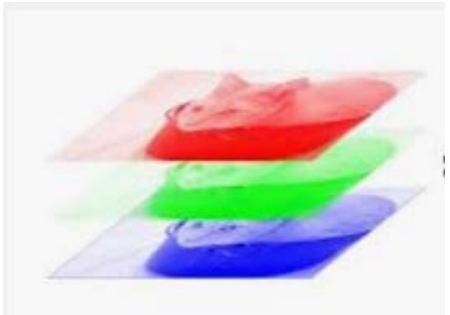
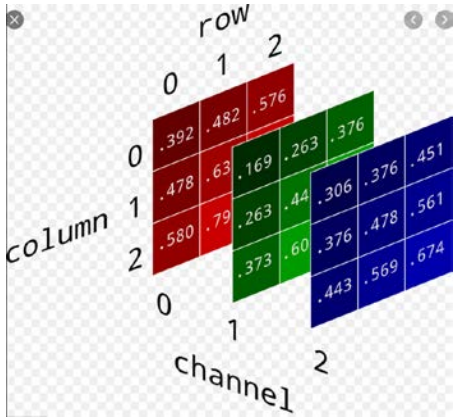
Introduction to Convolutional Neural Networks and application to face recognition

Yaofeng Su, An Vu.

First, let us Identify cats

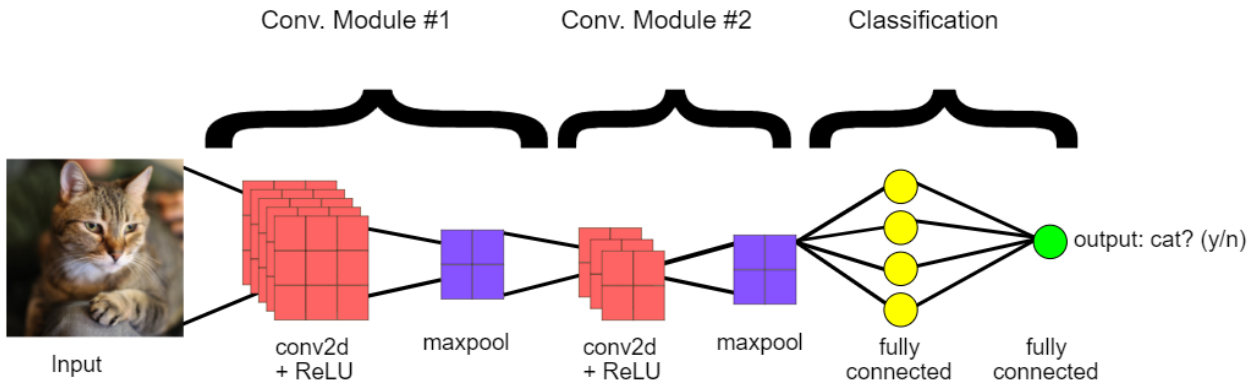
=





Original Photo

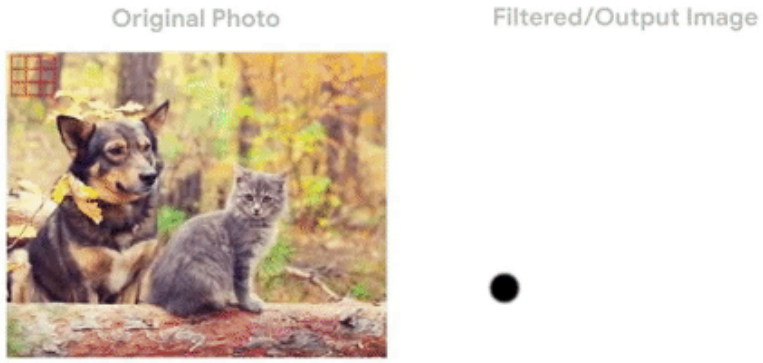
Filtered/Output Image

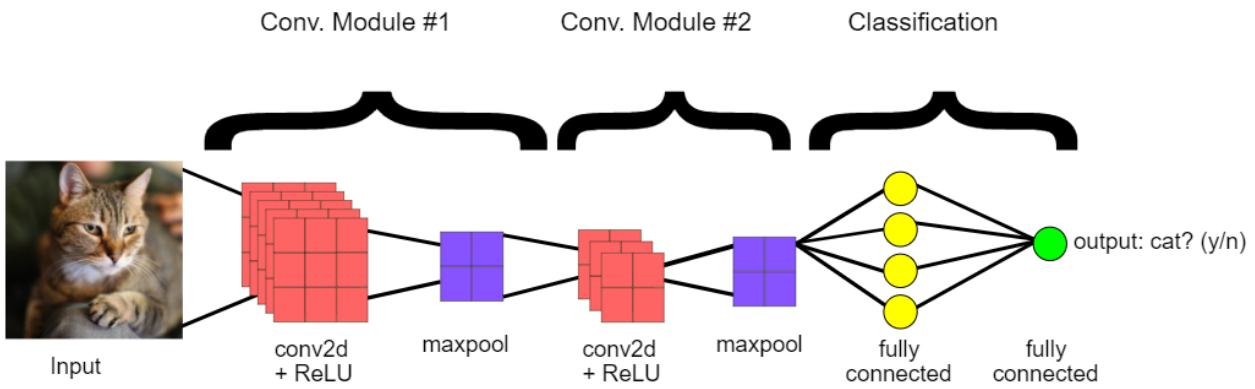


Introducing Convolutional Neural Networks

A CNN progressively extracts higher- and higher-level representations of the image content. Instead of preprocessing the data to derive features like textures and shapes, a CNN takes just the image's raw pixel data as input and "learns" how to extract these features, and ultimately infer what object they are looking at.

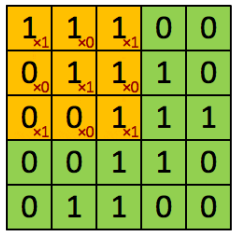
The CNN receives an input feature map: a three-dimensional matrix where the size of the first two dimensions corresponds to the length and width of the images in pixels. The size of the third dimension is 3 (corresponding to the 3 channels of a color image: red, green, and blue). The CNN comprises a stack of modules, each of which performs three operations:(Convolution, Relu, Pooling).



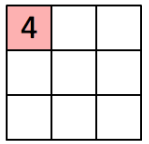


1st stage:The convolution operation

- A *convolution* extracts tiles of the input feature map, and applies filters to them to compute new features, producing an output feature map, or *convolved feature* (which may have a different size and depth than the input feature map). Convolutions are defined by two parameters: **Size of the tiles that are extracted** (typically 3x3 pixels), **The depth of the output feature map**, which corresponds to the number of filters that are applied.
- the filters(we call it-kernel) (matrices the same size as the tile size) effectively slide over the input feature map's grid horizontally and vertically, one pixel at a time, extracting each corresponding tile.

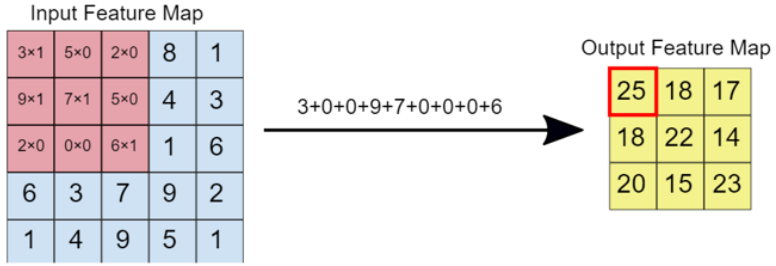


Image



Convolved Feature

A 3x3 convolution performed over a 5x5 input feature map, . There are nine possible 3x3 locations to extract tiles from the 5x5 feature map, so this convolution produces a 3x3 output feature map



The math behind convolution:

- Find the location of a spaceship with a laser sensor. Our laser sensor provides a output x(t). Now the, laser sensor is somewhat noisy. -> average together several measurements.
- Since more recent measurements are more relevant to its current position,

we want: a weighted average such that it gives more weight to recent measurements.

- A weighting function w(a), where a is the age of a measurement. apply such a weighted average operation at every moment, -> smoothed function s(t), estimate of the position s of the spaceship:

$$s(t) = \int x(a)w(t-a)da$$

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$ the **cross-correlation**

Examples:

Simple box blur

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Gaussian blur

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Line detection

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

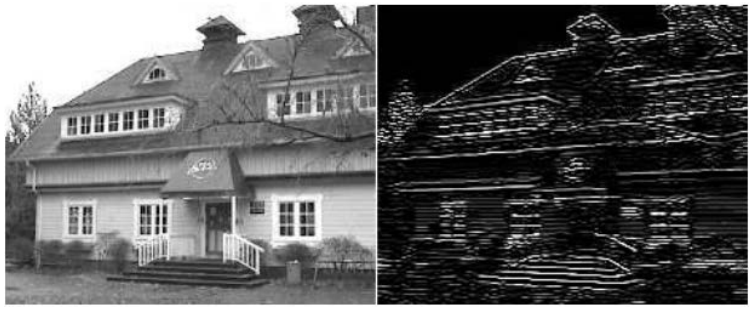
Vertical lines

-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines



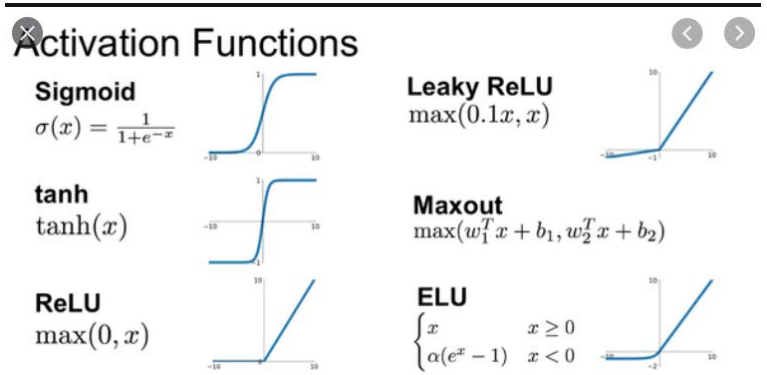
Edge detection

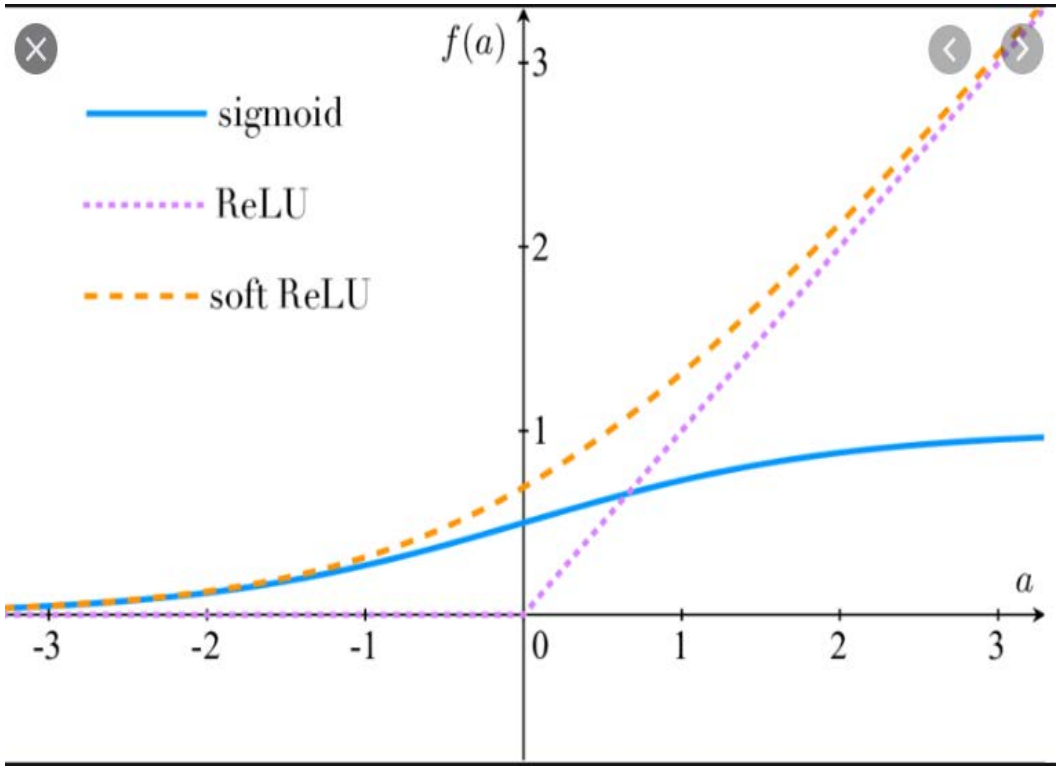
-1	-1	-1
-1	8	-1
-1	-1	-1

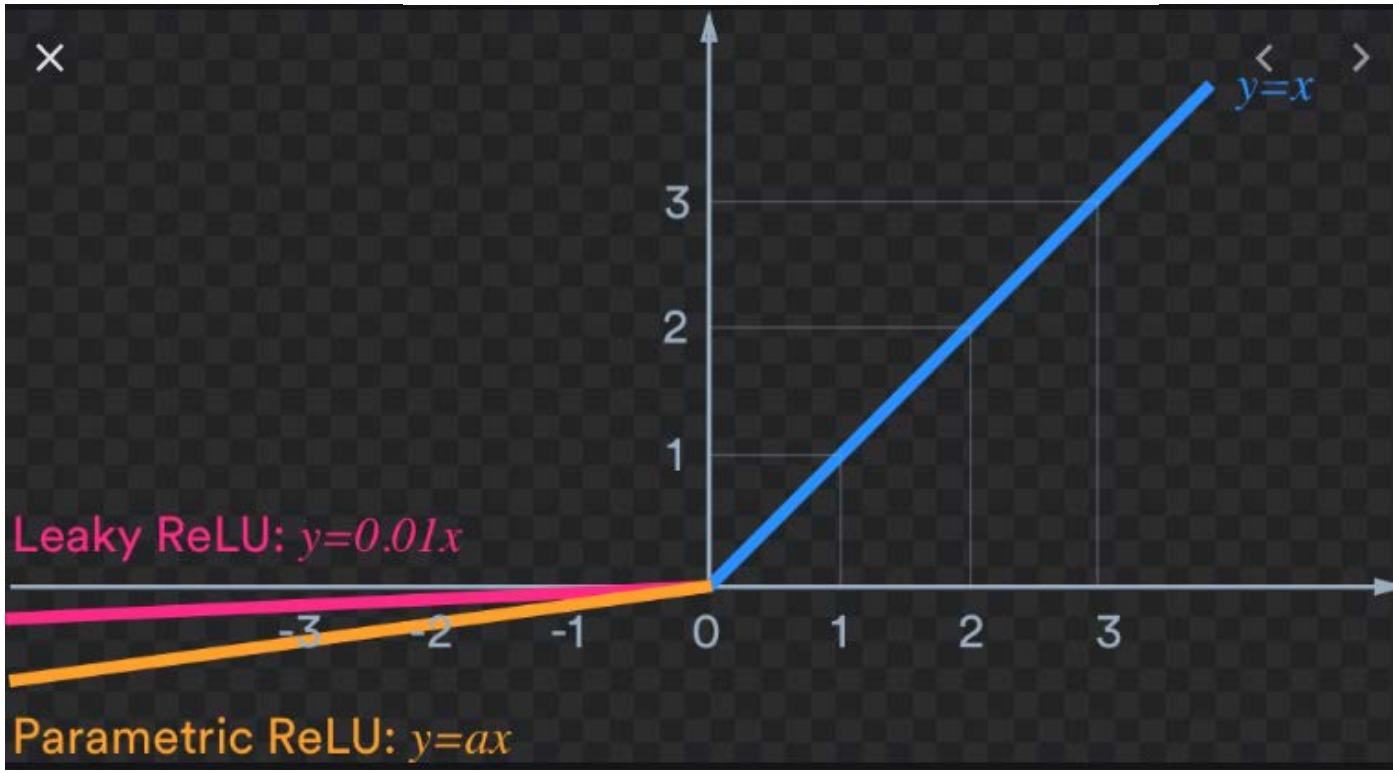
Figure 9.6: *Efficiency of edge detection.* The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all of the vertically oriented edges in the input image.

2nd stage: Detector stage with Activation function.

- After, performing several convolutions in parallel to produce a set of linear activations, In the second stage, each linear activation is run through a nonlinear activation function.
- We applies a Relu function to the convolved feature, in order to **introduce nonlinearity** into the model.

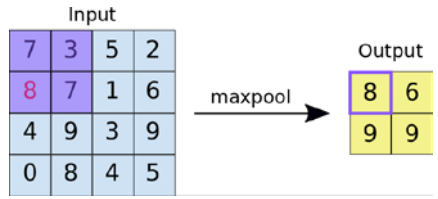




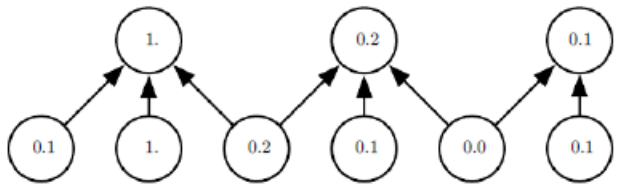


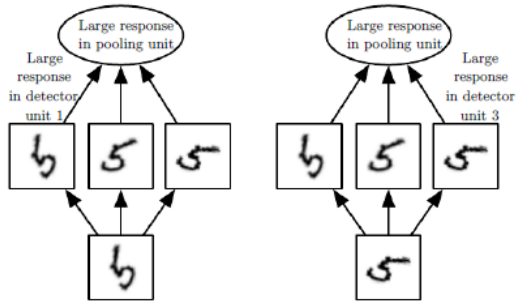
The ReLU function and its variations.

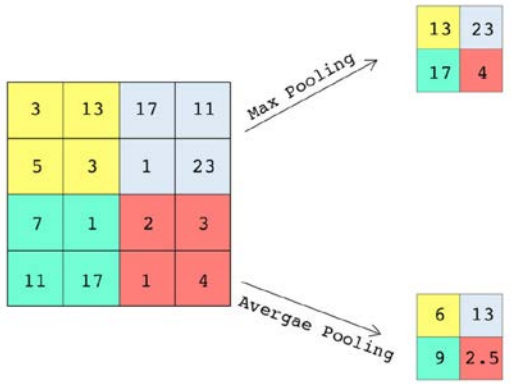
3rd stage: Pooling



- Max pooling operates in a similar fashion to convolution. We slide over the feature map. For each tile, the maximum value is output to a new feature map, and all other values are discarded.
- We downsample the convolved feature (to save computing time), reducing the number of dimensions of the feature map, while still preserving the most critical feature information.
- Max pooling operations take two parameters:
  - **Size** of the max-pooling filter (typically 2x2 pixels)
  - **Stride**: the distance, in pixels, separating each extracted tile. Unlike with convolution, where filters slide over the feature map pixel by pixel, in max pooling, the stride determines the locations where each tile is extracted. For a 2x2 filter, a stride of 2 specifies that the max pooling operation will extract all nonoverlappin[illegible] the feature map.

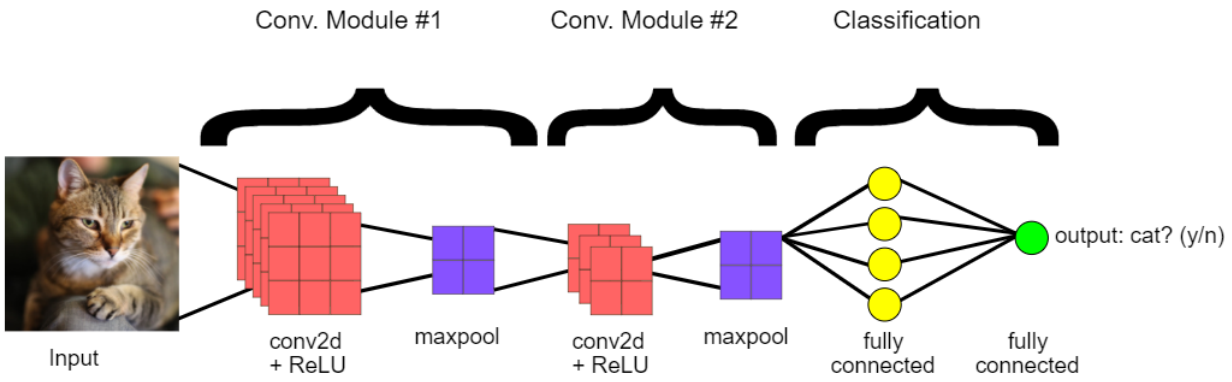






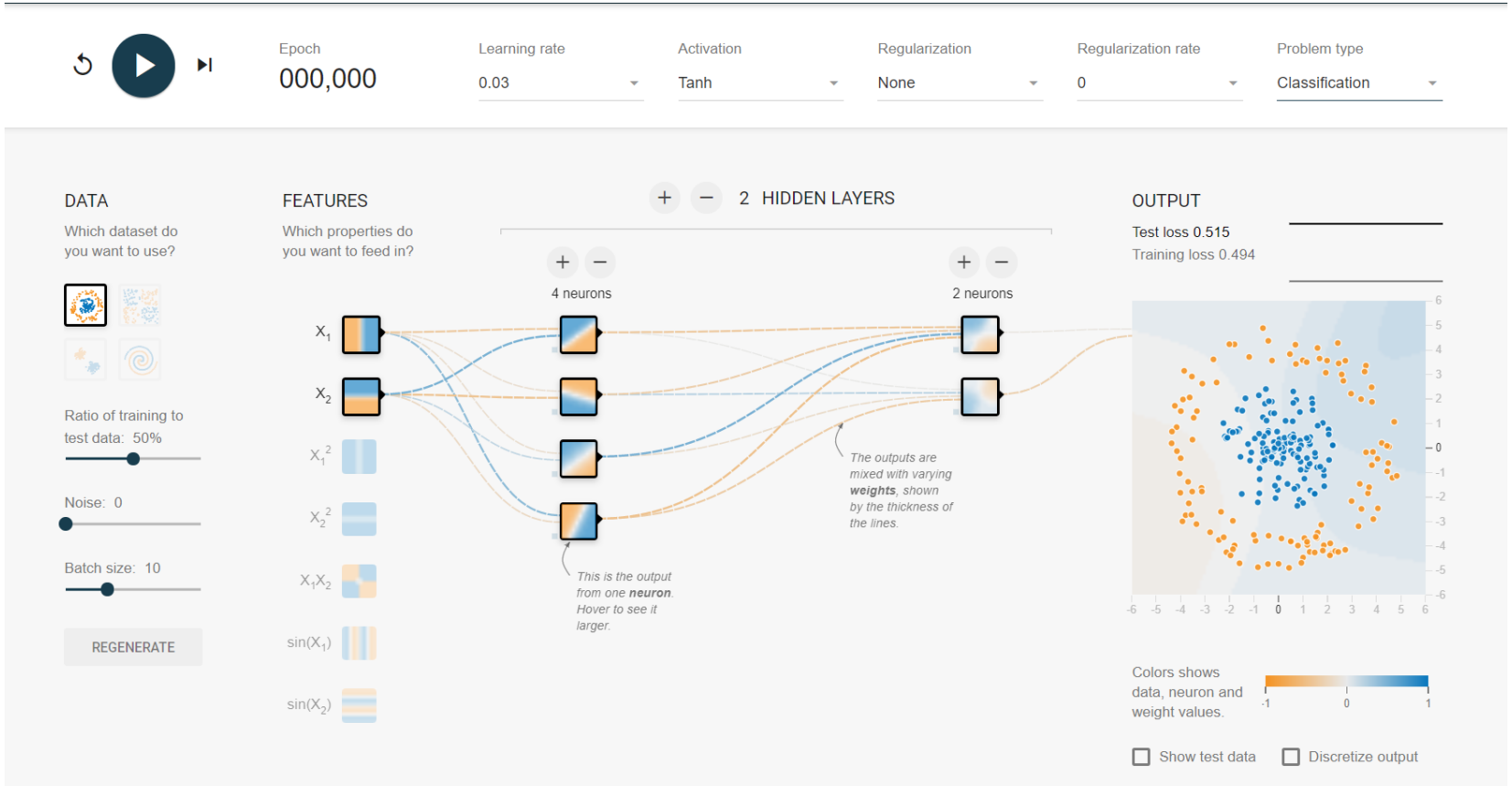
4) Fully Connected Layers

- At the end of a convolutional neural network are one or more fully connected layers (when two layers are "fully connected," every node in the first layer is connected to every node in the second layer). Their job is to **perform classification** based on the features extracted by the convolutions. Typically, the final fully connected layer contains a **softmax activation function**, which outputs a probability value from 0 to 1 for each of the classification labels the model is trying to predict.



Example 1- classification , regression of data points

- http://playground.tensorflow.org/



Example 2 Face Recognition by MultiTask Cascaded CNN (MT-CNN) and Faceness

- Reference 1: Zhang, Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks.

- Reference 2: FaceNet: A Unified Embedding for Face Recognition and Clustering

Overal:

- MT-CNN does Alignment in real time, Faceness does classification.

- The MT-CNNs consist of three Networks:

❑ 1st , it produces candidate windows quickly through a shallow CNN.

❑ 2nd , it refines the windows to reject a large number of non-faces windows through a more complex CNN.

❑ 3rd , it uses a more powerful CNN to refine the result and output facial landmarks positions.

- Thanks to this multi-task learning framework, the performance of the algorithm can be notably improved and the result is real time detection.
- After MT-CNN, the identified face with be fed into Faceness, which give us a feature vector for classification.

MTCNN

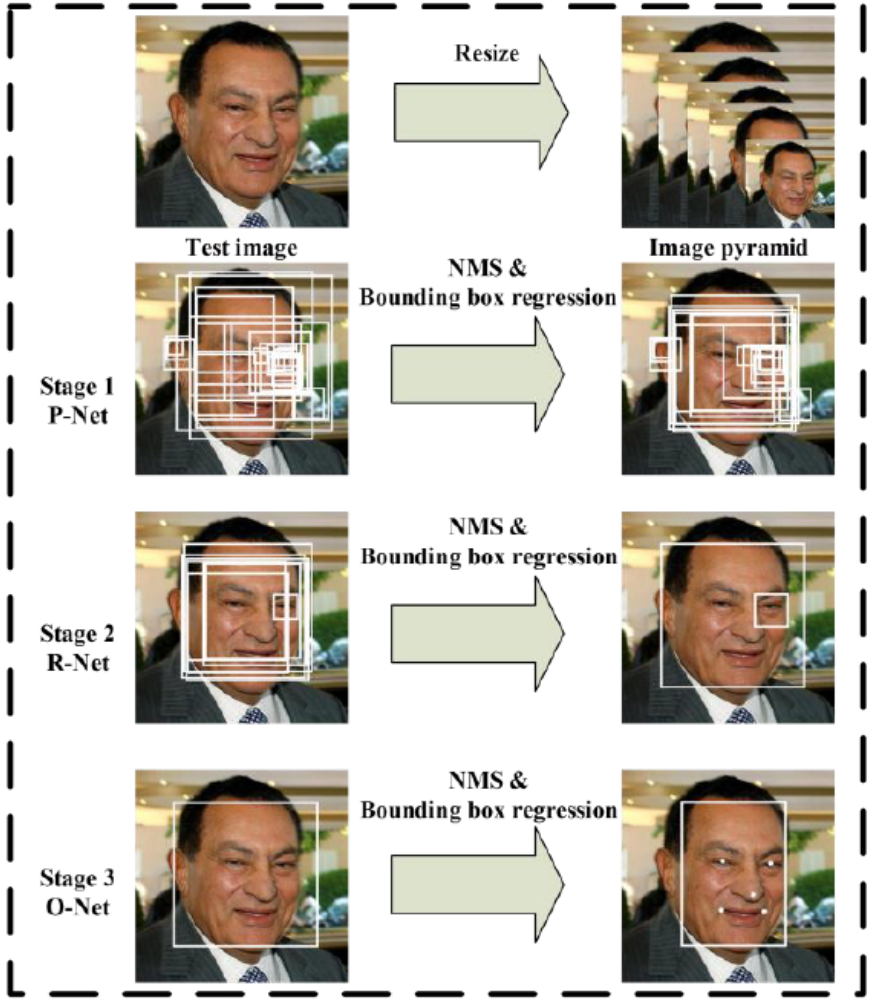
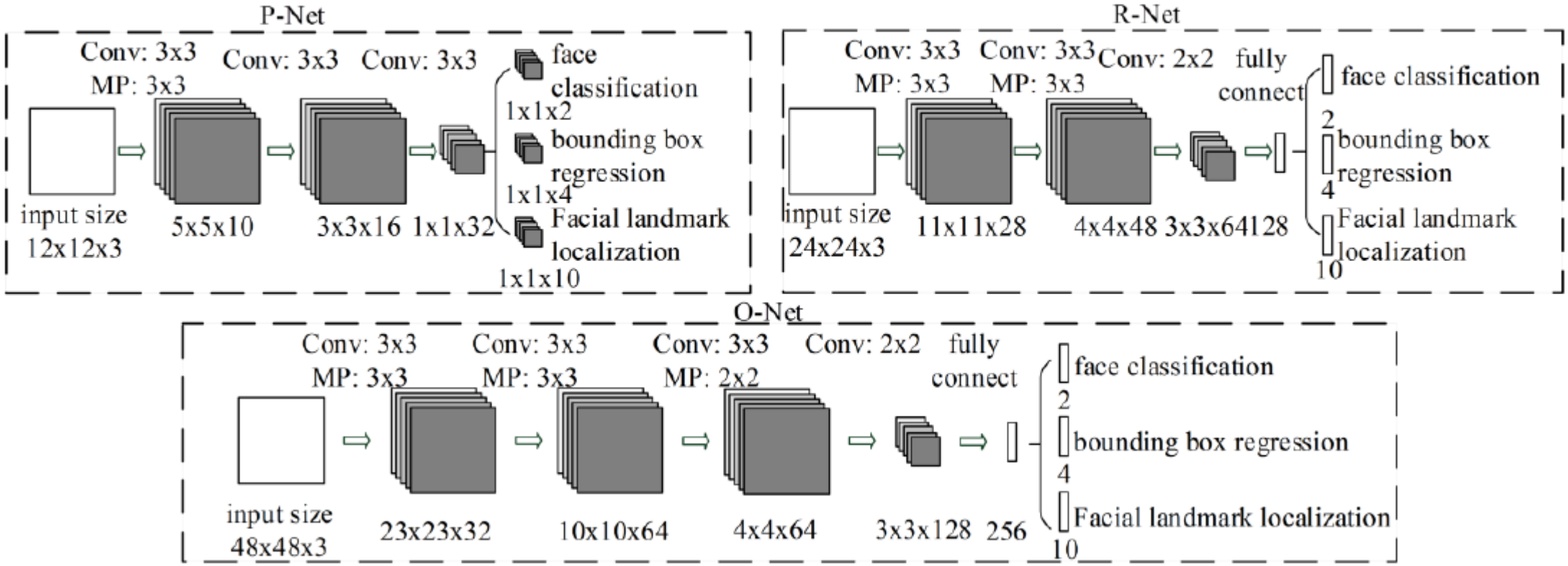


Fig. 1. Pipeline of our cascaded framework that includes three-stage multi-task deep convolutional networks. Firstly, candidate windows are produced through a fast Proposal Network (P-Net). After that, we refine these candidates in the next stage through a Refinement Network (R-Net). In the third stage, The Output Network (O-Net) produces final bounding box and facial landmarks position.

Stage 1: We exploit a fully convolutional network, called Proposal Network (P-Net), to obtain the candidate windows and their bounding box regression vectors. Use the estimated bounding box regression vectors to calibrate the candidates. Employ non-maximum suppression (NMS) to merge highly overlapped candidates.
Stage 2: all candidates are fed to another CNN, called Refine Network (R-Net), which further rejects a large number of false candidates, performs calibration with bounding box regression, and NMS candidate merge.

Stage 3: Output five facial landmarks' positions.

B. CNN Architectures



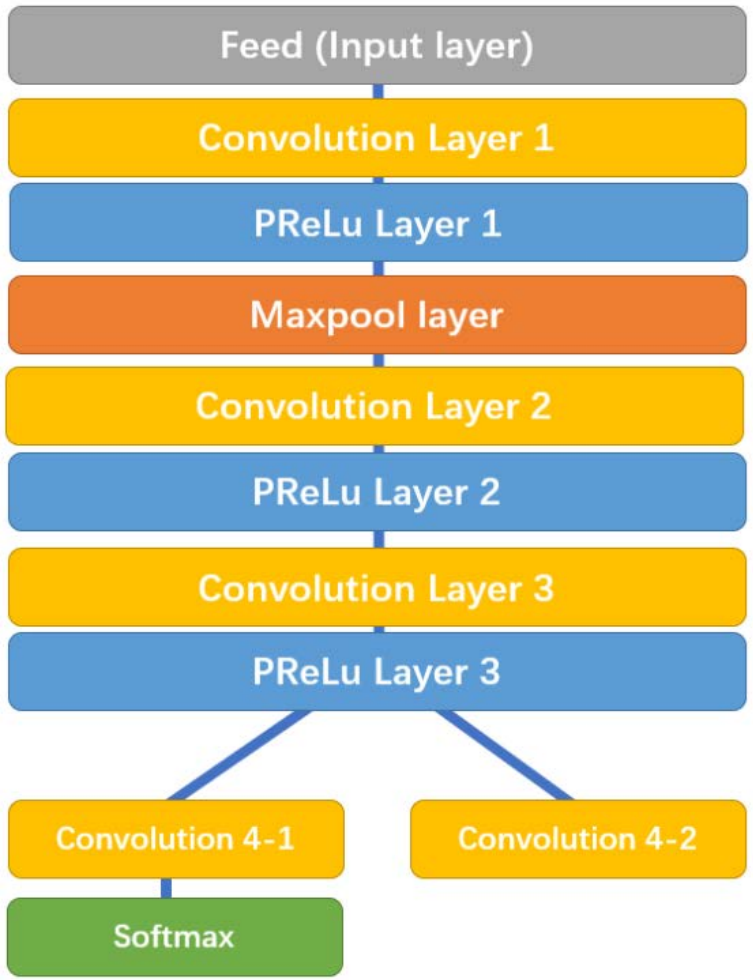


Image 5: P-Net

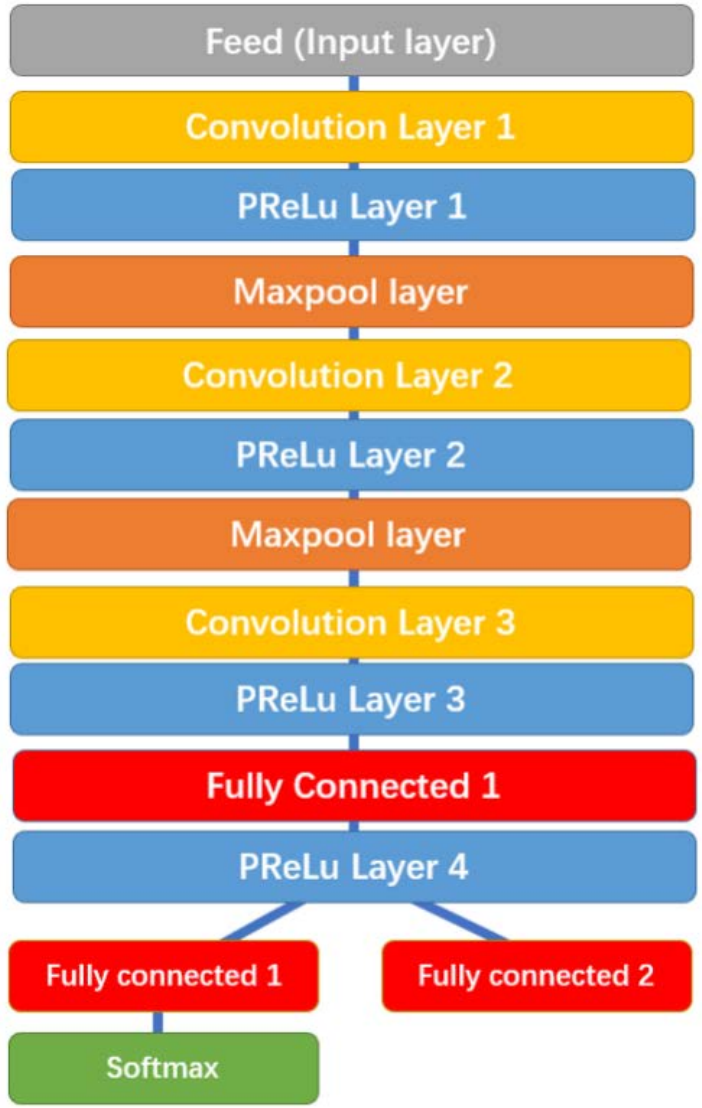


Image 6: R-Net

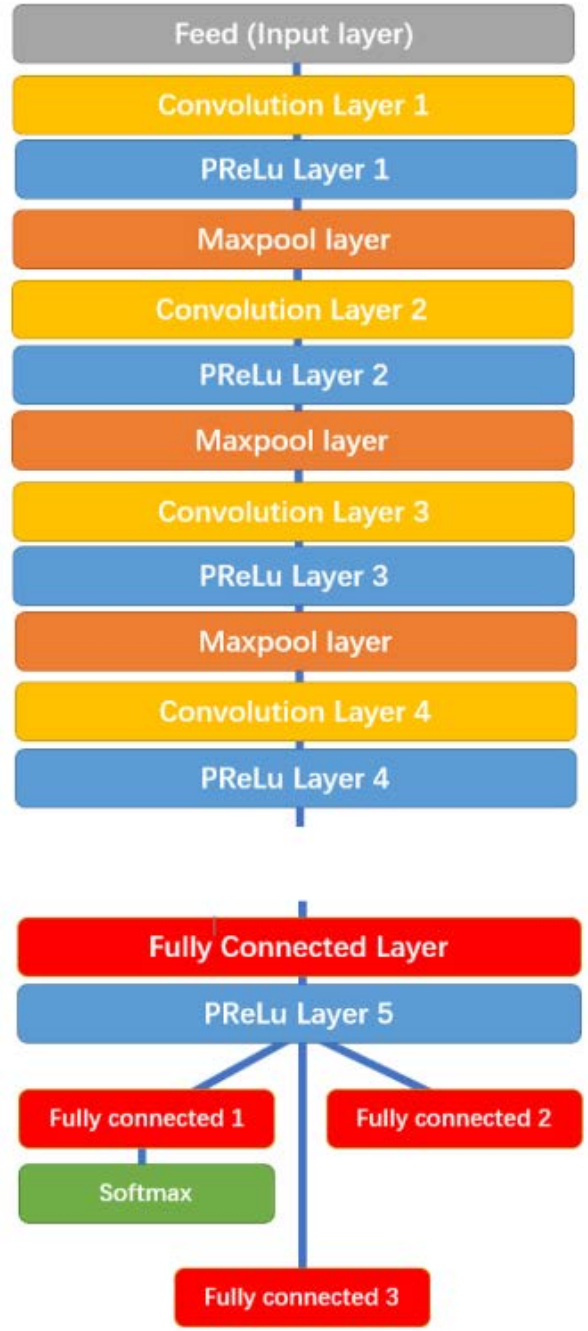


Image 7: O-Net

```
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)
```

```
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')
```

```
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')
```

Training.

We leverage three tasks to train our CNN detectors: face/non-face classification, bounding box regression, and facial landmark localization.

1) Face classification: The learning objective is formulated as a two-class classification problem. For each sample x_i, we use the cross-entropy loss:

$$L_i^{det} = -(y_i^{det} \log(p_i) + (1 - y_i^{det})(1 - \log(p_i))) \quad (1)$$

where p_i is the probability produced by the network that indicates a sample being a face. The notation $y_i^{det} \in \{0,1\}$ denotes the ground-truth label.

2) Bounding box regression: For each candidate window, we predict the offset between it and the nearest ground truth (i.e., the bounding boxes' left top, height, and width). The learning objective is formulated as a regression problem, and we employ the Euclidean loss for each sample x_i:

$$L_i^{box} = \left\| \hat{y}_i^{box} - y_i^{box} \right\|_2^2 \quad (2)$$

where $\hat{y}_i^{box}$ regression target obtained from the network and y_i^{box} is the ground-truth coordinate. There are four coordinates, including left top, height and width, and thus $y_i^{box} \in \mathbb{R}^4$.

3) Facial landmark localization: Similar to the bounding box regression task, facial landmark detection is formulated as a regression problem and we minimize the Euclidean loss:

$$L_i^{landmark} = \left\| \hat{y}_i^{landmark} - y_i^{landmark} \right\|_2^2 \quad (3)$$

where $\hat{y}_i^{landmark}$ is the facial landmark's coordinate obtained from the network and $y_i^{landmark}$ is the ground-truth coordinate. There are five facial landmarks, including left eye, right eye, nose, left mouth corner, and right mouth corner, and thus $y_i^{landmark} \in \mathbb{R}^{10}$.

4) Multi-source training: Since we employ different tasks in each CNNs, there are different types of training images in the learning process, such as face, non-face and partially aligned face. In this case, some of the loss functions (i.e., Eq. (1)-(3)) are not used. For example, for the sample of background region, we only compute L_i^{det}, and the other two losses are set as 0. This can be implemented directly with a sample type indicator. Then the overall learning target can be formulated as:

$$\min \sum_{i=1}^{N} \sum_{j \in \{det, box, landmark\}} \alpha_j \beta_i^j L_i^j \quad (4)$$

where N is the number of training samples. α_j denotes on the task importance. We use ($\alpha_{det} = 1, \alpha_{box} = 0.5, \alpha_{landmark} = 0.5$) in P-Net and R-Net, while ($\alpha_{det} = 1, \alpha_{box} = 0.5, \alpha_{landmark} = 1$) in O-Net for more accurate facial landmarks localization. $\beta_i^j \in \{0,1\}$ is the sample type indicator. In this case, it is natural to employ stochastic gradient descent to train the CNNs.

5) Online Hard sample mining: Different from conducting traditional hard sample mining after original classifier had been trained, we do online hard sample mining in face classification task to be adaptive to the training process.

In particular, in each mini-batch, we sort the loss computed in the forward propagation phase from all samples and select the top 70% of them as hard samples. Then we only compute the gradient from the hard samples in the backward propagation phase. That means we ignore the easy samples that are less helpful to strengthen the detector while training. Experiments show that this strategy yields better performance without manual sample selection. Its effectiveness is demonstrated in the Section III.

References:

- Refercence: Intro. CNN by Google :https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks
- Goodfellow, et al. Deep Learning. Chapter 9, Convolutional Networks.
- https://aishack.in/tutorials/image-convolution-examples/
- Zhang, Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks.
- FaceNet: A Unified Embedding for Face Recognition and Clustering