Convolutional Neural Networks (CNN) in my understanding: //source//

When we see an image of a dog, certain neurons in our brain are stimulated, sending signals to other neurons which send signals to even more neurons, ultimately resulting in certain neurons being fired that "tell" us that we see a dog. Neural networks attempt to simulate that process, building a "mini-brain" that can complete simple tasks such as distinguishing cats from dogs.

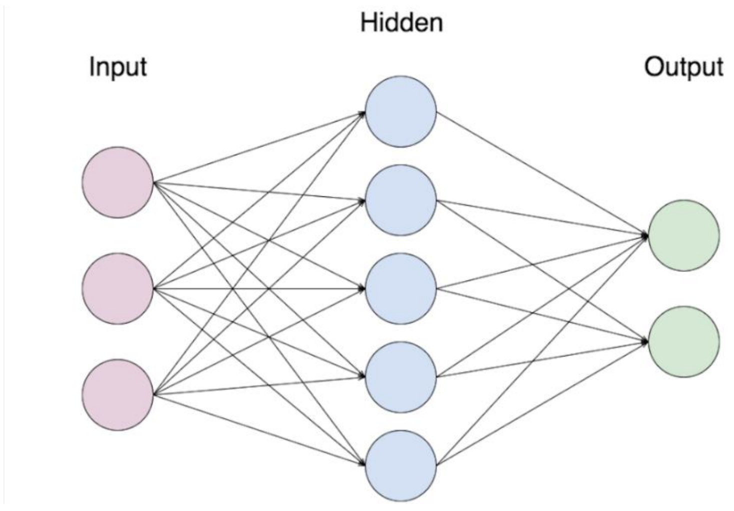


Image 1: Basic Neural Network

The most basic neural network looks something like this. We start out with an input layer of neurons, which activate neurons in the hidden layers, which then activate neurons in the output layer. Think of each circle in the diagram above as a neuron. Each neuron contains a number, knows as its activation.

Basic Components of CNN: //source//

- ❖ **Weight:** Coefficient numbers of the respective coordinates.
- ❖ **Bias:** Bias is a certain fixed value that is added to the final output to generalize the data. The addition of bias reduces the variance and hence introduces flexibility and better generalisation to the neural network.

```
output  =  sum (weights * inputs) + bias
```

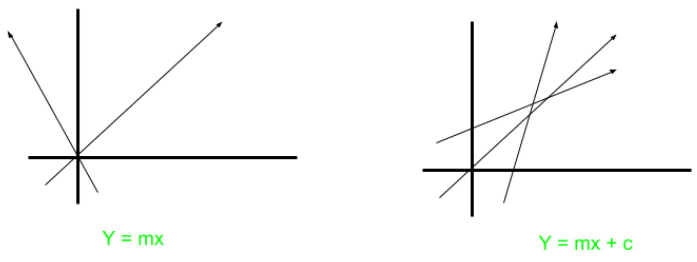


Image 2: Need of Bias

Here,

m = weight

c = bias

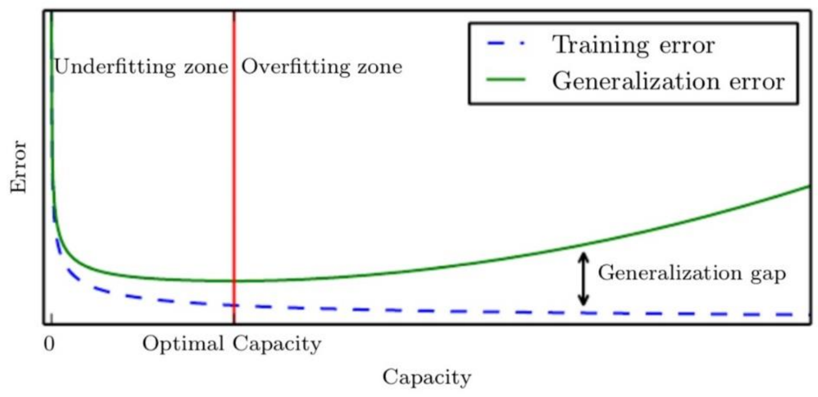


Image 3: Underfitting and Overfitting Concept

You can see that that the training error (blue dotted line) keeps on decreasing. In the initial phase, it's too high (High Bias). Later, it decreases (Low Bias).

High Bias means the model is not even fitting on the training data. So, we have to make the bias low.

- How to lower the bias?
  - ➢ Increase the epochs (iterations)
  - ➢ Try a Bigger network

❖ Variance: The Variance of a model is the difference between validation error and training error. In the figure, you can see that the gap between validation error and training error is increasing. That is, the variance is increasing (Overfitting).

- What is the importance of variance?

  Variance gives us the information about the generalization power of our model.
  If the Variance is high, the model is not performing well on the validation set. We always want a low variance.

- How to lower the variance?
  1. Increase the training set data
  2. Try Regularisation
  3. Try a different Neural Network Architecture

We always want a low bias and low variance.

- How do we take decision whether we need to higher or lower the bias/variance? Let's look for the following examples:

- Human error ~ 0%
- Train set error: 1%
- Validation set error: 11%

Variance = Validation set error - Train set error = 11 - 1 =10%
Bias = Train set error - Human error = 1%

Low Bias and High Variance (Overfitting). Since the Variance is greater than bias, this is a Variance problem. We have to lower the variance.

- Let's look at another example:

- Human error~ 0%
- Train set error: 15%
- Validation set error: 16%

Variance = Validation set error - Train set error = 16 – 15 = 1%

Bias = Train set error - Human error = 15%

High Bias and Low Variance (Underfitting). Since the Bias is greater than Variance, this is a Bias problem. We have to lower the Bias.

- Training Set: To fit the parameters [i.e., Weights]
- Validation Set: To tune the parameters [i.e., Architecture]
- Test Set: To assess the performance [i.e., Generalization and predictive power]

Multi-task Cascaded Convolutional Neural Networks (MTCNN) in my understanding: //source//

The MTCNN model consists of 3 separate networks: The P-Net, the R-Net, and the O-Net:

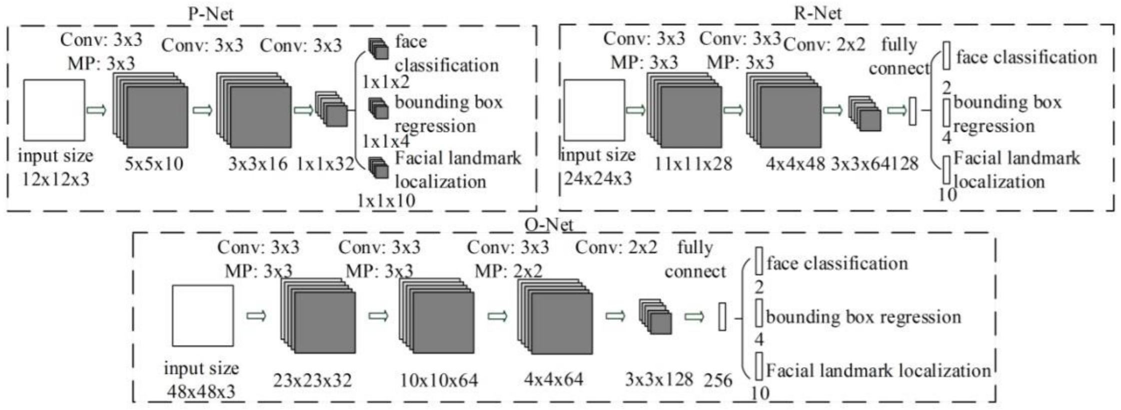


Image 3: MTCNN Basic Structure

For every image we pass in, the network creates an image pyramid: that is, it creates multiple copies of that image in different sizes.

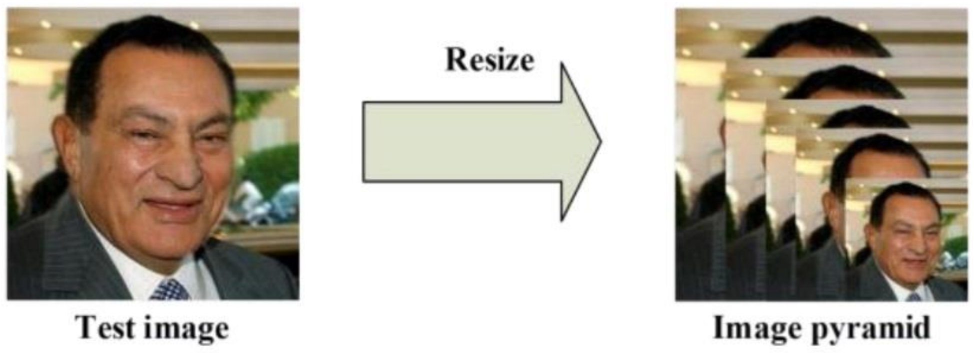
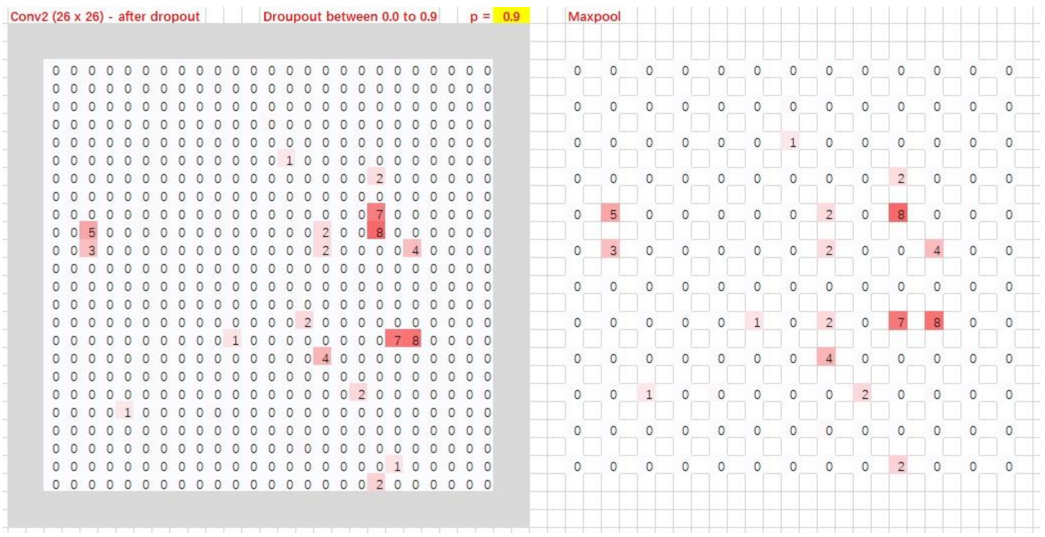


Image 4: Image Pyramid

In the P-Net, for each scaled image, a 12x12 kernel runs through the image, searching for a face. In the image below, the red square represents the kernel, which slowly moves across and down the image, searching for a face.

Image 5: 12x12 kernel in the top right corner. After scanning this corner, it shifts sideways (or downwards) by 1 pixel, and continues doing that until it has gone through the entire image.

Within each of these 12x12 kernels, 3 convolutions are run through with 3x3 kernels. After every convolution layer, a PreLu layer is implemented (when you multiply every negative pixel with a certain number ‘alpha’. ‘Alpha’ is to be determined through training). In addition, a max-pool layer is put in after the first PreLu layer (max-pool takes out every other pixel, leaving only the largest one in the vicinity).



Conv2 (26 x 26) - after dropout

Droupout between 0.0 to 0.9

p = 0.9

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

Maxpool

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

Image 6: Max-pool

After the third convolution layer, the network splits into two layers. The activations from the third layer are passed to two separate convolution layers, and a soft-max layer after one of those convolution layers (soft-max assigns decimal probabilities to every result, and the probabilities add up to 1. In this case, it outputs 2 probabilities: the probability that there is a face in the area and the probability that there isn't a face).

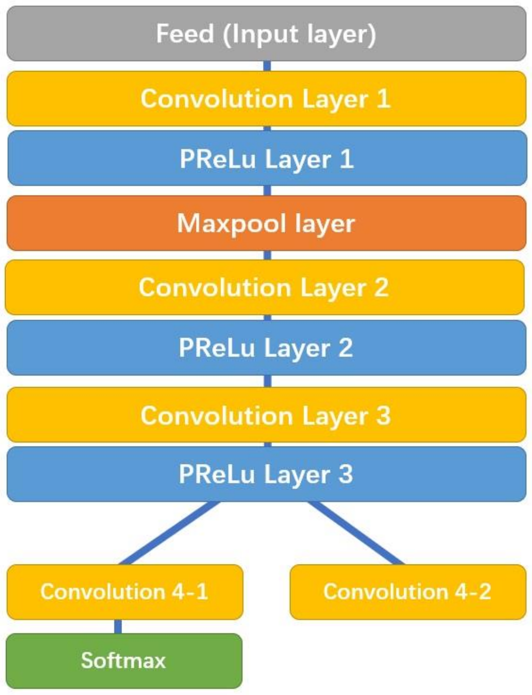


Image 7: P-Net

Convolution 4–1 outputs the probability of a face being in each bounding box, and convolution 4–2 outputs the coordinates of the bounding boxes.

```
class PNet(Network):

def _config(self):

layer_factory = LayerFactory(self)

layer_factory.new_feed(name='data', layer_shape=(None, None, None,
3))

layer_factory.new_conv(name='conv1', kernel_size=(3, 3),
channels_output=10, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu1')

layer_factory.new_max_pool(name='pool1', kernel_size=(2, 2),
stride_size=(2, 2))

layer_factory.new_conv(name='conv2', kernel_size=(3, 3),
channels_output=16, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu2')

layer_factory.new_conv(name='conv3', kernel_size=(3, 3),
channels_output=32, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu3')

layer_factory.new_conv(name='conv4-1', kernel_size=(1, 1),
channels_output=2, stride_size=(1, 1), relu=False)

layer_factory.new_softmax(name='prob1', axis=3)

layer_factory.new_conv(name='conv4-2', kernel_size=(1, 1),
channels_output=4, stride_size=(1, 1),

input_layer_name='prelu3', relu=False)
```

R-Net has a similar structure, but with even more layers. It takes the P-Net bounding boxes as its inputs, and refines its coordinates.

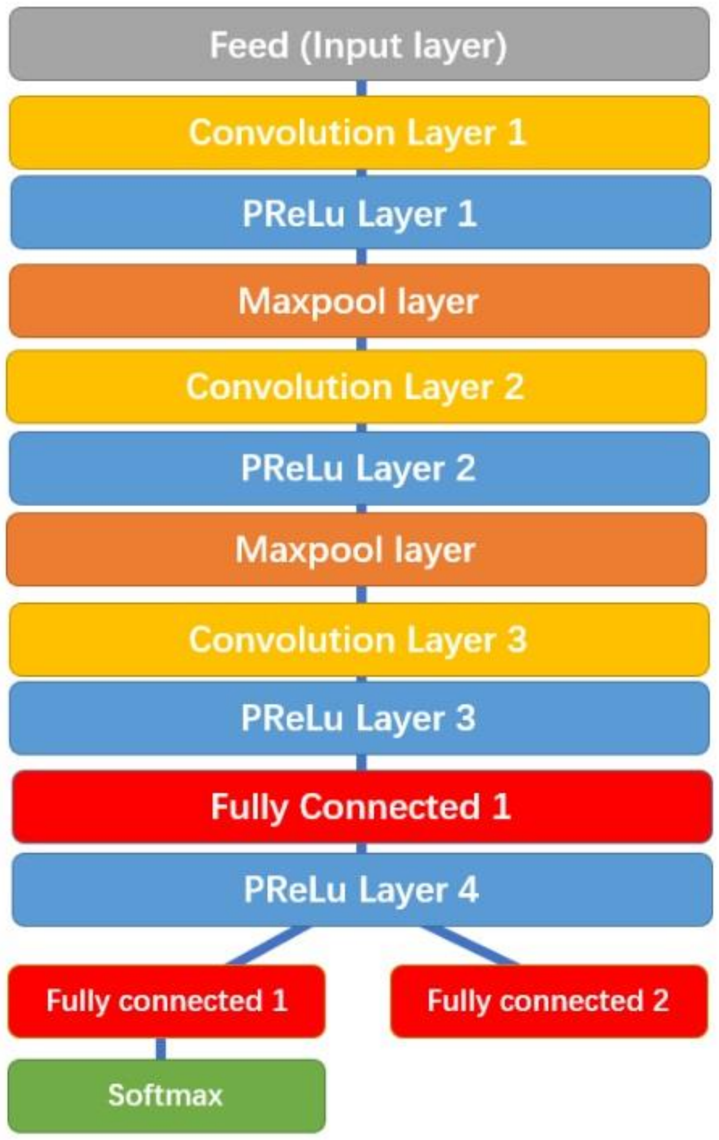


Image 8: R-Net

Similarly, R-Net splits into two layers in the end, giving out two outputs: the coordinates of the new bounding boxes and the machine's confidence in each bounding box.

```
class RNet(Network):

def _config(self):

layer_factory = LayerFactory(self)

layer_factory.new_feed(name='data', layer_shape=(None, 24, 24, 3))

layer_factory.new_conv(name='conv1', kernel_size=(3, 3),
channels_output=28, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu1')

layer_factory.new_max_pool(name='pool1', kernel_size=(3, 3),
stride_size=(2, 2))

layer_factory.new_conv(name='conv2', kernel_size=(3, 3),
channels_output=48, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu2')

layer_factory.new_max_pool(name='pool2', kernel_size=(3, 3),
stride_size=(2, 2), padding='VALID')

layer_factory.new_conv(name='conv3', kernel_size=(2, 2),
channels_output=64, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu3')

layer_factory.new_fully_connected(name='fc1', output_count=128,
relu=False)

layer_factory.new_prelu(name='prelu4')

layer_factory.new_fully_connected(name='fc2-1', output_count=2,
relu=False)

layer_factory.new_softmax(name='prob1', axis=1)

layer_factory.new_fully_connected(name='fc2-2', output_count=4,
relu=False, input_layer_name='prelu4')
```

Finally, O-Net takes the R-Net bounding boxes as inputs and marks down the coordinates of facial landmarks.

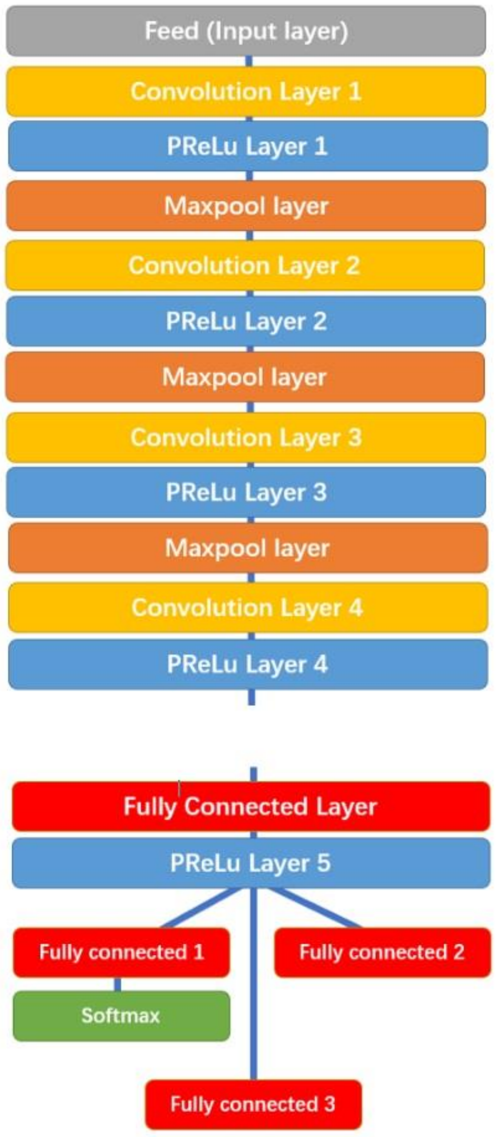
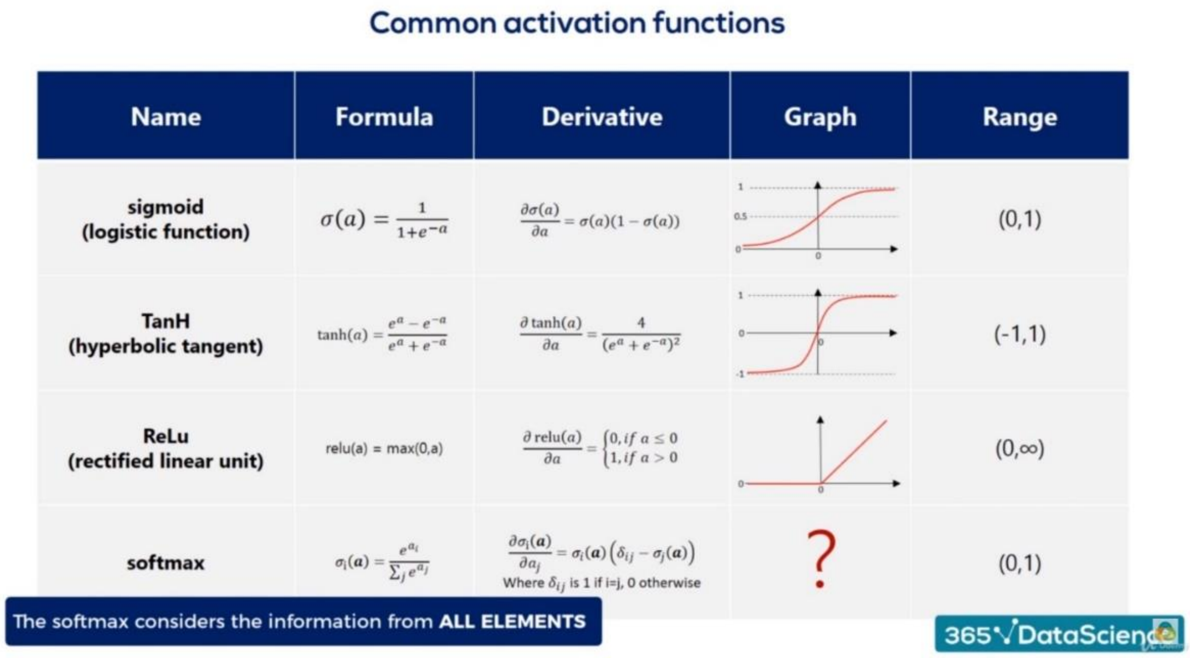


Image 9: O-Net

O-Net splits into 3 layers in the end, giving out 3 different outputs: the probability of a face being in the box, the coordinates of the bounding box, and the coordinates of the facial landmarks (locations of the eyes, nose, and mouth).

```
class ONet(Network):

def _config(self):

layer_factory = LayerFactory(self)

layer_factory.new_feed(name='data', layer_shape=(None, 48, 48, 3))

layer_factory.new_conv(name='conv1', kernel_size=(3, 3),
channels_output=32, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu1')

layer_factory.new_max_pool(name='pool1', kernel_size=(3, 3),
stride_size=(2, 2))

layer_factory.new_conv(name='conv2', kernel_size=(3, 3),
channels_output=64, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu2')

layer_factory.new_max_pool(name='pool2', kernel_size=(3, 3),
stride_size=(2, 2), padding='VALID')

layer_factory.new_conv(name='conv3', kernel_size=(3, 3),
channels_output=64, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu3')

layer_factory.new_max_pool(name='pool3', kernel_size=(2, 2),
stride_size=(2, 2))

layer_factory.new_conv(name='conv4', kernel_size=(2, 2),
channels_output=128, stride_size=(1, 1),

padding='VALID', relu=False)

layer_factory.new_prelu(name='prelu4')

layer_factory.new_fully_connected(name='fc1', output_count=256,
relu=False)

layer_factory.new_prelu(name='prelu5')

layer_factory.new_fully_connected(name='fc2-1', output_count=2,
relu=False)

layer_factory.new_softmax(name='prob1', axis=1)

layer_factory.new_fully_connected(name='fc2-2', output_count=4,
relu=False, input_layer_name='prelu5')

layer_factory.new_fully_connected(name='fc2-3', output_count=10,
relu=False, input_layer_name='prelu5')
```

```
[{'box': [277, 90, 48, 63], 'confidence': 0.9985162615776062,
'keypoints': {'left_eye': (291, 117), 'right_eye': (314, 114),
'nose': (303, 131), 'mouth_left': (296, 143), 'mouth_right': (313,
141)}}]
```

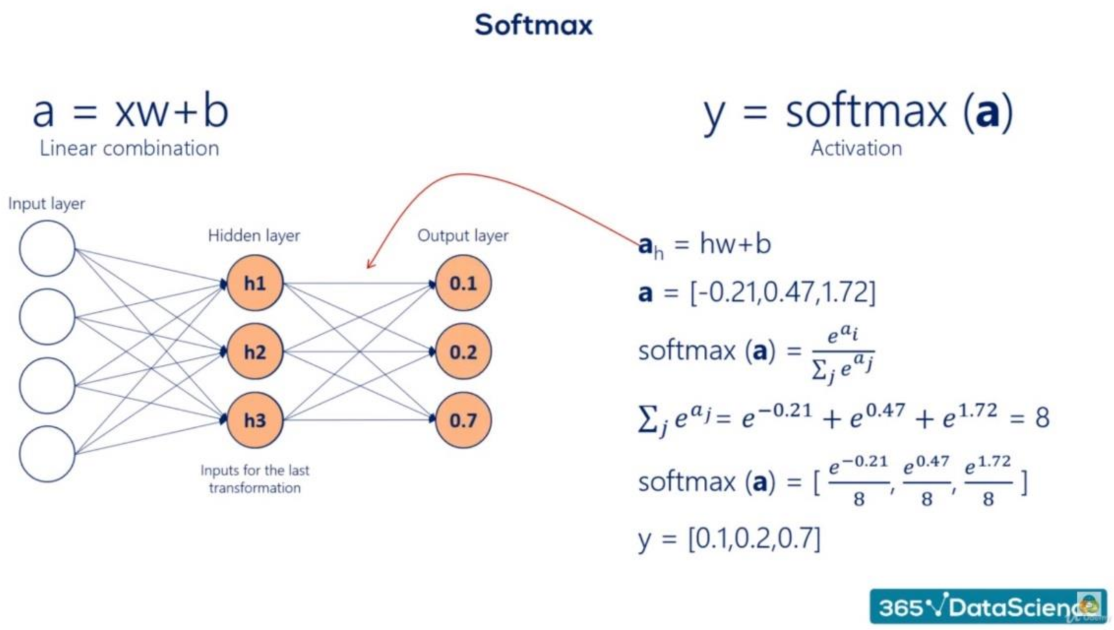


Common activation functions

Name	Formula	Derivative	Graph	Range
sigmoid (logistic function)	$\sigma(a) = \frac{1}{1+e^{-a}}$	$\frac{\partial\sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a))$		(0,1)
TanH (hyperbolic tangent)	$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$	$\frac{\partial \tanh(a)}{\partial a} = \frac{4}{(e^a + e^{-a})^2}$		(-1,1)
ReLu (rectified linear unit)	relu(a) = max(0,a)	$\frac{\partial\, \text{relu}(a)}{\partial a} = \begin{cases} 0, if\ a \le 0 \\ 1, if\ a > 0 \end{cases}$		(0,∞)
softmax	$\sigma_i(\mathbf{a}) = \frac{e^{a_i}}{\sum_j e^{a_j}}$	$\frac{\partial\sigma_i(\mathbf{a})}{\partial a_j} = \sigma_i(\mathbf{a})\left(\delta_{ij} - \sigma_j(\mathbf{a})\right)$ Where δ_{ij} is 1 if i=j, 0 otherwise	?	(0,1)

The softmax considers the information from **ALL ELEMENTS**

Image 10: Activation Functions //source//



Softmax

a = xw+b
Linear combination

y = softmax (**a**)
Activation

$\mathbf{a}_h$ = hw+b

$\mathbf{a}$ = [-0.21,0.47,1.72]

$\text{softmax}(\mathbf{a}) = \frac{e^{a_i}}{\sum_j e^{a_j}}$

$\sum_j e^{a_j} = e^{-0.21} + e^{0.47} + e^{1.72} = 8$

$\text{softmax}(\mathbf{a}) = [\frac{e^{-0.21}}{8}, \frac{e^{0.47}}{8}, \frac{e^{1.72}}{8}]$

y = [0.1,0.2,0.7]

Image 11: Soft-Max Function //source//

*** The soft-max transformation transforms a bunch of arbitrarily large or small number into a valid probability distribution. ***

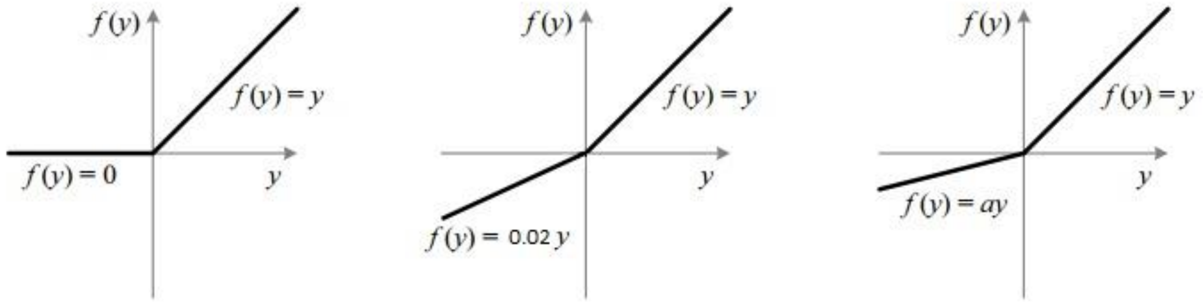


Image 12: ReLu (Left), LeakyReLu (Middle), PReLu (Right) //source//

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \le 0 \end{cases}$$

Image 13: Mathematical Definition of PReLu //source//

- if $a_i = 0$, f becomes ReLU
- if $a_i > 0$, f becomes leaky ReLU
- if a_i is a learnable parameter, f becomes PReLU

Articles I explored:

1. A Newbie's Introduction to Convolutional Neural Networks
2. Bias and Variance in Neural Network
3. How Does A Face Detection Program Work? (Using Neural Networks)
4. What Does A Face Detection Neural Network Look Like?
5. I Implemented a Face Detection Model. Here's How I Did It.

Project link: https://github.com/ipazc/mtcnn