$$

# Convolution 2D - slide the window

"Scan" the image with the filter

The **next** window

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

Apply the **same** filter with a **stride** $(1,1)$

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

The output value (output channel)

$$\text{Second value:}\quad h_{1,2} = \sum_{i,j} w_{i,j} \times x_{i,j}$$

$$\text{In general:}\quad h_{l,c} = \sum_{i,j=(0,0)}^{(I,J)} w_{i,j} \times x_{l+i,c+j}$$

# 2D Convolution: animated version (1st row)

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

| | | | |
|---|---|---|---|
| $\mathbf{h}_{1,1}$ | $h_{1,2}$ | $h_{1,3}$ | $h_{1,4}$ |

# 2D Convolution: animated version (1st row)

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

| | | | |
|---|---|---|---|
| $h_{1,1}$ | $\mathbf{h_{1,2}}$ | $h_{1,3}$ | $h_{1,4}$ |

# 2D Convolution: animated version (1st row)

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

| | | | |
|---|---|---|---|
| $h_{1,1}$ | $h_{1,2}$ | $\boxed{\mathbf{h_{1,3}}}$ | $h_{1,4}$ |

# 2D Convolution: animated version (1st row)

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

$h_{1,1}$ $h_{1,2}$ $h_{1,3}$ $\mathbf{h_{1,4}}$

# 2D Convolution: animated version (2nd row)

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

| | | | |
|---|---|---|---|
| $h_{1,1}$ | $h_{1,2}$ | $h_{1,3}$ | $h_{1,4}$ |
| $\mathbf{h_{2,1}}$ | $h_{2,2}$ | $h_{2,3}$ | $h_{2,4}$ |

# 2D Convolution: animated version (2nd row)

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

| | | | |
|---|---|---|---|
| $h_{1,1}$ | $h_{1,2}$ | $h_{1,3}$ | $h_{1,4}$ |
| $h_{2,1}$ | $\boxed{\mathbf{h_{2,2}}}$ | $h_{2,3}$ | $h_{2,4}$ |

# 2D Convolution: animated version (2nd row)

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

| | | | |
|---|---|---|---|
| $h_{1,1}$ | $h_{1,2}$ | $h_{1,3}$ | $h_{1,4}$ |
| $h_{2,1}$ | $h_{2,2}$ | $\mathbf{h_{2,3}}$ | $h_{2,4}$ |

# 2D Convolution: animated version (2nd row)

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

| | | | |
|---|---|---|---|
| $h_{1,1}$ | $h_{1,2}$ | $h_{1,3}$ | $h_{1,4}$ |
| $h_{2,1}$ | $h_{2,2}$ | $h_{2,3}$ | $\mathbf{h_{2,4}}$ |

# 2D Convolution: the final result

With a stride of $(1, 1)$

The input image:

| | | | | | |
|---|---|---|---|---|---|
| $x_{1,1}$ | $x_{2,1}$ | $x_{3,1}$ | $x_{4,1}$ | $x_{5,1}$ | $x_{6,1}$ |
| $x_{1,2}$ | $x_{2,2}$ | $x_{3,2}$ | $x_{4,2}$ | $x_{5,2}$ | $x_{6,2}$ |
| $x_{1,3}$ | $x_{2,3}$ | $x_{3,3}$ | $x_{4,3}$ | $x_{5,3}$ | $x_{6,3}$ |
| $x_{1,4}$ | $x_{2,4}$ | $x_{3,4}$ | $x_{4,4}$ | $x_{5,4}$ | $x_{6,4}$ |
| $x_{1,5}$ | $x_{2,5}$ | $x_{3,5}$ | $x_{4,5}$ | $x_{5,5}$ | $x_{6,5}$ |
| $x_{1,6}$ | $x_{2,6}$ | $x_{3,6}$ | $x_{4,6}$ | $x_{5,6}$ | $x_{6,6}$ |

The filter:

| | | |
|---|---|---|
| $w_{1,1}$ | $w_{1,2}$ | $w_{1,3}$ |
| $w_{2,1}$ | $w_{2,2}$ | $w_{2,3}$ |
| $w_{3,1}$ | $w_{3,2}$ | $w_{3,3}$ |

$\downarrow$

| | | | |
|---|---|---|---|
| $h_{1,1}$ | $h_{2,1}$ | $h_{3,1}$ | $h_{4,1}$ |
| $h_{1,2}$ | $h_{2,2}$ | $h_{3,2}$ | $h_{4,2}$ |
| $h_{1,3}$ | $h_{2,3}$ | $h_{3,3}$ | $h_{4,3}$ |
| $h_{1,4}$ | $h_{2,4}$ | $h_{3,4}$ | $h_{4,4}$ |

# Exercise

### Stride

- With an input image of length $(L, C)$, a kernel size of $ks$ and a stride $= (1, 1)$, what is the output dimension ?
- And with a stride $= (2, 2)$, what is the output dimension ?
- With a stride of $s$ ?

### Side effect

- The first and last rows and columns are not processed as the others. How to correct this aspects ?
- How to ensure the same dimension in output (assuming a stride of 1) ?
- How to ensure that every inputs are "seen" equally ?

# Padding

### Constant padding of $(1,2)$ with $c$

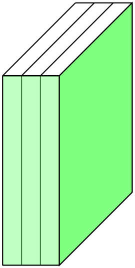
$$\begin{matrix} 1 & 4 & 3 \\ 0 & 6 & 5 \\ 2 & 7 & 9 \end{matrix} \longrightarrow \begin{matrix} c & c & c & c & c & c & c \\ c & c & 1 & 4 & 3 & c & c \\ c & c & 0 & 6 & 5 & c & c \\ c & c & 2 & 7 & 9 & c & c \\ c & c & c & c & c & c & c \end{matrix}$$

### Reflection padding of $(1,2)$

$$\begin{matrix} 1 & 4 & 3 \\ 0 & 6 & 5 \\ 2 & 7 & 9 \end{matrix} \longrightarrow \begin{matrix} 5 & 6 & 0 & 6 & 5 & 6 & ? \\ 3 & 4 & 1 & 4 & 3 & 4 & 1 \\ 5 & 6 & 0 & 6 & 5 & 6 & 0 \\ 9 & 7 & 2 & 7 & 9 & 7 & 2 \\ 5 & 6 & 0 & 6 & 5 & 6 & ? \end{matrix}$$

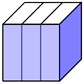
# Convolution in 2D

## The basics



$3 \times 32 \times 32$ image

$\longrightarrow$



$3 \times 5 \times 5$ filter

Convolution of the filter with the image:

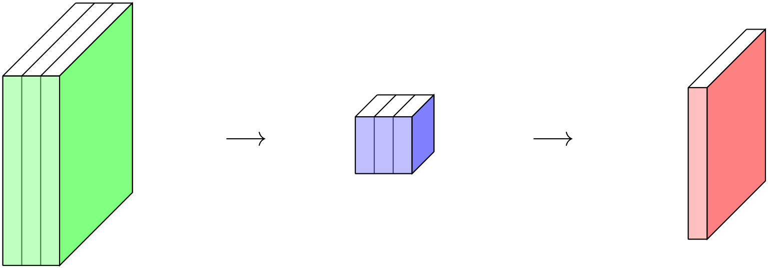
- Sliding the filter along the two axis
- Computing the "dot product" at each step

→ preserve the spatial structure along the channels

→ each step extract a "local" and spatial feature

# Convolution in 2D

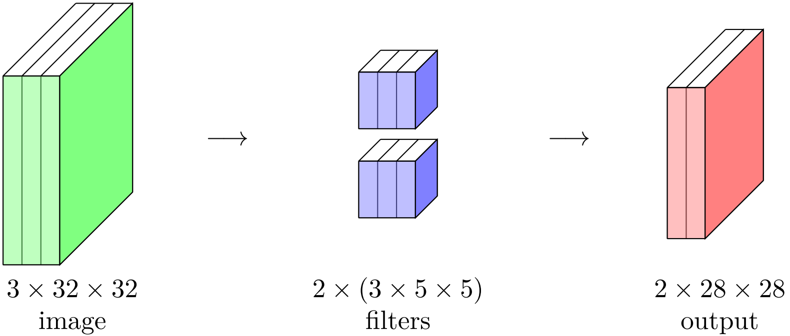
## With a single output channel



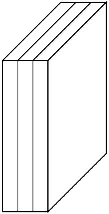
$3 \times 32 \times 32$ image $\quad$ $3 \times 5 \times 5$ filter $\quad$ $1 \times 28 \times 28$ output

# Convolution in 2D

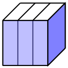
## Add a second output channel



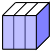
# Shape of things



- An image is $(C_{in}, L, C)$

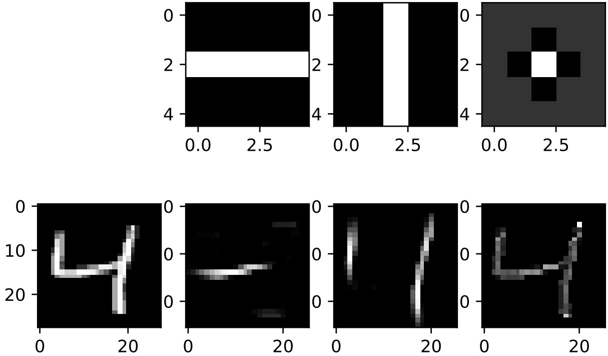


- A convolution filter is $(C_{in}, W, V)$

- A convolution filter with $(C_{out}, C_{in}, W, V)$ $C_{out} \times$ (  )

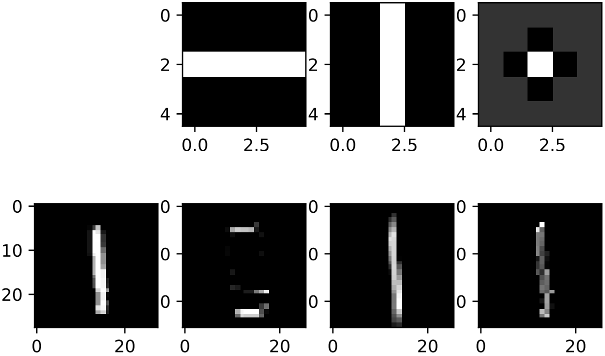
# Motivation for convolution

## Extract “low level” features



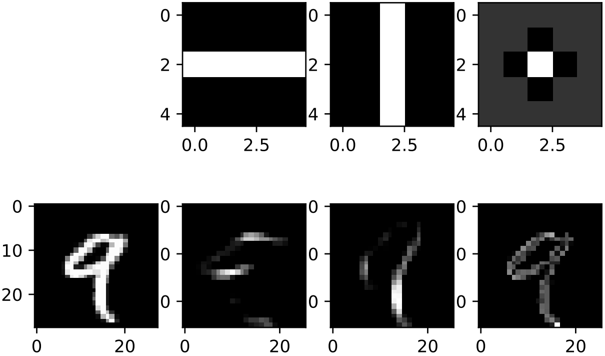
# Motivation for convolution

## Extract “low level” features



# Motivation for convolution

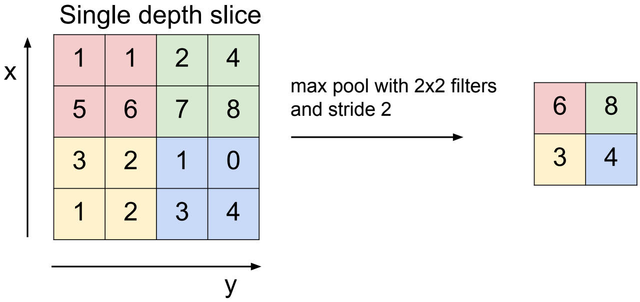
## Extract “low level” features



# Max-pooling or Downsampling in 2D

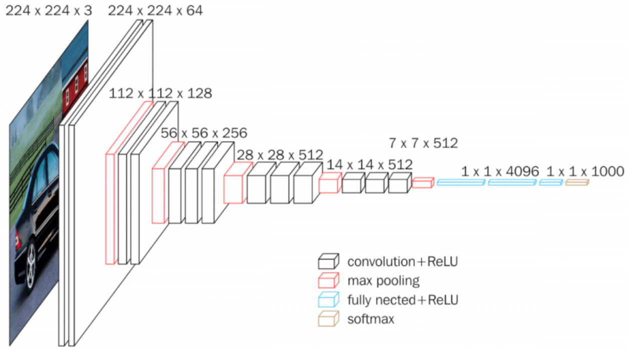
## The goal

- Convolution extracts local features (followed by non-linearity)
- Max-pooling acts as a selection, compression, or contraction operator
- The back-propagation promote feature saliency for each channel



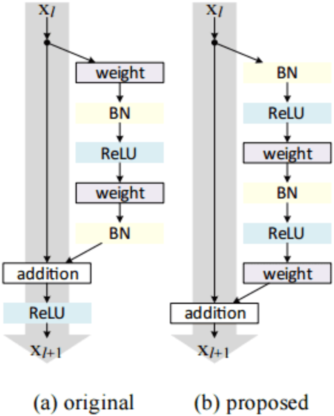
# Architecture of deep convolution NNet for image processing

## VGGNet (Simonyan and Zisserman2015)




Conv. layers with kernel size of $3 \times 3$, stride and padding, followed by pooling layers (max on $2 \times 2$ with stride 2).

# ResNet for Imagenet



Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton.
2016.
Layer normalization.

G. Cybenko.
1989.
Approximation by superpositions of a sigmoidal function.
*Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December.

Xavier Glorot and Yoshua Bengio.
2010.
Understanding the difficulty of training deep feedforward neural networks.
In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 249–256, May.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio.
2011.
Deep sparse rectifier neural networks.
In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
2016.

Deep residual learning for image recognition.
In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber.
2001.
Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.
In Kremer and Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press.

Sergey Ioffe and Christian Szegedy.
2015.
Batch normalization: Accelerating deep network training by reducing internal covariate shift.
In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul. PMLR.

Yann LeCun, Léon Bottou, Genevieve Orr, and Klaus-Robert Müller.
2012.
Efficient backprop.
In Grégoire Montavon, GenevièveB. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer Berlin Heidelberg.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng.

2013.
Rectifier nonlinearities improve neural network acoustic models.
In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing.*

Karen Simonyan and Andrew Zisserman.
2015.
Very deep convolutional networks for large-scale image recognition.
In *International Conference on Learning Representations.*

Nitish Srivastava and Ruslan Salakhutdinov.
2014.
Multimodal learning with deep boltzmann machines.
*Journal of Machine Learning Research*, 15:2949–2980.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber.
2015.
Training very deep networks.
*CoRR*, abs/1507.06228.