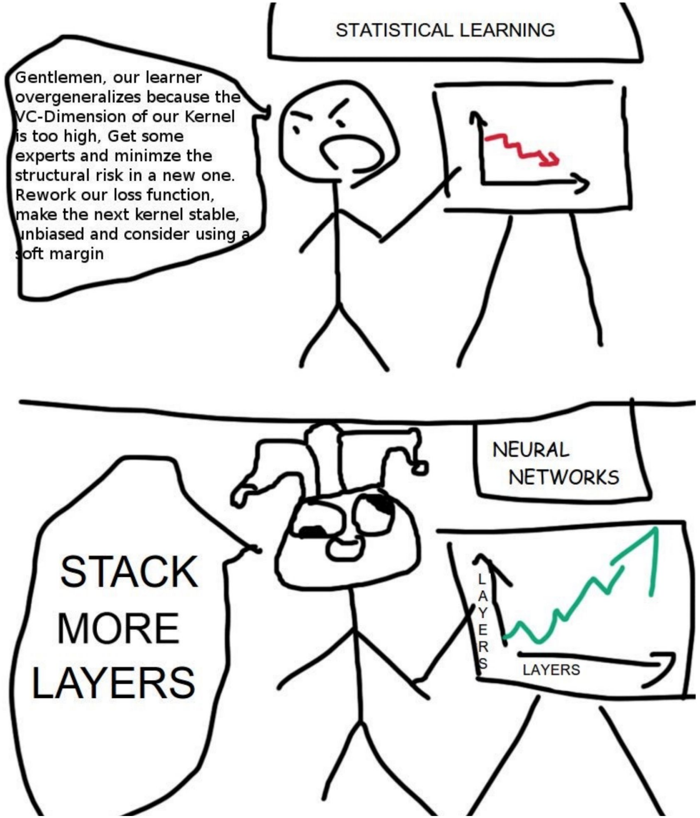


Neural Networks

Overview

Plan

1. Linear regression reminder
2. Fully Connected Neural Networks
3. History of Convolutional Neural Networks.

Usual approach

Small reminder about Linear Regression

$$\hat{y} = w_1 x + w_0$$

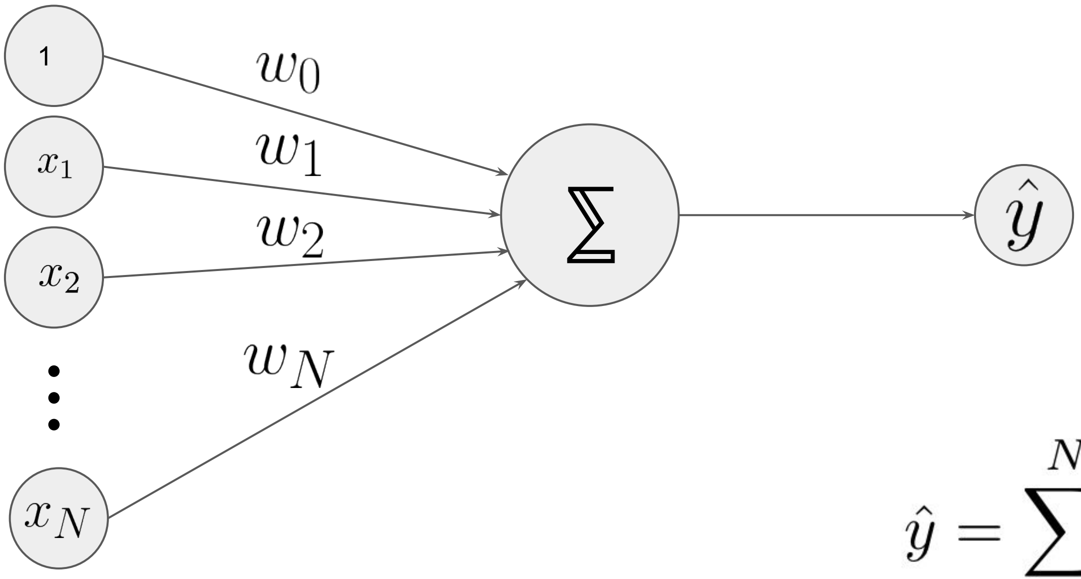
x – input

y – target

w – weights

Single Layer Perceptron

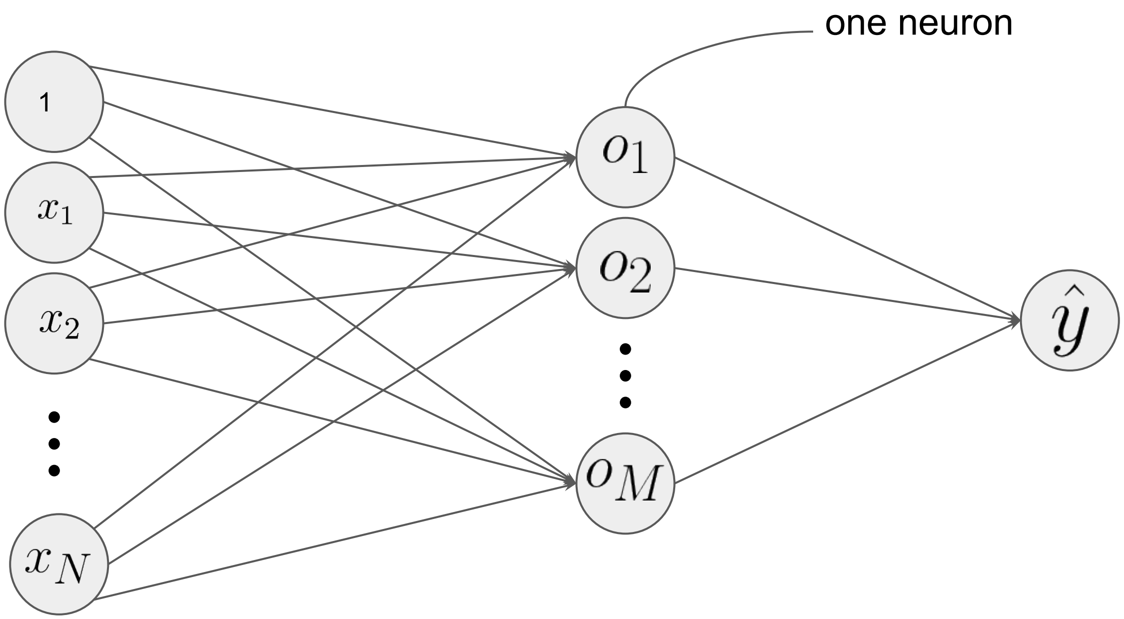
The same formula in different view



$$\hat{y} = \sum_{i=1}^{N} x_i w_i + w_0$$

Fully Connected Neural Network

What if we stack them?



$$o_1 = \sum_{i=2}^{N} x_i w_{i1}^{(1)} + w_{11}^{(1)}$$

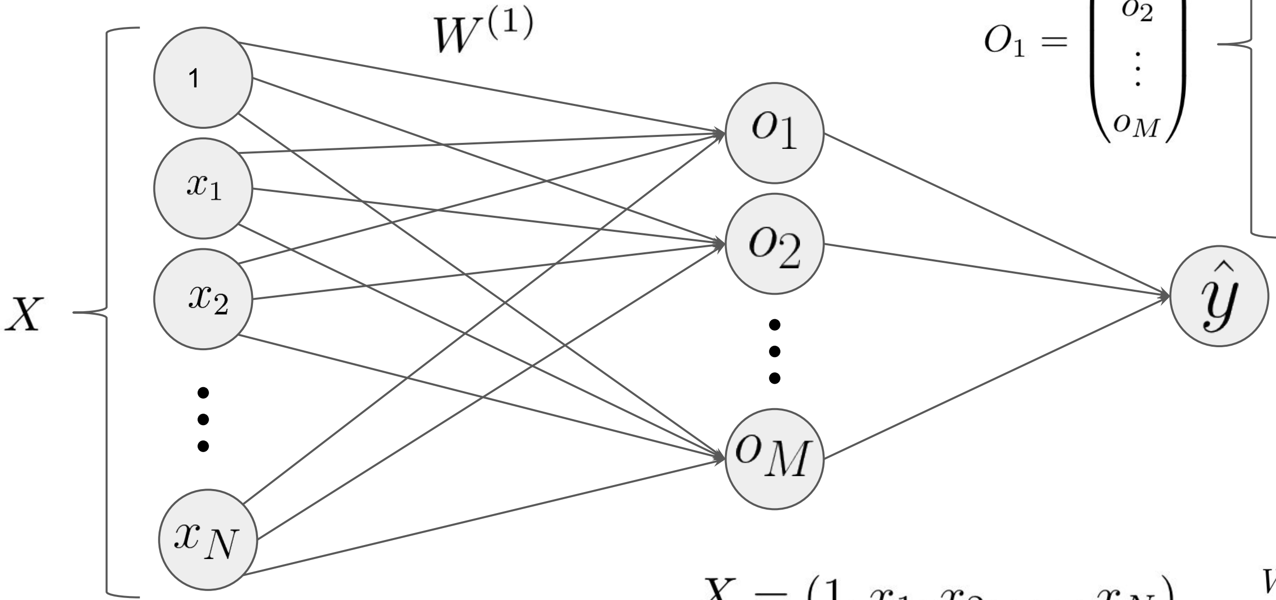
$$o_2 = \sum_{i=2}^{N} x_i w_{i2}^{(1)} + w_{12}^{(1)}$$

$$\vdots$$

$$o_M = \sum_{i=2}^{N} x_i w_{iM}^{(1)} + w_{1M}^{(1)}$$

Fully Connected Neural Network

What if we stack them?



$$X = (1, x_1, x_2, \dots, x_N)$$

$$O_1 = \begin{pmatrix} o_1 \\ o_2 \\ \vdots \\ o_M \end{pmatrix}$$

$$\begin{cases} o_1 = \sum_{i=2}^{N} x_i w_{i1}^{(1)} + w_{11}^{(1)} \\ o_2 = \sum_{i=2}^{N} x_i w_{i2}^{(1)} + w_{12}^{(1)} \\ \vdots \\ o_M = \sum_{i=2}^{N} x_i w_{iM}^{(1)} + w_{1M}^{(1)} \end{cases}$$

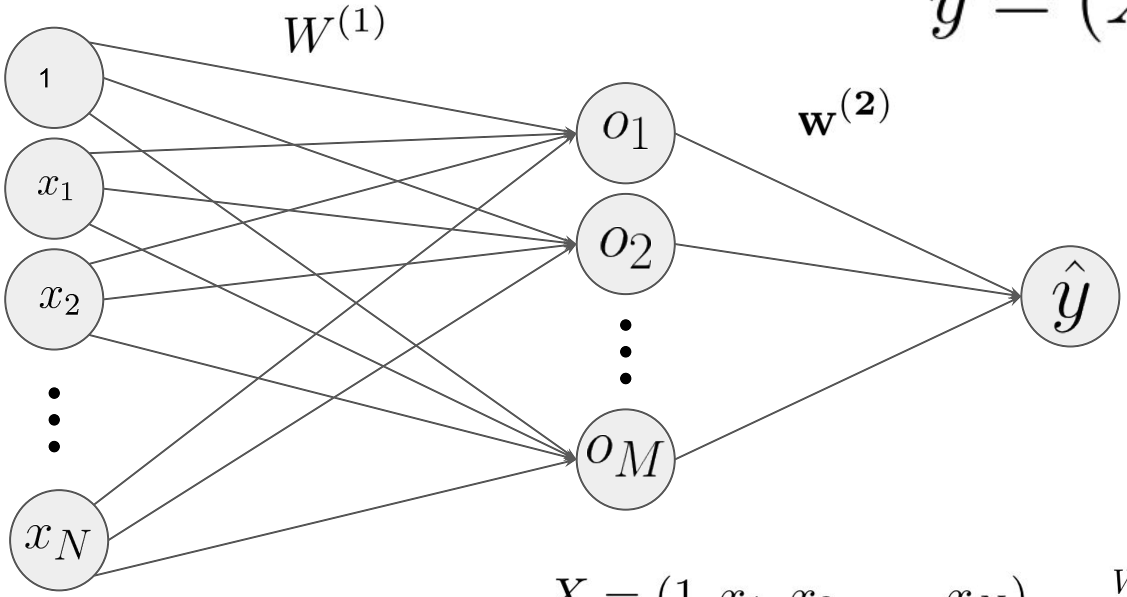
$$O_1 = X \times W^{(1)}$$

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \dots & w_{1M}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \dots & w_{2M}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1}^{(1)} & w_{N2}^{(1)} & \dots & w_{NM}^{(1)} \end{pmatrix}$$

Fully Connected Neural Network

What if we stack them?

$$\hat{y} = (X \times W^{(1)})\mathbf{w}^{(2)}$$



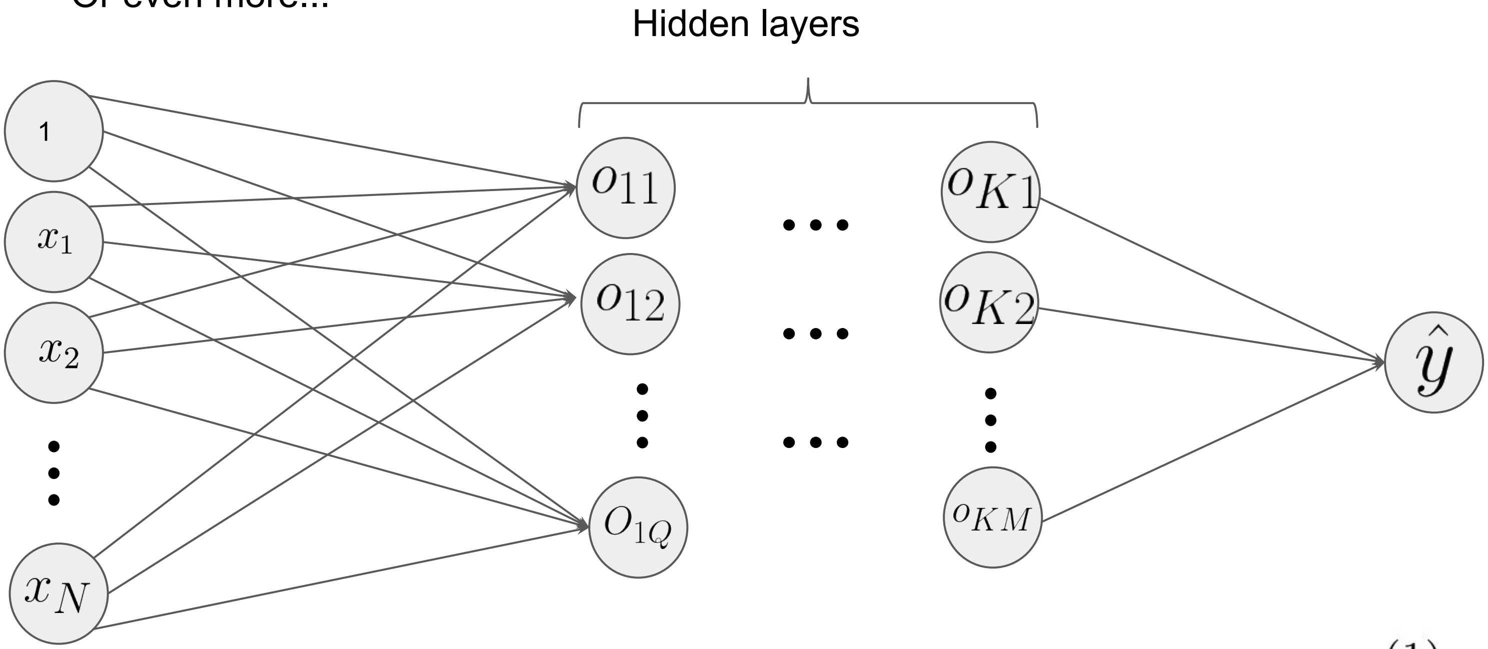
$$X = (1, x_1, x_2, \ldots, x_N)$$

$$\mathbf{w}^{(2)} = \begin{pmatrix} w_1^{(2)} \\ w_2^{(2)} \\ \vdots \\ w_M^{(2)} \end{pmatrix}$$

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \cdots & w_{1M}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \cdots & w_{2M}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1}^{(1)} & w_{N2}^{(1)} & \cdots & w_{NM}^{(1)} \end{pmatrix}$$

Fully Connected Neural Network

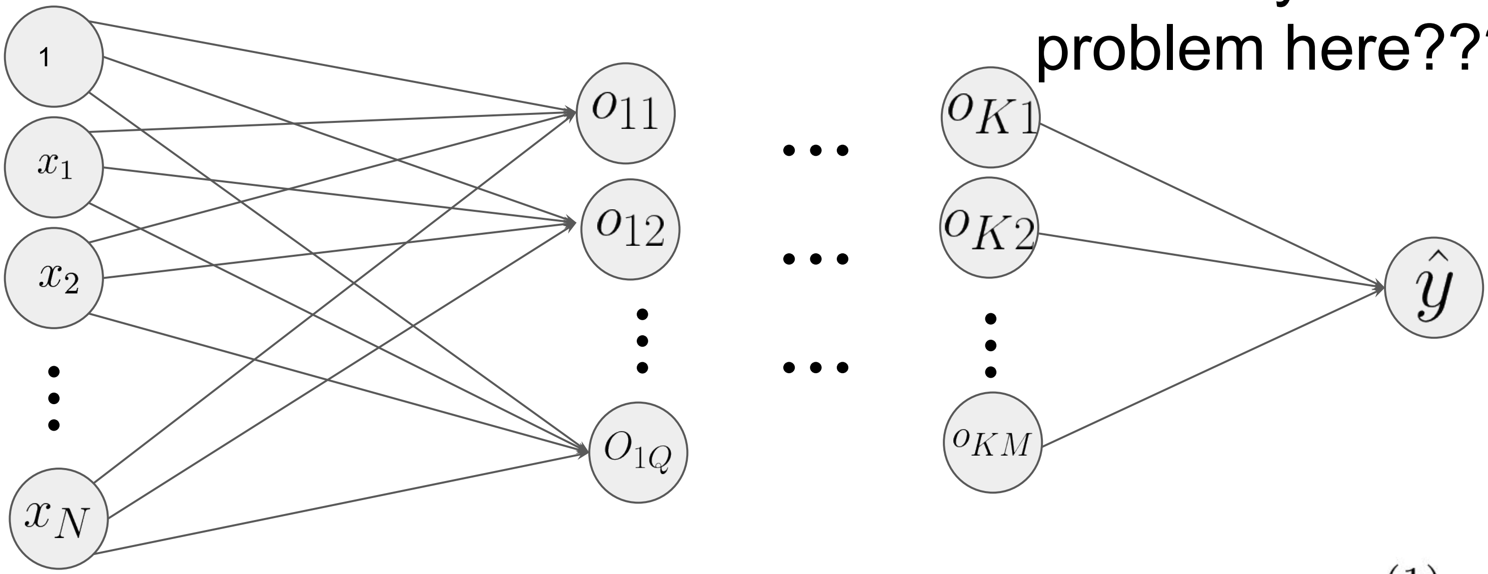
Or even more...



$$\hat{y} = (\dots((X \times W^{(1)})W^{(2)})\dots)\mathbf{w}^{(\mathbf{M})}$$

Fully Connected Neural Network

Or even more...



Does anyone see the problem here????

$$\hat{y} = (\ldots((X \times W^{(1)})W^{(2)})\ldots)\mathbf{w}^{(\mathbf{M})}$$

Fully Connected Neural Network

This function is still linear...

$$\hat{y} = (\ldots((X \times W^{(1)})W^{(2)})\ldots)\mathbf{w}^{(\mathbf{M})} = (X \times W^{(q)})\mathbf{w}^{(\mathbf{M})}$$

Fully Connected Neural Network

This function is still linear...

$$\hat{y} = (\ldots((X \times W^{(1)})W^{(2)})\ldots)\mathbf{w}^{(\mathbf{M})} = (X \times W^{(q)})\mathbf{w}^{(\mathbf{M})}$$

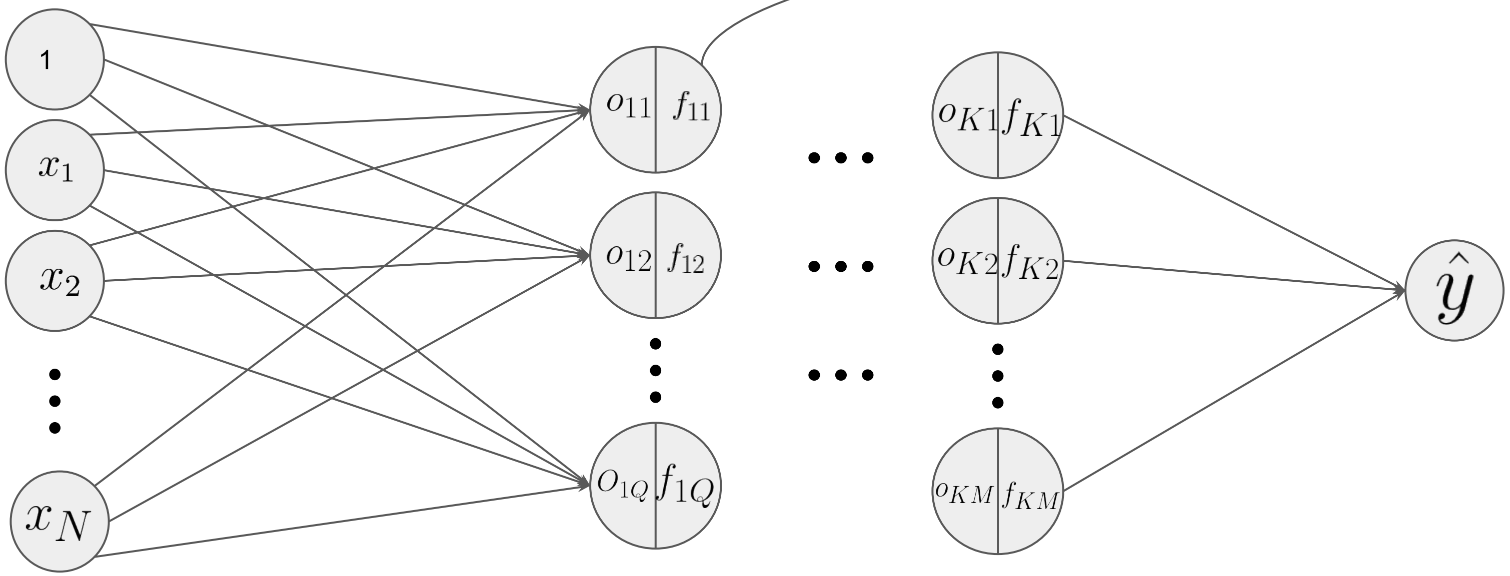
Let’s add some nonlinearity here!

$$\hat{y} = f_M(f_{M-1}(\ldots f_2(f_1(X \times W^{(1)})W^{(2)})\ldots)\mathbf{w}^{(\mathbf{M})})$$

Fully Connected Neural Network

Neuron output:

$$z_{11} = f_{11}(o_{11})$$

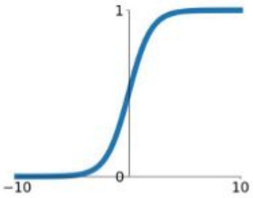


$$\hat{y} = f_M(f_{M-1}(\ldots f_2(f_1(X \times W^{(1)})W^{(2)})\ldots)\mathbf{w}^{(\mathbf{M})})$$

Activation functions

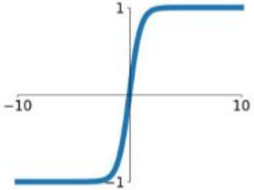
Sigmoid

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



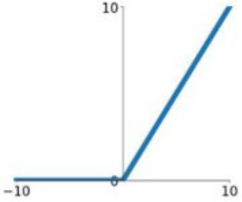
tanh

$\tanh(x)$



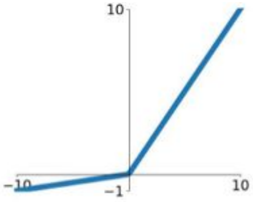
ReLU

$\max(0, x)$



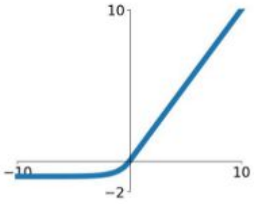
Leaky ReLU

$\max(0.1x, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Example: Regression task

How much the house costs?

Data(X):

1500 examples

ID	Number of rooms	Floor	LotArea	...	Distance from center (m)
1	2	5	64	...	3620
2	3	3	80	...	1076
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
N	2	10	40	..	6400

80 features

Targets(y):

Price($)
254000
500000
$\vdots$
210000

Example: Regression task

How can we put data into NN?

Data(X):

1500 examples

ID	Number of rooms	Floor	LotArea	...	Distance from center (m)
1	2	5	64	...	3620
2	3	3	80	...	1076
⋮	⋮	⋮	⋮	⋱	⋮
N	2	10	40	..	6400

80 features

Split data to small pieces

Batches:

X_1

1	2	5	64	...	3620
⋮	⋮	⋮	⋮	⋱	⋮
k	1	1	24	..	7800

k examples

X_2

k+1	4	7	98	...	3001
⋮	⋮	⋮	⋮	⋱	⋮
s	2	2	45	..	1200

⋮

80 features

Example: Regression task

Batches:

X_1

1	2	5	64	...	3620
⋮	⋮	⋮	⋮	⋱	⋮
k	1	1	24	..	7800

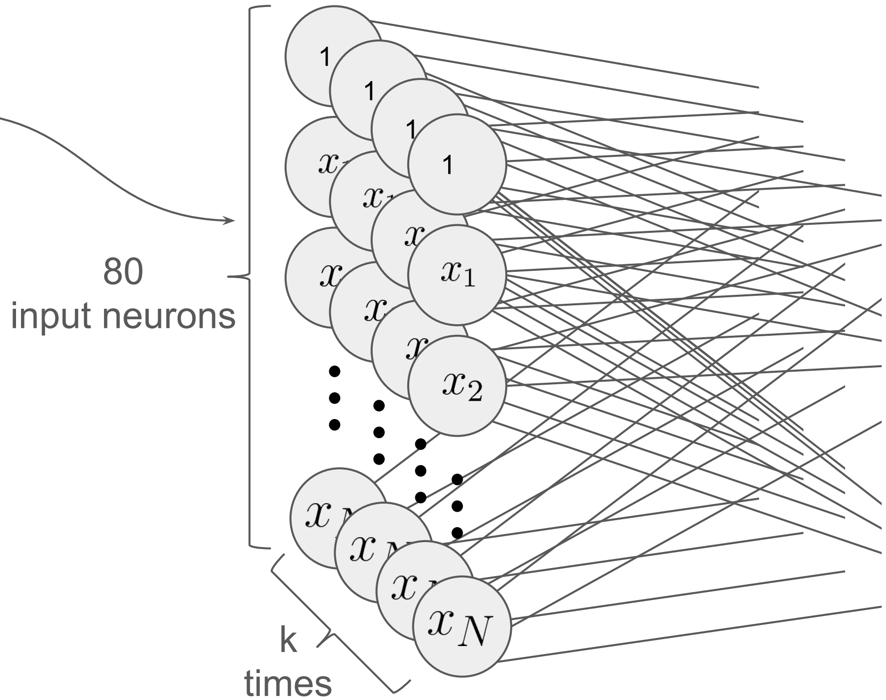
k examples X_2

k+1	4	7	98	...	3001
⋮	⋮	⋮	⋮	⋱	⋮
s	2	2	45	..	1200

⋮

80 features

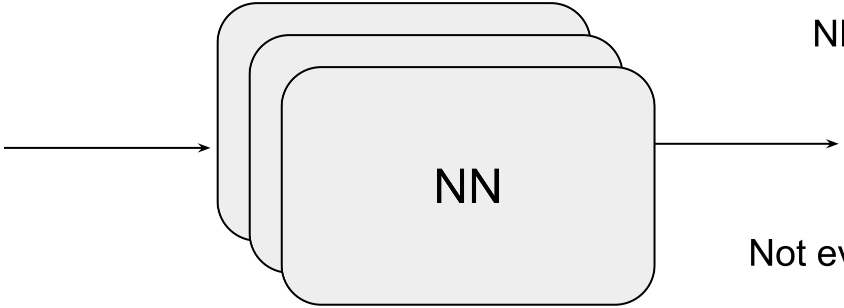
Many networks with the same weights...



Example: Prediction

X_1

1	2	5	64	...	3620
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
k	1	1	24	..	7800



NN's Prediction

$\hat{y}$

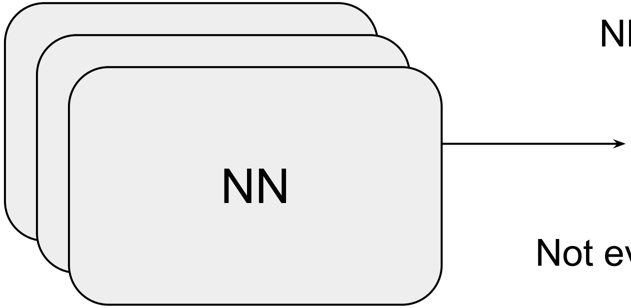
Not even close..

$$\hat{y} = \begin{pmatrix} 381620 \\ 93402 \\ \vdots \\ 103864 \end{pmatrix}$$

Example: Prediction

X_1

1	2	5	64	...	3620
⋮	⋮	⋮	⋮	⋱	⋮
k	1	1	24	..	7800



NN's Prediction

$\hat{y}$

Not even close..

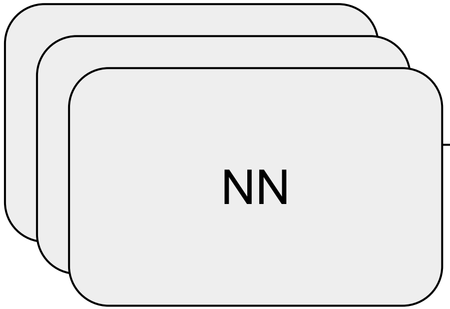
$$L(X, y) = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 = 40571256169.85$$

$$\hat{y} = \begin{pmatrix} 381620 \\ 93402 \\ \vdots \\ 103864 \end{pmatrix}$$

Example: Prediction

X_1

1	2	5	64	...	3620
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
k	1	1	24	..	7800



NN's Prediction

$\hat{y}$

Not even close..

$$L(X, y) = \frac{1}{N} \sum_{i}^{N} (y_i - \hat{y}_i)^2 = 40571256169.85$$

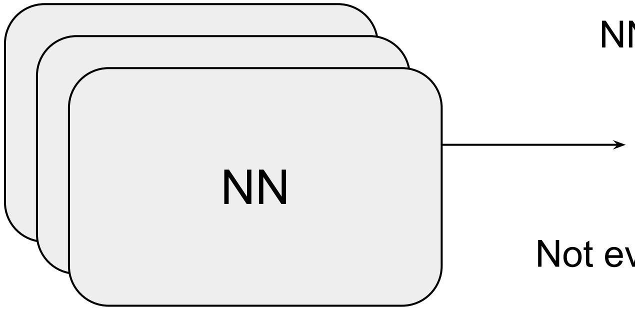
$$\hat{y} = \begin{pmatrix} 381620 \\ 93402 \\ \vdots \\ 103864 \end{pmatrix}$$

How to decrease this huge error?

Example: Prediction

X_1

1	2	5	64	...	3620
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
k	1	1	24	..	7800



NN's Prediction

$$\hat{y}$$

Not even close..

$$L(X, y) = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 = 40571256169.85$$

$$\hat{y} = \begin{pmatrix} 381620 \\ 93402 \\ \vdots \\ 103864 \end{pmatrix}$$

How to decrease this huge error?

Use gradients methods!

Yet another small reminder about linear regression

Linear regression parameters
training:

$$\hat{y} = w_0 + w_1 x$$

$$L(y, \hat{y}) = (y - \hat{y})^2$$
$$= (y - w_1 x - w_0)^2$$

$$w_{new} = w_{old} - \alpha \frac{\partial L(w,x)}{\partial w}$$
$$\alpha \in \mathbb{R} - \text{learning rate}$$

Neural Network Training

Linear regression parameters training:

$$\hat{y} = w_0 + w_1 x$$

$$L(y, \hat{y}) = (y - \hat{y})^2$$
$$= (y - w_1 x - w_0)^2$$

$$w_{new} = w_{old} - \alpha \frac{\partial L(w,x)}{\partial w}$$
$$\alpha \in \mathbb{R} - \text{learning rate}$$

Fully Connected Neural Network:

$$L(X, y) = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2$$

$$\hat{y} = (\ldots((X \times W^{(1)})W^{(2)})\ldots)\mathbf{w}^{(\mathbf{M})}$$

$$w_{new} = w_{old} - \alpha \frac{\partial L(w,x)}{\partial w}$$

Chain Rule

$$\frac{\partial L}{\partial w} = \frac{\partial L}{f_{out}} * \frac{\partial f_{out}}{W_N} * \ldots * \frac{\partial f_1}{W_1} X$$

Neural Network Training

Fully Connected Neural Network:

$$L(X, y) = \frac{1}{N}\sum_i^N (y_i - \hat{y}_i)^2$$

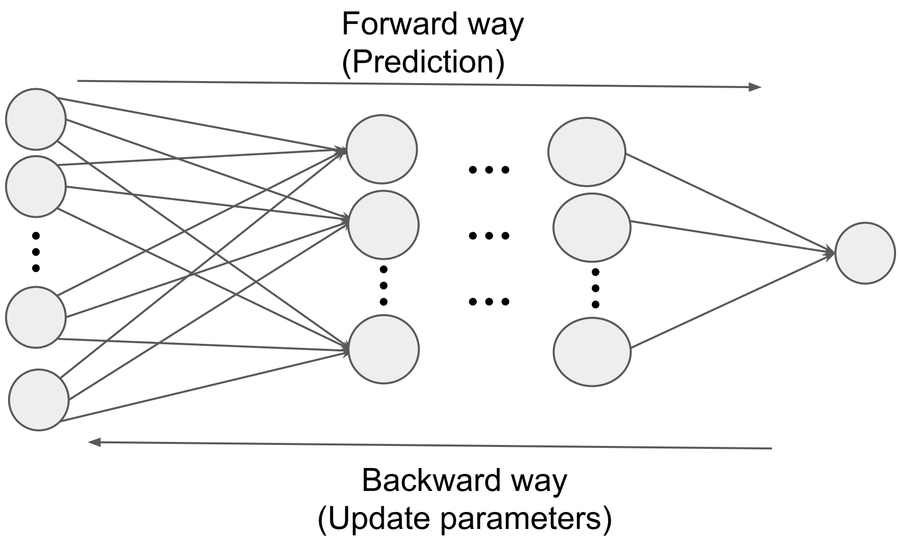
$$\hat{y} = (\ldots((X \times W^{(1)})W^{(2)})\ldots)\mathbf{w}^{(\mathbf{M})}$$

$$w_{new} = w_{old} - \alpha\frac{\partial L(w,x)}{\partial w}$$

Chain Rule

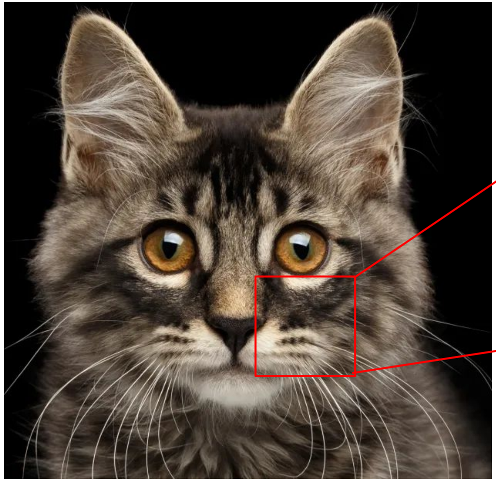
$$\frac{\partial L}{\partial w} = \frac{\partial L}{f_{out}} * \frac{\partial f_{out}}{W_N} * \ldots * \frac{\partial f_1}{W_1} X$$

This approach named Backpropagation:



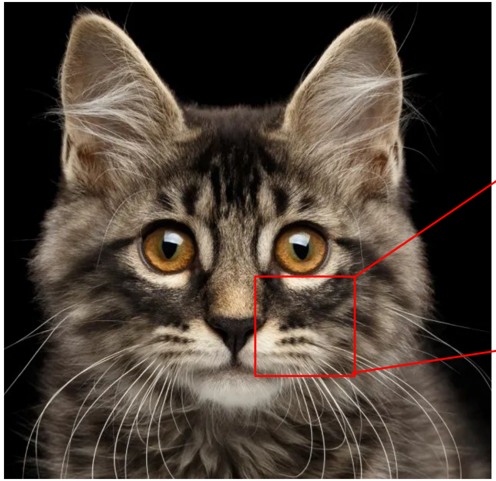
Fully Connected Neural Network with images

Fully Connected Neural Network with images



```
[[111, 135, 163, 138, 102, 138, 171, 138, 117, 106, 118, 132,  85],
 [161, 156, 169, 154, 115, 123, 154, 151, 144, 125, 103, 115, 107],
 [109, 143, 149, 134, 126, 113, 108, 124, 132, 154, 148, 130, 110],
 [102, 141, 151, 135, 129, 122, 117, 126, 126, 148, 153, 140, 116],
 [130, 155, 156, 147, 158, 167, 163, 162, 122, 135, 152, 151, 127],
 [192, 179, 152, 144, 162, 170, 157, 145, 130, 122, 134, 139, 127],
 [135, 112,  95, 110, 144, 161, 161, 160, 137, 121, 126, 131, 131],
 [119, 111, 109, 120, 133, 139, 143, 149, 131, 132, 148, 155, 158],
 [138, 146, 149, 140, 134, 139, 144, 142, 128, 138, 153, 158, 169],
 [136, 152, 151, 128, 122, 143, 151, 136, 136, 133, 125, 121, 148],
 [143, 152, 120, 117, 145, 121, 101, 132, 134, 117, 122, 155, 136],
 [172, 141, 107, 120, 151, 141, 132, 155, 120, 108, 120, 155, 137],
 [121,  99,  94, 106, 118, 119, 116, 124,  93,  89, 112, 144, 124],
 [ 87,  94, 103, 105, 130, 155, 145, 136, 112, 108, 130, 151, 127],
 [148, 135, 112, 102, 149, 171, 131, 118, 143, 121, 130, 143, 132],
 [135, 108,  98, 111, 157, 155, 114, 138, 138, 103, 107, 126, 136],
 [140, 106, 122, 139, 148, 130, 106, 149, 126, 103, 121, 139, 147],
 [137,  82,  93, 101,  99, 106, 100, 128, 114, 110, 145, 152, 135],
 [161, 132,  99, 106, 126, 144, 142, 101,  84, 114, 124, 121, 125],
 [157, 149, 123, 118, 121, 126, 128, 103,  78, 111, 127, 122, 115],
 [146, 159, 142, 132, 128, 119, 113,  95, 103, 135, 152, 147, 132],
 [142, 160, 144, 138, 143, 131, 111,  83,  90, 109, 116, 113, 103],
 [154, 167, 149, 140, 143, 138, 122,  91, 112, 115, 106, 103, 105],
 [169, 176, 159, 140, 127, 131, 140, 114, 145, 141, 121, 113, 120],
 [166, 166, 156, 138, 114, 129, 155, 126, 115, 118,  97,  77,  77]]
```

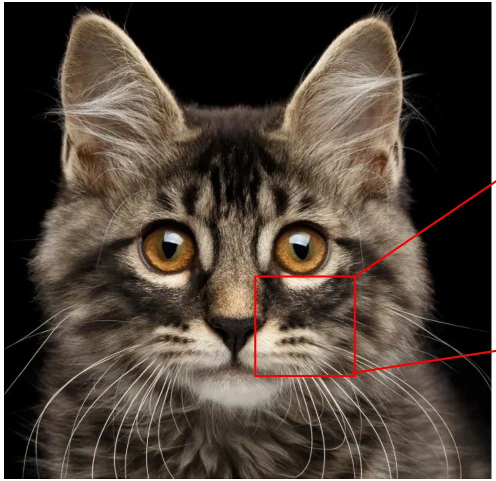
Fully Connected Neural Network with images



```
[[111, 135, 163, 138, 102, 138, 171, 138, 117, 106, 118, 132,  85],
 [161, 156, 169, 154, 115, 123, 154, 151, 144, 125, 103, 115, 107],
 [109, 143, 149, 134, 126, 113, 108, 124, 132, 154, 148, 130, 110],
 [102, 141, 151, 135, 129, 122, 117, 126, 126, 148, 153, 140, 116],
 [130, 155, 156, 147, 158, 167, 163, 162, 122, 135, 152, 151, 127],
 [192, 179, 152, 144, 162, 170, 157, 145, 130, 122, 134, 139, 127],
 [135, 112,  95, 110, 144, 161, 161, 160, 137, 121, 126, 131, 131],
 [119, 111, 109, 120, 133, 139, 143, 149, 131, 132, 148, 155, 158],
 [138, 146, 149, 140, 134, 139, 144, 142, 128, 138, 153, 158, 169],
 [136, 152, 151, 128, 122, 143, 151, 136, 136, 133, 125, 121, 148],
 [143, 152, 120, 117, 145, 121, 101, 132, 134, 117, 122, 155, 136],
 [172, 141, 107, 120, 151, 141, 132, 155, 120, 108, 120, 155, 137],
 [121,  99,  94, 106, 118, 119, 116, 124,  93,  89, 112, 144, 124],
 [ 87,  94, 103, 105, 130, 155, 145, 136, 112, 108, 130, 151, 127],
 [148, 135, 112, 102, 149, 171, 131, 118, 143, 121, 130, 143, 132],
 [135, 108,  98, 111, 157, 155, 114, 138, 138, 103, 107, 126, 136],
 [140, 106, 122, 139, 148, 130, 106, 149, 126, 103, 121, 139, 147],
 [137,  82,  93, 101,  99, 106, 100, 128, 114, 110, 145, 152, 135],
 [161, 132,  99, 106, 126, 144, 142, 101,  84, 114, 124, 121, 125],
 [157, 149, 123, 118, 121, 126, 128, 103,  78, 111, 127, 122, 115],
 [146, 159, 142, 132, 128, 119, 113,  95, 103, 135, 152, 147, 132],
 [142, 160, 144, 138, 143, 131, 111,  83,  90, 109, 116, 113, 103],
 [154, 167, 149, 140, 143, 138, 122,  91, 112, 115, 106, 103, 105],
 [169, 176, 159, 140, 127, 131, 140, 114, 145, 141, 121, 113, 120],
 [166, 166, 156, 138, 114, 129, 155, 126, 115, 118,  97,  77,  77]]
```

Thus model input will contains *WxHxC* features… (**It is a lot!)**

Fully Connected Neural Network with images

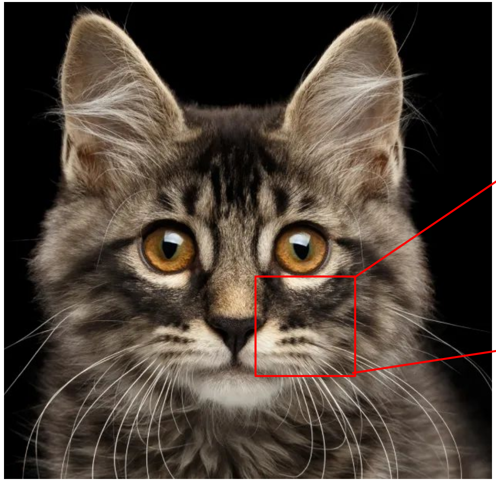


```
[[111, 135, 163, 138, 102, 138, 171, 138, 117, 106, 118, 132,  85],
 [161, 156, 169, 154, 115, 123, 154, 151, 144, 125, 103, 115, 107],
 [109, 143, 149, 134, 126, 113, 108, 124, 132, 154, 148, 130, 110],
 [102, 141, 151, 135, 129, 122, 117, 126, 126, 148, 153, 140, 116],
 [130, 155, 156, 147, 158, 167, 163, 162, 122, 135, 152, 151, 127],
 [192, 179, 152, 144, 162, 170, 157, 145, 130, 122, 134, 139, 127],
 [135, 112,  95, 110, 144, 161, 161, 160, 137, 121, 126, 131, 131],
 [119, 111, 109, 120, 133, 139, 143, 149, 131, 132, 148, 155, 158],
 [138, 146, 149, 140, 134, 139, 144, 142, 128, 138, 153, 158, 169],
 [136, 152, 151, 128, 122, 143, 151, 136, 136, 133, 125, 121, 148],
 [143, 152, 120, 117, 145, 121, 101, 132, 134, 117, 122, 155, 136],
 [172, 141, 107, 120, 151, 141, 132, 155, 120, 108, 120, 155, 137],
 [121,  99,  94, 106, 118, 119, 116, 124,  93,  89, 112, 144, 124],
 [ 87,  94, 103, 105, 130, 155, 145, 136, 112, 108, 130, 151, 127],
 [148, 135, 112, 102, 149, 171, 131, 118, 143, 121, 130, 143, 132],
 [135, 108,  98, 111, 157, 155, 114, 138, 138, 103, 107, 126, 136],
 [140, 106, 122, 139, 148, 130, 106, 149, 126, 103, 121, 139, 147],
 [137,  82,  93, 101,  99, 106, 100, 128, 114, 110, 145, 152, 135],
 [161, 132,  99, 106, 126, 144, 142, 101,  84, 114, 124, 121, 125],
 [157, 149, 123, 118, 121, 126, 128, 103,  78, 111, 127, 122, 115],
 [146, 159, 142, 132, 128, 119, 113,  95, 103, 135, 152, 147, 132],
 [142, 160, 144, 138, 143, 131, 111,  83,  90, 109, 116, 113, 103],
 [154, 167, 149, 140, 143, 138, 122,  91, 112, 115, 106, 103, 105],
 [169, 176, 159, 140, 127, 131, 140, 114, 145, 141, 121, 113, 120],
 [166, 166, 156, 138, 114, 129, 155, 126, 115, 118,  97,  77,  77]]
```

Thus model input will contains *WxHxC* features… (**It is a lot!)**

Also, we do not use information about neighboring pixels.

Fully Connected Neural Network with images



```
[[111, 135, 163, 138, 102, 138, 171, 138, 117, 106, 118, 132,  85],
 [161, 156, 169, 154, 115, 123, 154, 151, 144, 125, 103, 115, 107],
 [109, 143, 149, 134, 126, 113, 108, 124, 132, 154, 148, 130, 110],
 [102, 141, 151, 135, 129, 122, 117, 126, 126, 148, 153, 140, 116],
 [130, 155, 156, 147, 158, 167, 163, 162, 122, 135, 152, 151, 127],
 [192, 179, 152, 144, 162, 170, 157, 145, 130, 122, 134, 139, 127],
 [135, 112,  95, 110, 144, 161, 161, 160, 137, 121, 126, 131, 131],
 [119, 111, 109, 120, 133, 139, 143, 149, 131, 132, 148, 155, 158],
 [138, 146, 149, 140, 134, 139, 144, 142, 128, 138, 153, 158, 169],
 [136, 152, 151, 128, 122, 143, 151, 136, 136, 133, 125, 121, 148],
 [143, 152, 120, 117, 145, 121, 101, 132, 134, 117, 122, 155, 136],
 [172, 141, 107, 120, 151, 141, 132, 155, 120, 108, 120, 155, 137],
 [121,  99,  94, 106, 118, 119, 116, 124,  93,  89, 112, 144, 124],
 [ 87,  94, 103, 105, 130, 155, 145, 136, 112, 108, 130, 151, 127],
 [148, 135, 112, 102, 149, 171, 131, 118, 143, 121, 130, 143, 132],
 [135, 108,  98, 111, 157, 155, 114, 138, 138, 103, 107, 126, 136],
 [140, 106, 122, 139, 148, 130, 106, 149, 126, 103, 121, 139, 147],
 [137,  82,  93, 101,  99, 106, 100, 128, 114, 110, 145, 152, 135],
 [161, 132,  99, 106, 126, 144, 142, 101,  84, 114, 124, 121, 125],
 [157, 149, 123, 118, 121, 126, 128, 103,  78, 111, 127, 122, 115],
 [146, 159, 142, 132, 128, 119, 113,  95, 103, 135, 152, 147, 132],
 [142, 160, 144, 138, 143, 131, 111,  83,  90, 109, 116, 113, 103],
 [154, 167, 149, 140, 143, 138, 122,  91, 112, 115, 106, 103, 105],
 [169, 176, 159, 140, 127, 131, 140, 114, 145, 141, 121, 113, 120],
 [166, 166, 156, 138, 114, 129, 155, 126, 115, 118,  97,  77,  77]]
```

Thus model input will contains *WxHxC* features… (**It is a lot!**)

Also, we do not use information about neighboring pixels.

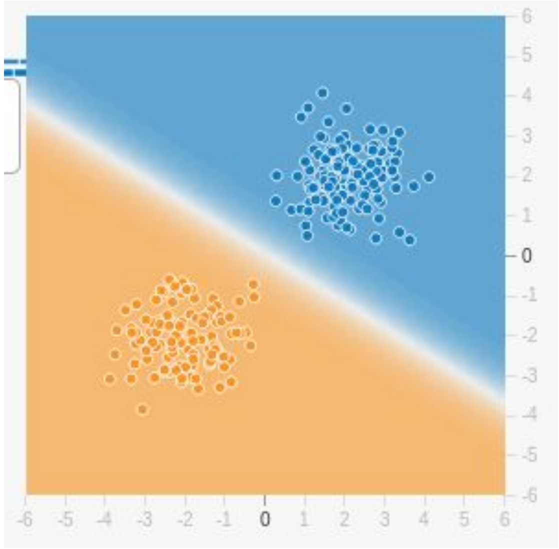
How can we use it??

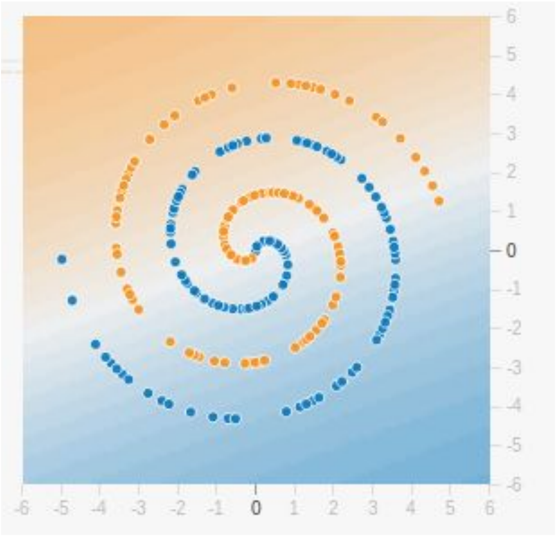
Fully Connected Neural Network. Summarizing.

1. Can be represented in the form of several linear regressions with the activations functions.

$$\hat{y} = f_M(f_{M-1}(\dots f_2(f_1(X \times W^{(1)})W^{(2)}) \dots)\mathbf{w}^{(\mathbf{M})})$$

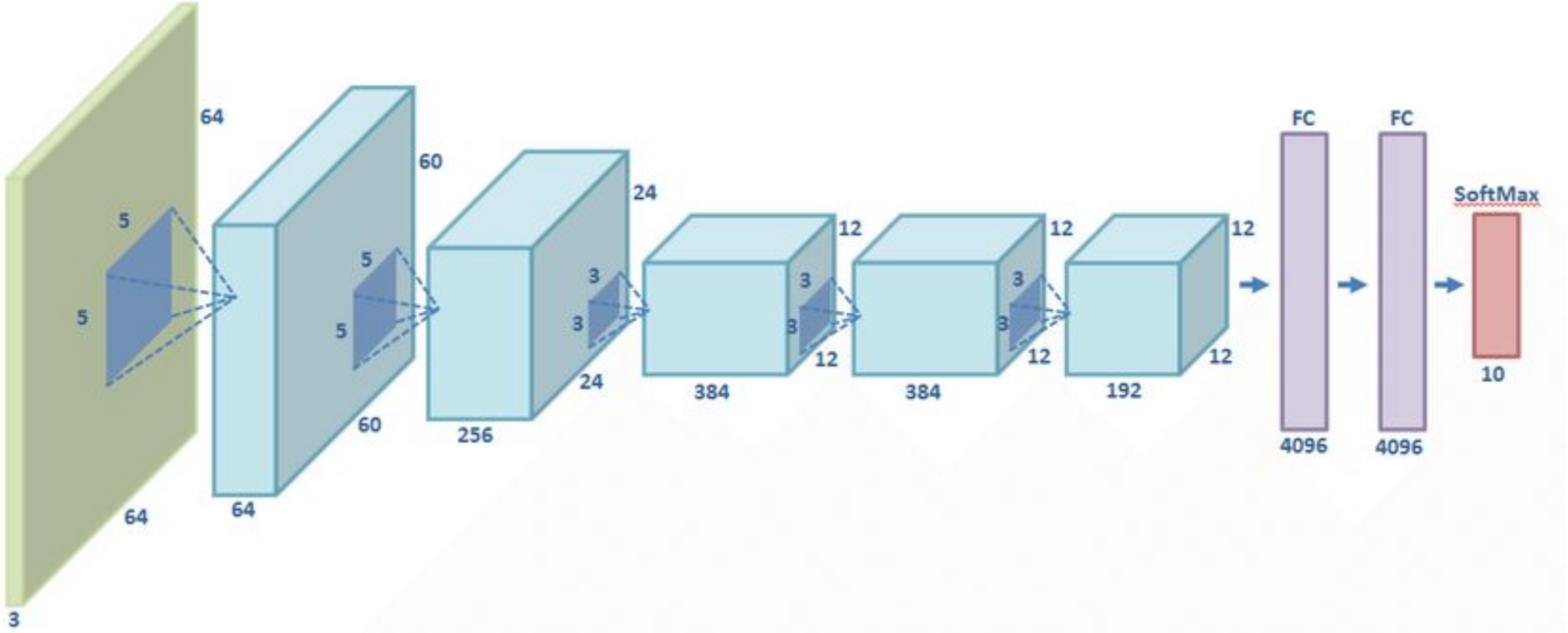
2. There are many hyperparameters for optimization.
3. For training use Backpropagation algorithm.
4. Not used with images because:
   a. Huge number of input parameters.
   b. Information about the relative position of pixels is lost.



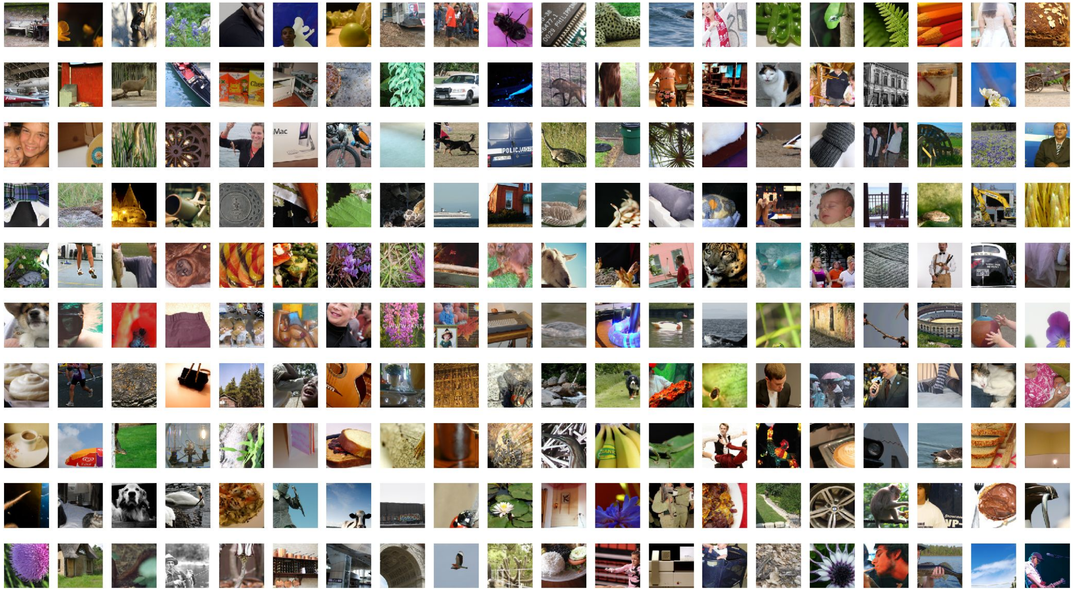


http://playground.tensorflow.org

History of Convolutional Neural Networks.



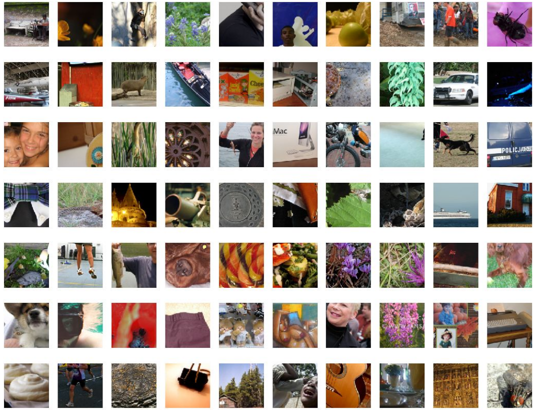
ImageNet dataset



Dataset size:
14,000,000 + images
30,000+ classes

Image size 224x224

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Dataset size:
20,000 images
1000 classes

Task:
Predict which class picture belongs to.

Evaluation types:

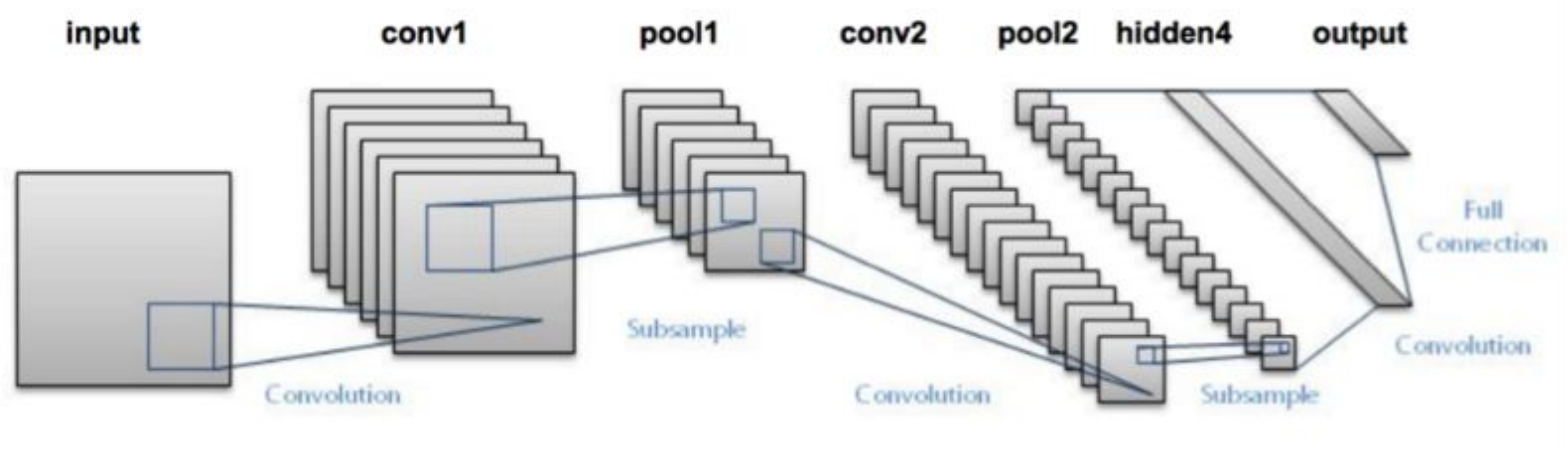
1. Top 5 error.

The percentage of the images that the classifier did not include the correct class among its top 5 guesses.

2. Top 1 error.

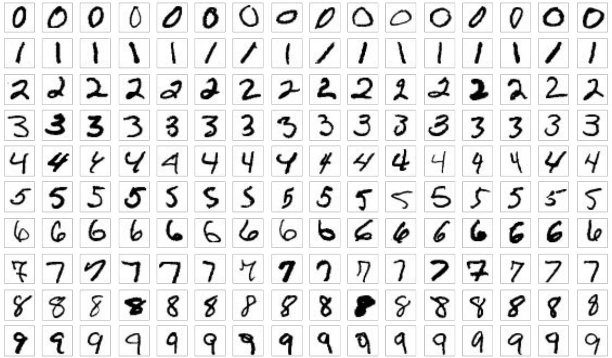
The percentage of the images that the classifier did not give the correct class the highest score.

LeNet-5 (1998)



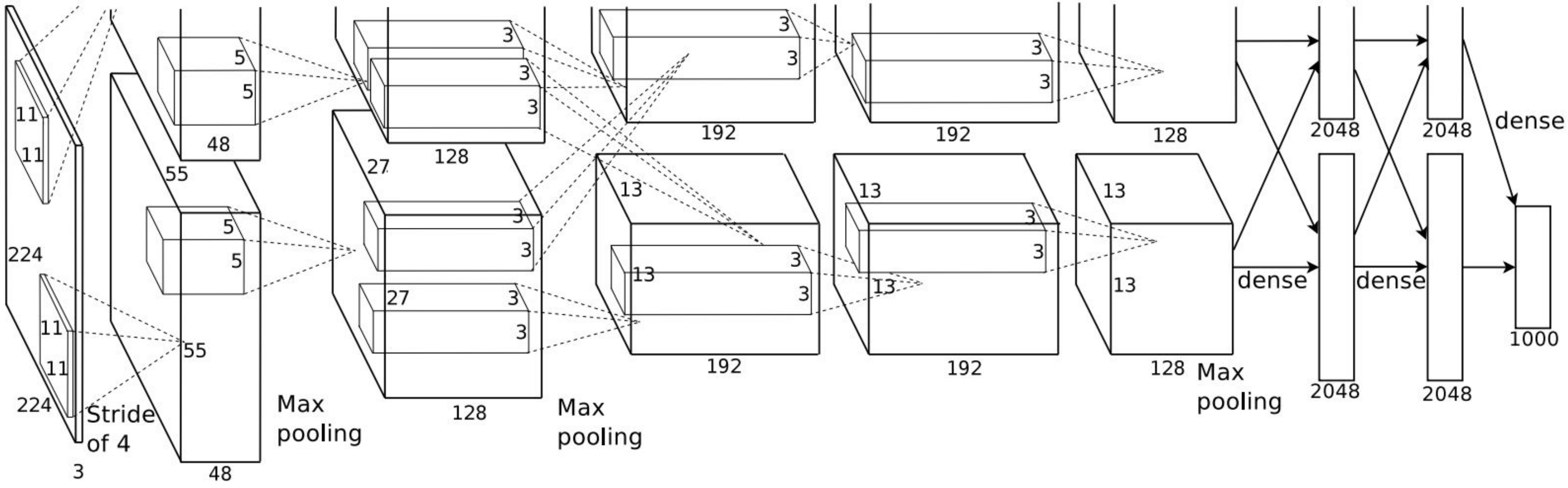
input
conv1
pool1
conv2
pool2
hidden4
output
Convolution
Subsample
Convolution
Subsample
Convolution
Full Connection

LeNet-5 (1998)

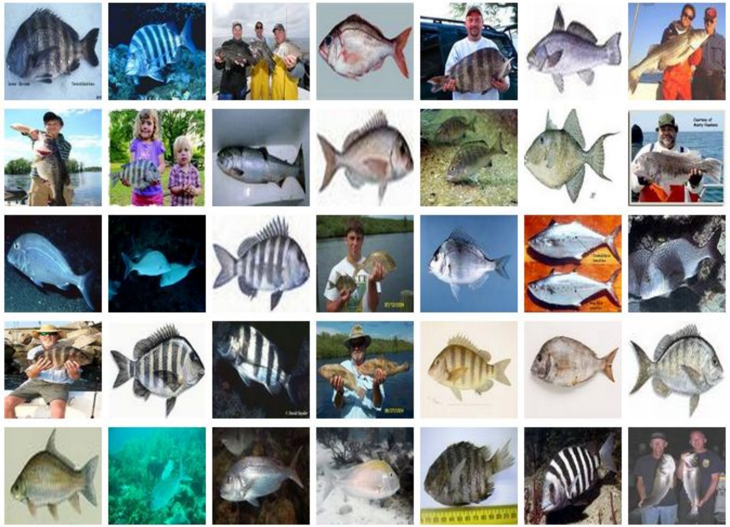


- Small 7 layers network developed by Yann Lecun.
- Used for recognizing 32x32 hand-written numbers.
- It required a lot of resources for training that were not available in 1998.

Alexnet (2012)

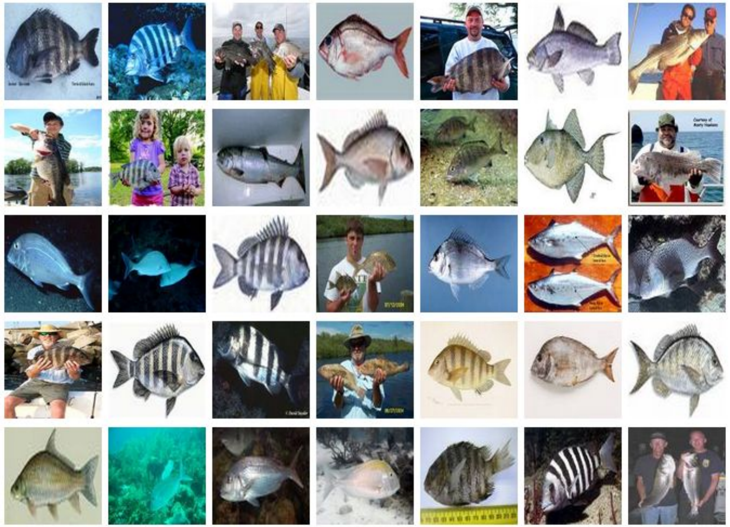


Alexnet (2012)



- Winner of ILSVRC 2012 with 15.3% of top 5 error.
- Deeper architecture with huge convolutions 11x11.
- Was trained on 2 GPU for 6 days.

Alexnet (2012)

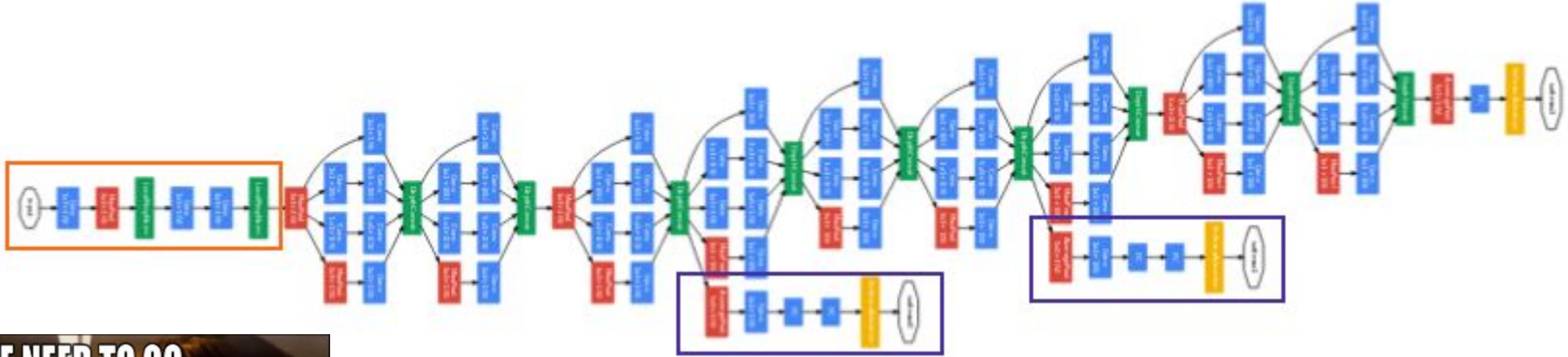


- Winner of ILSVRC 2012 with 15.3% of top 5 error.
- Deeper architecture with huge convolutions 11x11.
- Was trained on 2 GPU for 6 days.
- 60 million parameters.

Drawbacks:

- Huge convolutions with a lots of weights.

GoogLeNet or Inception v1 (2014)



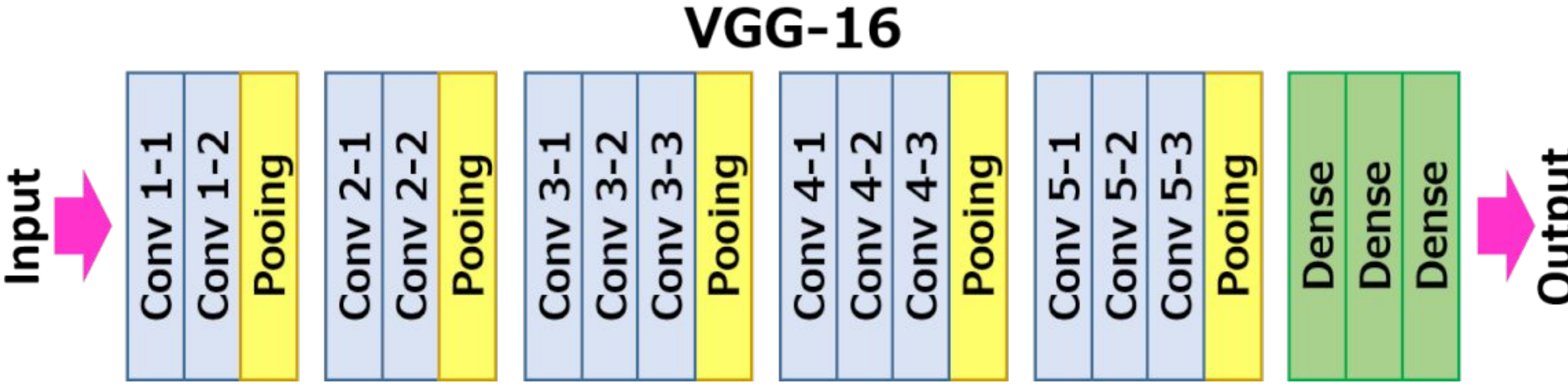
GoogLeNet or Inception v1 (2014)

- Winner of ILSVRC 2014 with 6.67% of top 5 error.
  (Right now it is better than human)
- Huge block structure with 22 layers.
- Small number of parameters. (4 million)

Drawbacks:

- Need a lot of time to reach a good quality.

VGG (2014)


VGG-16
Input
Conv 1-1
Conv 1-2
Pooing
Conv 2-1
Conv 2-2
Pooing
Conv 3-1
Conv 3-2
Conv 3-3
Pooing
Conv 4-1
Conv 4-2
Conv 4-3
Pooing
Conv 5-1
Conv 5-2
Conv 5-3
Pooing
Dense
Dense
Dense
Output

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG (2014)

- 2d place on ILSVRC 2014 with 7.32% of top 5 error.
- Only 3x3 convolutions and poolings!
- There are a lot of different versions from small VGG7 to huge VGG19. (133 - 144 million parameters)

Drawbacks:

- All parameters in fully connected layers in the end of the network.
- Vanishing gradients.

Vanishing gradients

Reminder about weights update process

$$w_{new} = w_{old} - \alpha \frac{\partial L(w,x)}{\partial w}$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{f_{out}} * \frac{\partial f_{out}}{W_N} * \ldots * \frac{\partial f_1}{W_1} X$$

< 1

Vanishing gradients

Reminder about weights update process

$$w_{new} = w_{old} - \alpha \frac{\partial L(w,x)}{\partial w}$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{f_{out}} * \frac{\partial f_{out}}{W_N} < 1 * \dots * \frac{\partial f_1}{W_1} X < 1$$

Vanishing gradients

Reminder about weights update process

$$w_{new} = w_{old} - \alpha \frac{\partial L(w,x)}{\partial w}$$

$$\underset{\approx 0}{\frac{\partial L}{\partial w}} = \frac{\partial L}{f_{out}} * \underbrace{\overset{<1}{\frac{\partial f_{out}}{W_N}} * \ldots * \overset{<1}{\frac{\partial f_1}{W_1} X}}_{\text{A lot of parameters!}}$$

Vanishing gradients

Reminder about weights update process

$$w_{new} = w_{old} - \alpha \frac{\partial L(w,x)}{\partial w}$$

$$\underbrace{\frac{\partial L}{\partial w}}_{\cong 0} = \frac{\partial L}{f_{out}} * \frac{\partial f_{out}}{W_N}^{<1} * \ldots * \frac{\partial f_1}{W_1} X^{<1}$$

A lot of parameters!

Thus,

$$w_{new} = w_{old} - \alpha \cancelto{0}{\frac{\partial L(w,x)}{\partial w}}$$

$w_{new} = w_{old}$ Almost no updates!

Vanishing gradients

Reminder about weights update process

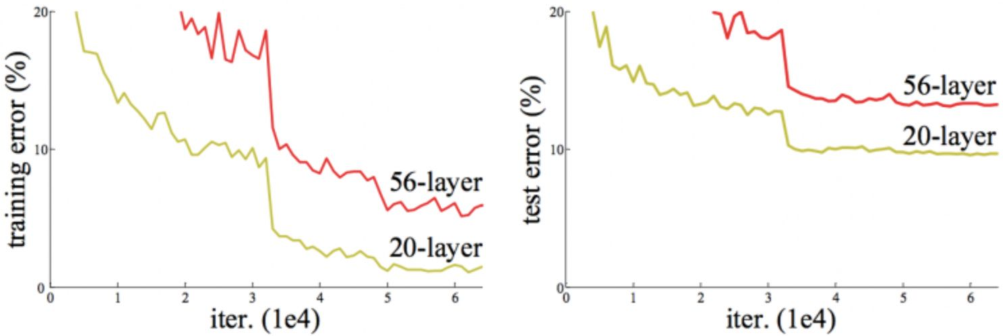
$$w_{new} = w_{old} - \alpha \frac{\partial L(w,x)}{\partial w}$$

$$\underset{\cong 0}{\frac{\partial L}{\partial w}} = \frac{\partial L}{f_{out}} * \overset{<1}{\frac{\partial f_{out}}{W_N}} \underbrace{* \dots * \overset{<1}{\frac{\partial f_1}{W_1} X}}_{\text{A lot of parameters!}}$$

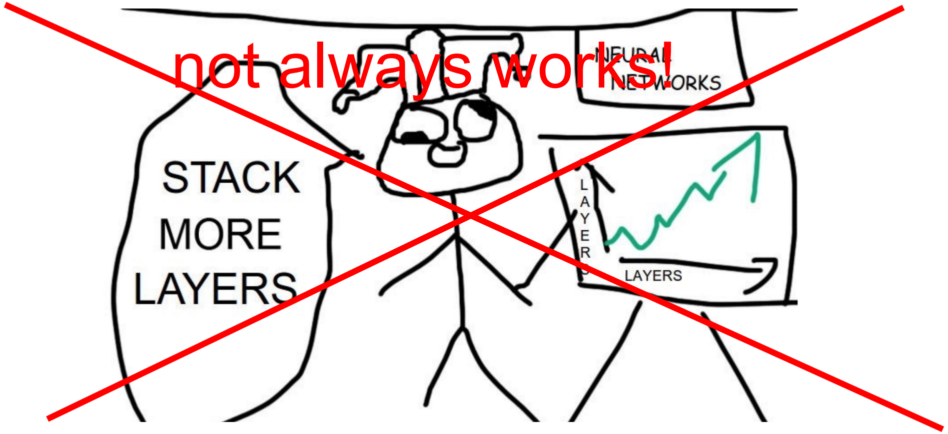
Thus,

$$w_{new} = w_{old} - \alpha \cancelto{0}{\frac{\partial L(w,x)}{\partial w}}$$

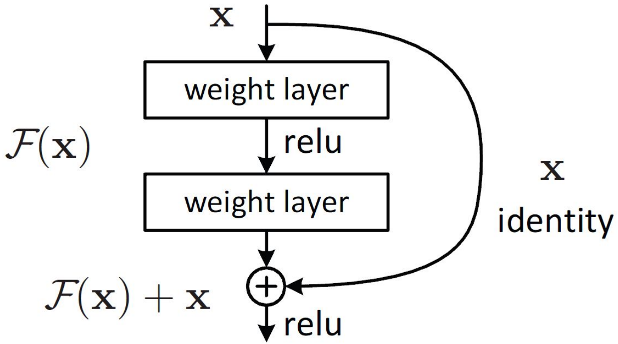
$$w_{new} = w_{old}$$ Almost no updates!



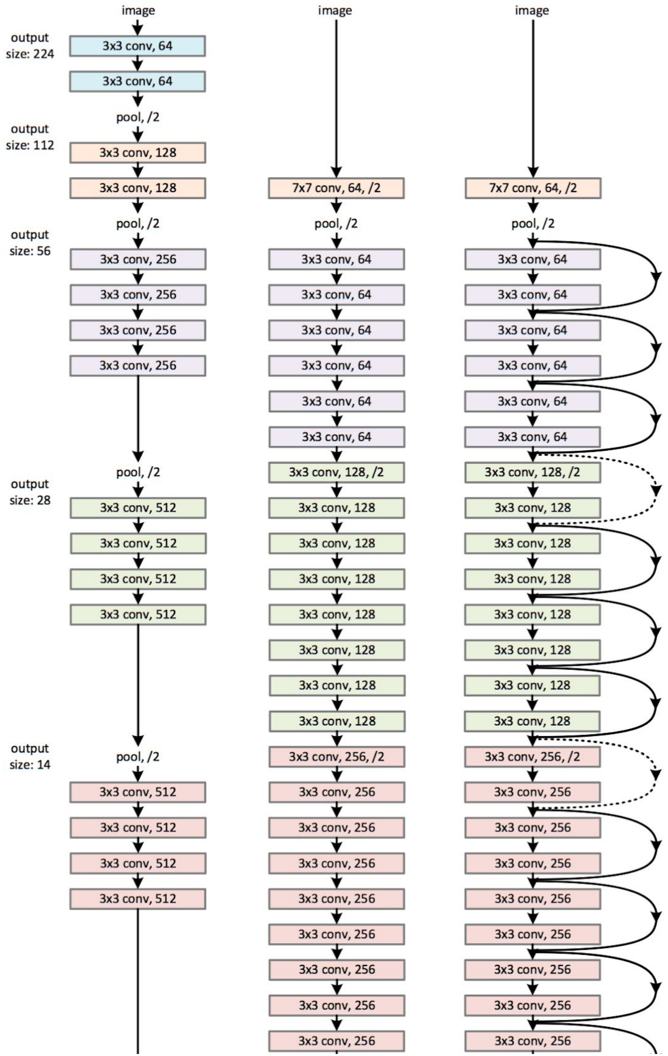
It means that networks with huge number of layers will be train very slowly.



ResNet (2015)



x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

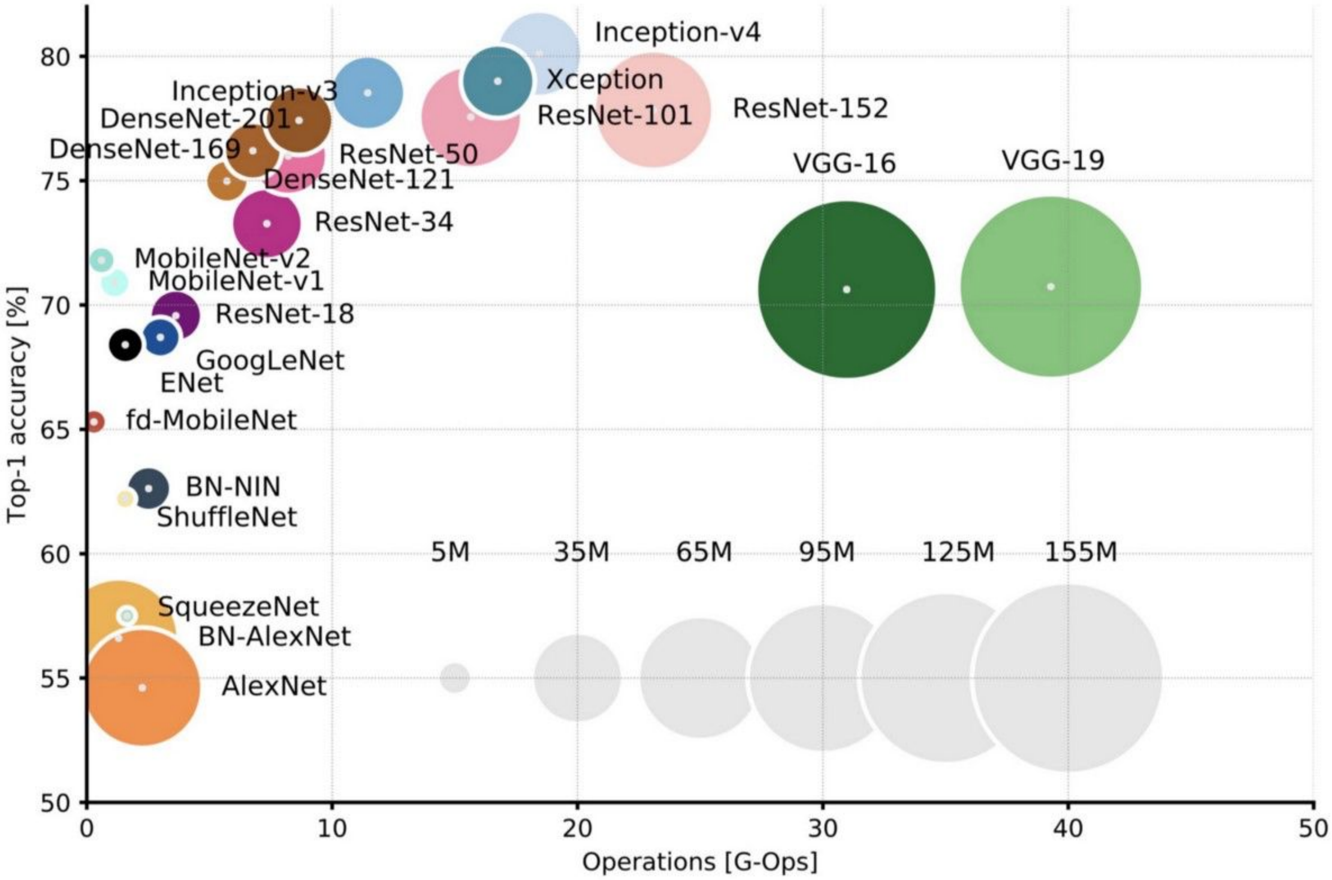


output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
image
3x3 conv, 64
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 128
pool, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256

ResNet (2015)

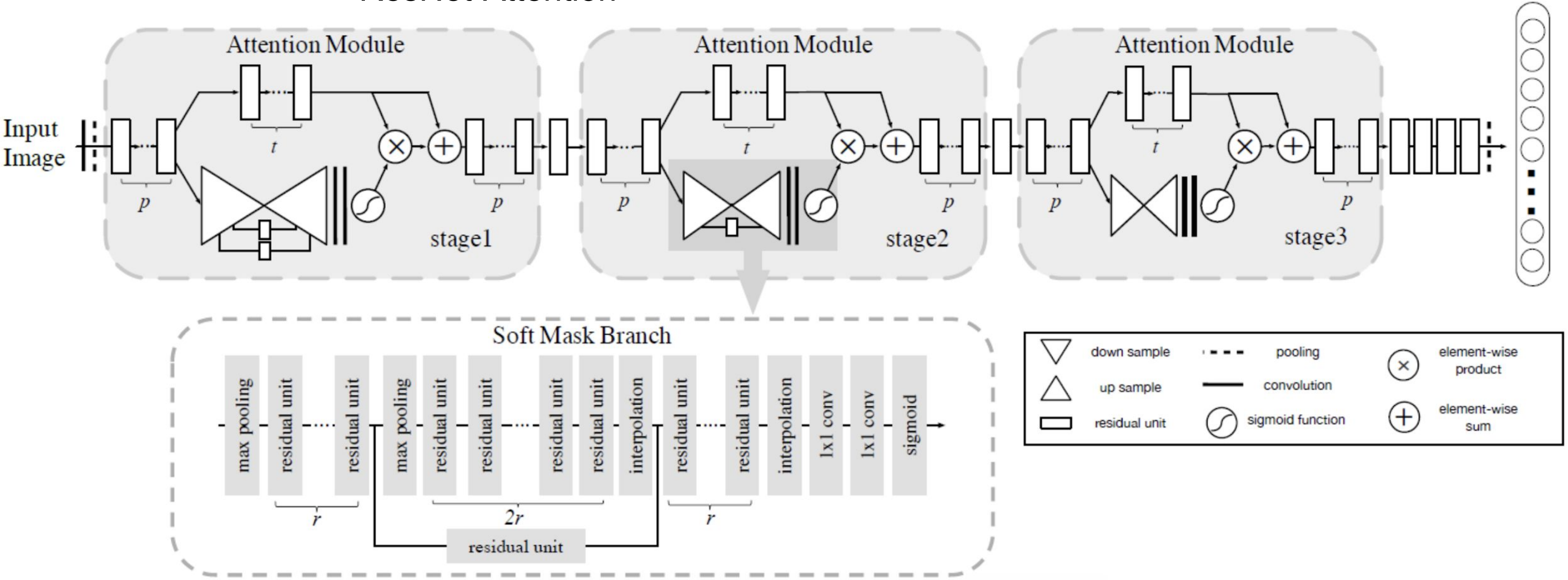
- Winner of ILSVRC 2015 with top 5 error 3.5%
- No vanishing gradients
- Really huge networks, started from ResNet18 to ResNet152.

A lots of other architectures here..

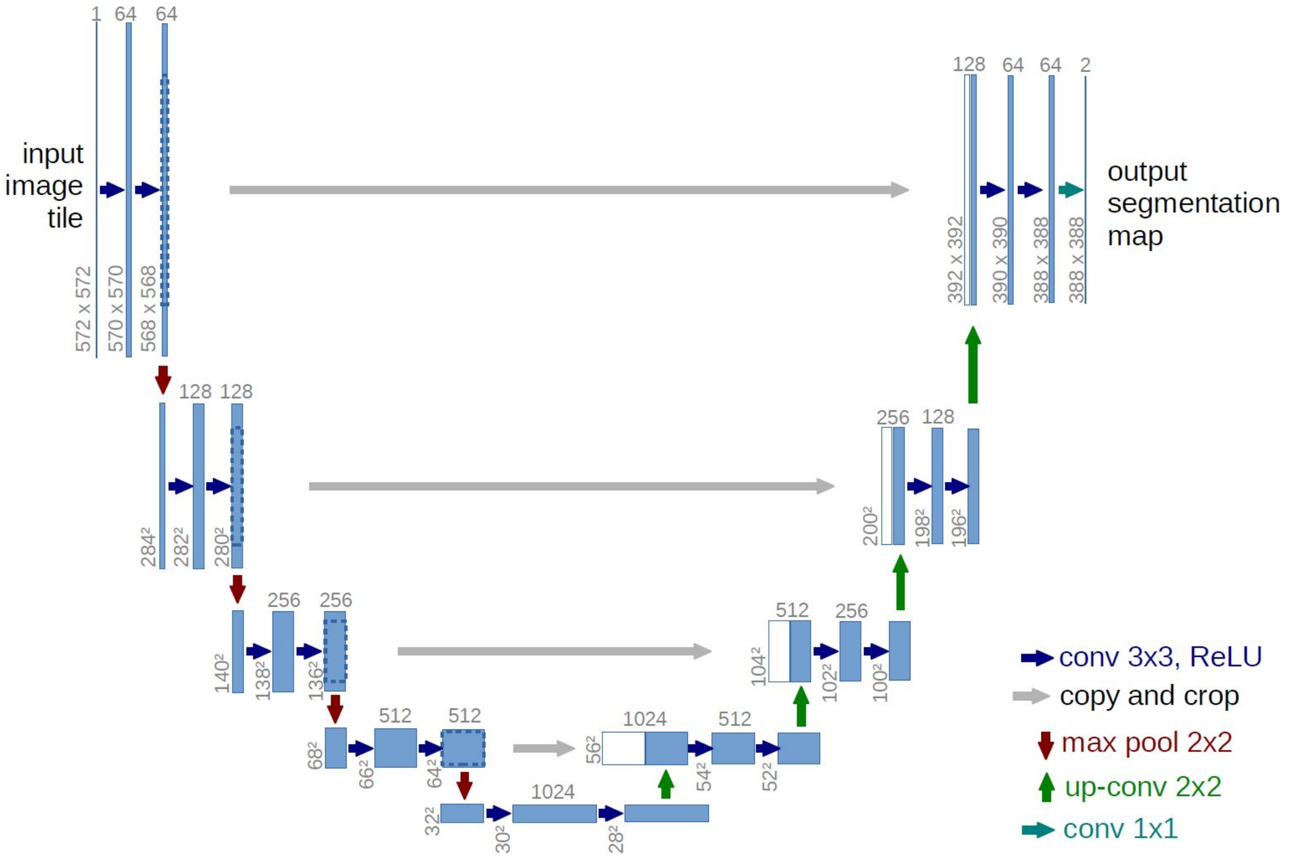


Attention models

ResNet-Attention



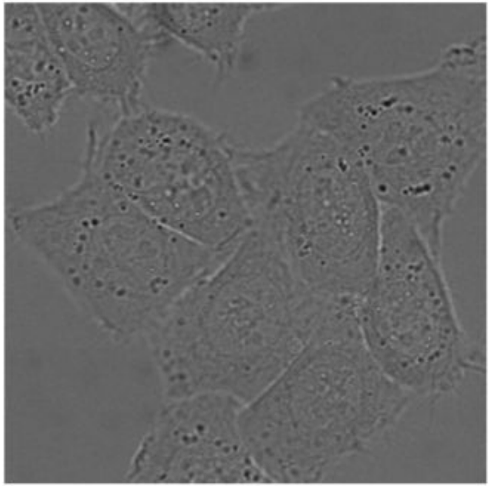
UNet (2015)

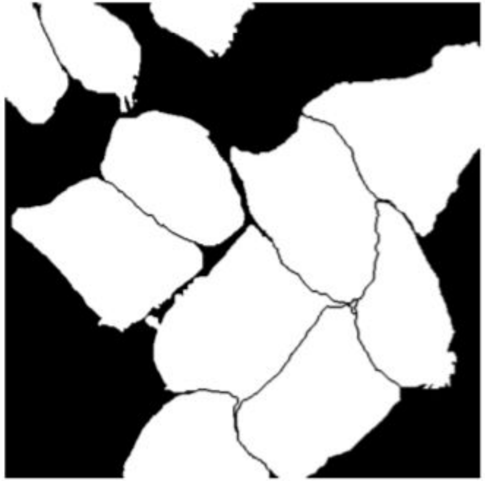


1
64
64
input
image
tile
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512
256
104²
102²
100²
256
128
200²
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
output
segmentation
map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

UNet (2015)

Binary segmentation

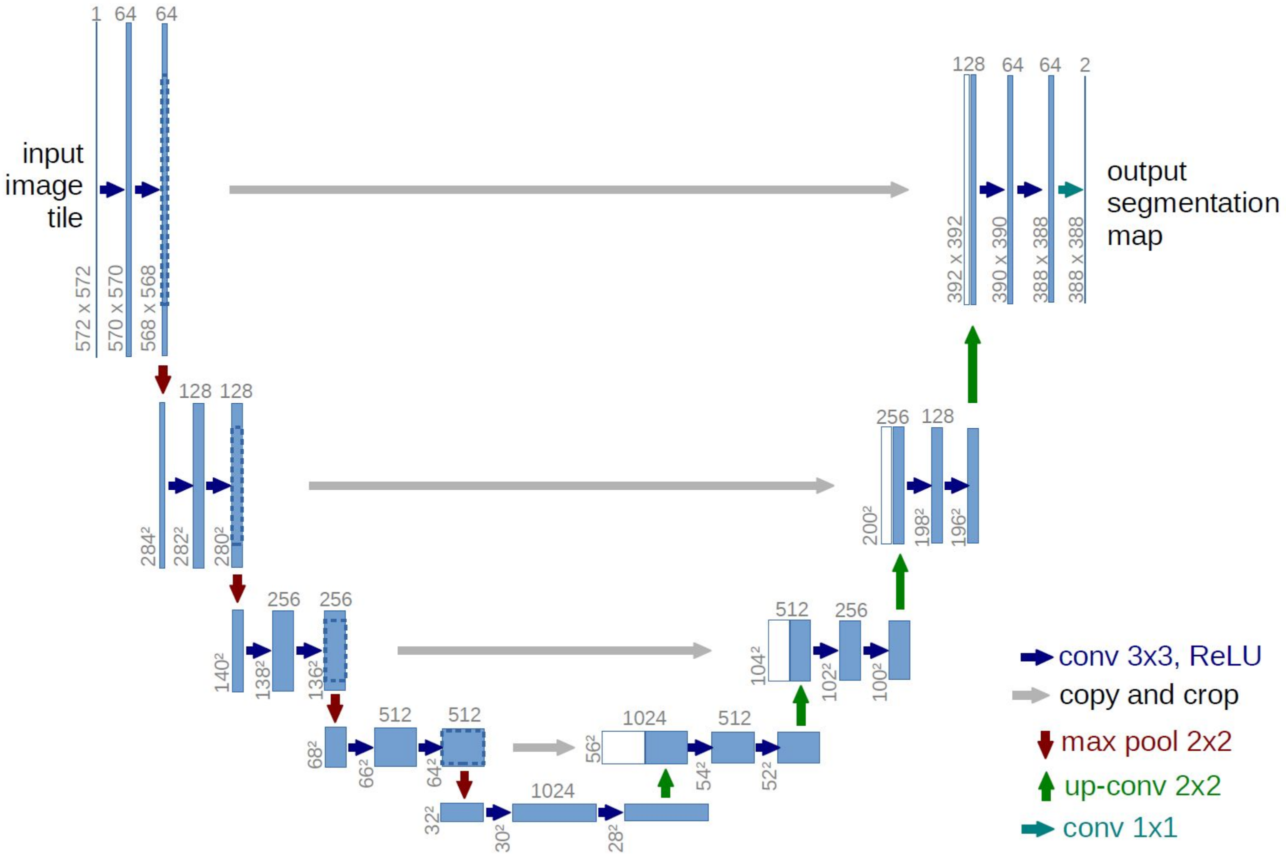




UNet (2015)

2015? To old!

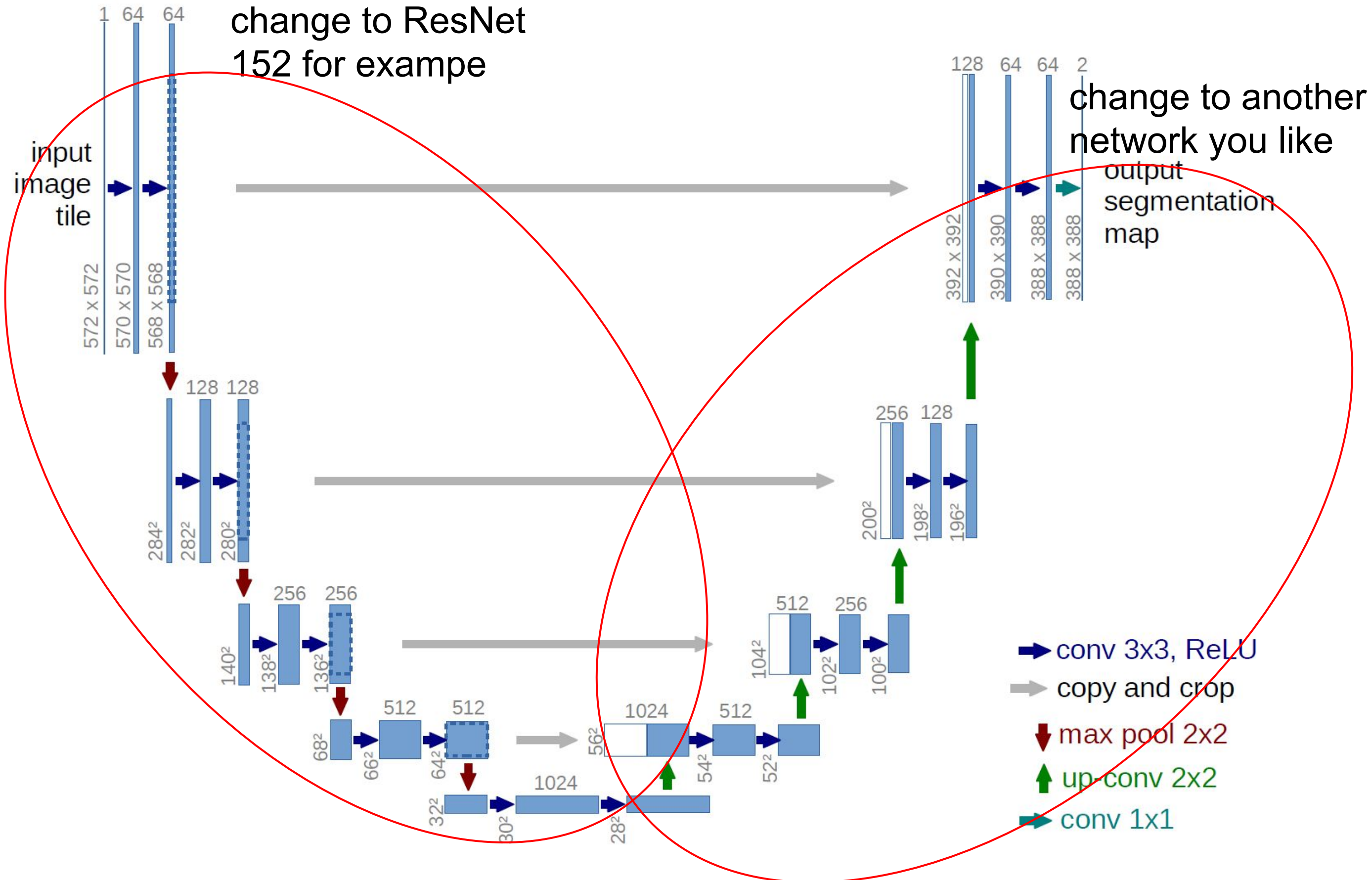
What should we do then?



UNet (2015)

2015? To old!

What should we do then?



What tasks we can solve by CNN?

- regression
- classification
- segmentation
- detection

What tasks we can solve by CNN?

- regression
- classification
- segmentation
- detection

What is the output of the network?

CNN with regression

CNN

Prediction

4

number of
dogs

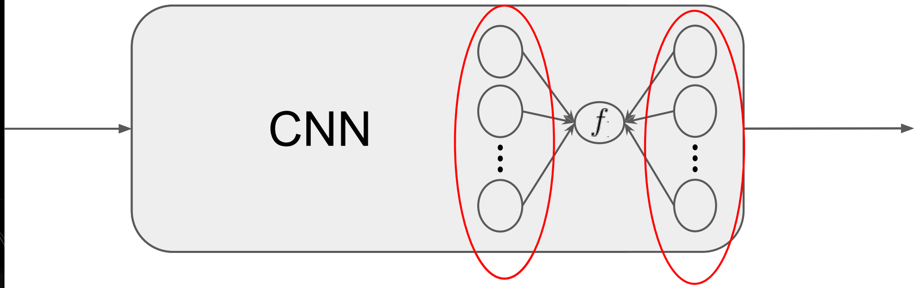
CNN with classification

f :

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$ Softmax

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

Probability for each class

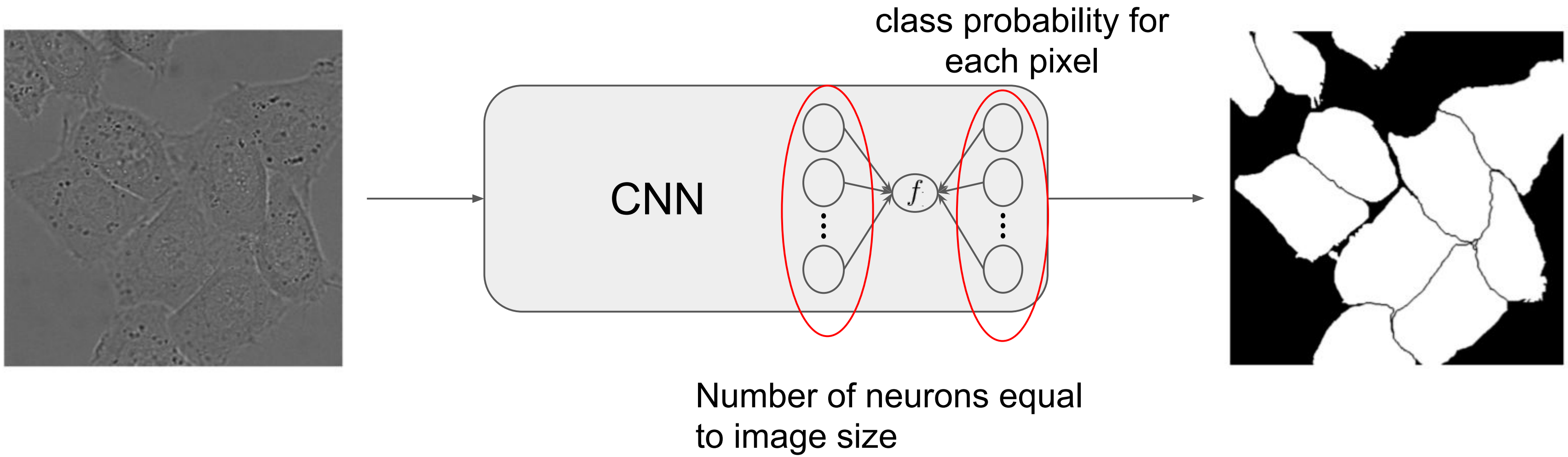


class (cat)

Number of neurons equal
to number of classes

CNN with classification

CNN with segmentation



CNN with detection one object

4 neurons:
- length of bb
- width of bb
- x
- y

CNN with detection for several objects

Number of neurons in output
layer is 5xWxH.

CNN with detection for several objects

Number of neurons in output layer is WxHx(5+NumClass).

- confidence in this bb
- length of bb
- width of bb
- x
- y
- probability for each class

план

1. Полносвязные сети.
   a. Когда хорошо? (Табличные данные можно исследовать таким образом)
   b. Когда плохо? (картинки, сеть становится слишком большой)
   c. Что делать?
2. Свертки.
   a. Сверточные сети с картинками
      i. Какие типы сетей? Что использовали раньше, что используют сейчас? (просто перечисление, какими они были и какими стали)
      ii. Чем отличается задача регрессии от классификации в терминах сетки?
      iii. Какие еще задачи существуют и как их решают? (pose estimation)
      iv. Плюсиком видео
   b. Сверточные сети со звуком
      i. ?????
   c. Сверточные сети с текстами
      i. text to numbers
      ii. типы сетей(только задачи со сверточными сетями)???
      iii. классификация текста, лейблинг, sentiment analysis
3. Рекуррентные сети.
   a. Примеры сетей для решения задач на текстах.
4. Генеративные сети.
   a. Какие задачи решаем?
   b. Какие типы сетей используются? Как развивалось?
5. Graph networks.
6. Комбинация сетей.
   a. По картинке генерить описание.