Face Recognition in the Presence of Non-Gaussian Noise

Oleg Rudenko[a], Oleksandr Bezsonov[a] and Denys Yakovliev[b]

[a] *Kharkiv National University of Radio, Nauky Ave. 14, Kharkiv, 61166, Ukraine*
[b] *Simon Kuznets Kharkiv National University of Economics , Nauky Ave.9, Kharkiv, 61166, Ukraine*

Abstract

A method for detecting a face and recognizing a person or a group of persons at pictures or videos that can be corrupted by non-gaussian noise using different architectures of convolutional neural networks is proposed. The technical details of building a deep learning-based face recognition system are discussed. The test results confirm the prospects of using the developed method for solving tasks of face detection and recognition.

Keywords 1
Convolutional neural network, learning algorithm, facial boundaries, marker, measure of similarity

1. Introduction

Visual pattern recognition is one of the most important components of information management and processing systems, automated systems and decision-making systems. Tasks related to the classification and identification of objects, phenomena and signals, characterized by a finite set of certain properties and characteristics arise in areas such as robotics, information retrieval, monitoring and analysis of visual data, and are the subject of artificial intelligence. The main task of artificial intelligence is to build intelligent information systems that would have a level of effectiveness in solving informal problems that can be compared with human capabilities or those that exceed them.

When interacting with other people, a person's face is an important source of information. Facial expressions, gestures when talking, head movements are a convenient and natural way to convey information. The inability of the computer on the one hand to perceive, and on the other hand to reproduce the natural human ways of communication complicates the transmission and perception of information when working with a computer. To achieve the goal of computer recognition of head movements, facial expressions, it is necessary to implement stable algorithms for analysis and classification of digital images of human faces keeping in mind that they can be corrupted by noisy pixels.

Such algorithms can perform a wide range of commercial and non-commercial tasks. To date, there is no algorithm that could identify a human face with 100% accuracy. However, it is already possible to implement programs for detecting faces, which can be successfully used in education, medicine, security and safety, computer games, advertising and other areas.

The purpose of this work is the development and testing of a method for detecting faces and recognizing a person or a group of persons at pictures and videos that can be corrupted by non-gaussian noise based on the use of convolutional neural networks.

CMIS-2021: The Fourth International Workshop on Computer Modeling and Intelligent Systems, 27 of April, 2021, , Zaporizhzia, Ukraine
EMAIL: oleh.rudenko@nure.ua (O. Rudenko); oleksandr.bezsonov@nure.ua (O. Bezsonov); denys.yakovliev@hneu.net (D. Yakovliev);
ORCID: 0000-0003-0859-2015 (O. Rudenko); 0000-0001-6104-4275 (O. Bezsonov); 0000-0002-6369-6223 (D. Yakovliev);

2. Recognition stages

It is generally assumed that full face recognition consists of four main steps:

1. Face detection
2. Preliminary processing
3. Isolation of signs
4. Classification

The first step is to detect faces in the image, regardless of scale and location. To do this, an advanced filtering procedure is often used to distinguish locations representing faces and filter them using some classifiers. It is noteworthy that all changes in displacement, scaling and rotation must be processed during the face detection phase. Note that facial expressions and changing hairstyles or smiling and frowning faces still present difficulties at the stage of pattern recognition [1].

While the accuracy and speed of face detection systems has improved, the two biggest challenges remain to some extent. Face detectors are essential to deal with large and complex variations of facial changes and effectively distinguish between faces and non-faces in unrestricted environments. In addition, large differences in face position and size in a large search space create problems that reduce detection efficiency [2]. This requires a trade-off between high detection accuracy and computational efficiency of the detection procedure.

In the next step, a system based on an anthropometric dataset predicts the approximate location of key features such as eyes, nose and mouth. The entire procedure is repeated to predict small features in relation to major features and is checked with collocation statistics to reject any misplaced features.

Highlighted anchor points are generated as a result of geometric combinations on the face image, and then the actual recognition process begins. It does this by finding a local representation of the facial appearance at each of the anchor points. The presentation scheme depends on the used approach.

Feature extraction usually occurs immediately after face detection and can be considered one of the most important steps in face recognition systems, since their effectiveness depends on the quality of the extracted features. This is because the facial landmarks are identified by a given network that determines how accurately features are represented. Traditional landmark locators are model-based, while many recent methods are based on cascade regression [3].

After feature extraction, face recognition is performed. The next stage of classification is simply performed using the general nearest neighbor technique, while more specific algorithms are developed for the preprocessing stage. This step usually consists of cropping, zooming, and aligning the detected faces. Unless sophisticated techniques are required for cropping and zooming, alignment is not an easy problem to solve. In practice, alignment consists in detecting a series of landmarks on the face (nose, eyes, etc.) and then transforming the face pictures so that the position of these landmarks is constant.

3. Convolutional neural networks for face recognition

Significant progress has been made in face detection and recognition through the use of convolutional neural networks (CNN).

The first CNN framework, known as LeNet-5, was developed in 1990 to classify handwritten digits by recognizing visual patterns in image pixels without the need for preprocessing [4]. In [5], a neural network used for vertical, frontal face recognition in grayscale was presented for the first time, which, although quite simple as for today's architectures, is comparable in accuracy to modern methods of that time.

Because face recognition differs from object recognition in that it requires alignment before extraction, this is reflected in the differences between CNNs designed for face recognition and those used for object recognition.

Because of the proven ability of deep neural network (DCNN)-based systems to outperform human performance in face-checking tasks, research in this area has skyrocketed.

Face detection and recognition research is currently focusing on DCNNs, which have demonstrated impressive accuracy on highly complex databases such as the WIDER FACE dataset [6] and MegaFace Challenge [7], as well as older databases such as Labeled Faces in the Wild (LFW) [8].

The growth in deep neural network research has been accompanied by the emergence of many deep learning frameworks and the development of the Caffe, TensorFlow, Torch, MXNET, and Theano frameworks that use platforms such as CUDA and libraries such as cuDNN. Finally, these frameworks can be used with a number of programming languages such as C ++, Python, or Matlab.

Rapid progress in this direction has been driven by the increase in the availability of powerful GPUs and improvements in CNN's architecture for real-world applications. In addition, the development of large annotated datasets and a better understanding of nonlinear mapping between input images and class labels have contributed to increased research interest in these networks. DCNNs are very efficient due to their ability to approximate non-linear functions. Their significant disadvantage is high computational costs due to the presence of intensive convolution and nonlinear operations [9]. However, DCCN is expected to have a bright future, and is currently being developed by such large corporations as Google, Facebook and Microsoft [10].

Using CNN in face recognition tasks consists of two main stages: training and inference. Training is a global optimization process that involves identifying network parameters by observing huge datasets. Inference essentially involves deploying a trained CNN to classify observed data [11]. The training process includes minimization the loss function to determine the parameters of the network and the number of layers required, and the organization of connections between the layers.

CNN face recognition systems are characterized by the training data used to build the model, the network architecture and settings, and the type of the loss function [12]. DCNNs have the ability to learn highly discriminatory and invariant representations of objects when trained on very large datasets. Training is reduced to the application of optimization algorithms to minimize the loss function. In this case, the role of the loss function is to determine the forecast error, and its choice depends on the problem being solved (regression, classification, etc.). Different loss functions will produce different error values for an identical forecast and, thus, largely determine the network performance. To minimize the error, a backpropagation algorithm is used, the weights are adjusted (corrected) by using some algorithm.

Modern face recognition systems that use DCNN implement deep feature extraction and similarity comparison, which involves conversion of test images to deep representations and computation of Euclidean or cosine distance.

4. Convolutional neural network architecture

Initially, the structure of the convolutional neural network was created taking into account the structural features of some parts of the human brain that are responsible for vision. The development of such networks is based on three mechanisms:

- local perception;
- formation of layers in the form of a set of feature maps (shared weights);
- subsampling.

According to these mechanisms to build a convolutional neural network three main layers are applied: convolution, pooling (or subsampling), fully connected layer.

4.1. Convolution Layer

The convolution formula for the l-th $(l = 1,...,L)$ layer of the network, which looks like [13]

$$x_{ij}^{l} = \sum_{a=-\infty}^{\infty} \sum_{b=-\infty}^{\infty} w_{ab}^{l} \cdot y_{(is-a)(js-b)}^{l-1} + b^{l}, \quad (1)$$

reflects the movement of the nucleus w^l from the input image or feature map for this layer y^{l-1} ... Here $y_{ij}^{l-1} = f(x_{ij}^{l-1})$ – image after the $(l-1)$-th layer; $f(\bullet)$ – used activation function; b^l – offset. i, j, a, b – indices of elements in matrices, s – the size of the convolution step.

As can be seen from (1), convolution operations are performed for each element i, j of image matrices x^l.

Convolution preserves spatial relationships between pixels.

Each convolutional layer is followed by a downsampling (subsampling) layer, or a computational layer, which serves to reduce the image dimension by local averaging the neuron output values.

4.2. Subsampling layer (pooling, MAX-pooling)

The subsampling layer reduces the scale of the planes by locally averaging the neuron outputs. Thus, a hierarchical organization is achieved. Subsequent layers extract more general characteristics less dependent on image distortion.

The pooling layer is described by the expression

$$x_{ij}^l = \sum_{a=-\infty}^{\infty} \sum_{b=-\infty}^{\infty} \beta^l down\left(w_{ab}^l \cdot y_{(is-a)(js-b)}^{l-1} + b^l\right), \quad (2)$$

where $down(\bullet)$ – the pooling function. This function adds blocks of the input image, thus reducing the dimension of the output image of the layer. In this case, any output map is set by two offset parameters: multiplicative β^l and additive b^l.

The difference between the subsampling layer and the convolution layer is that the regions of neighboring neurons overlap, which does not happen in the subsampling layer.

The pooling layer operates independently of the slice depth of the input data and scales the volume spatially using a maximum function.

In the architecture of the convolutional network, it is generally accepted that the presence of a feature is more important than information about its location. Therefore, from several neighboring neurons in the feature map, the maximum is selected and its value is considered one neuron in the feature map of a lower dimension.

In addition to the maximum subsampling, pooling layers can perform other functions, such as an averaging subsampling or even an L2-normalized subsampling.

4.3. Non-linear activation layer

On these layers within the network, a non-linear activation function is applied to all input values $f(\bullet)$ and the result is sent to the output. Thus, the activation layer does not change the size of the entrance.

Usually, due to significant positive properties, ReLu is used for hidden layers. (ReLU(x) = max(0, x)) and its various modifications (Leaky ReLU, Parametric ReLU, Randomized ReLU), and for fully connected layer – SoftMax function (when solving classification problems) $f_j^L = e^{x_j^L} \left(\sum_{i=1}^{N^L} e^{x_i^L} \right)^{-1}$ or linear (for regression problems).

4.4. Dropout layer

Various regularization methods are used to avoid overfitting the network.

Dropout is a simple and effective method of regularization and consists in the fact that in the process of training a network from its aggregate topology, a subnet is repeatedly randomly allocated,

i.e. some neurons are turned off from the process and the next update of the weights occurs only within the dedicated subnet. Thus, weights change only in the remaining neurons. Each neuron is dropped from the aggregate network with a certain probability, which is called the dropout ratio.

This layer reduces the time of one training epoch due to the smaller number of parameters to be optimized, and also makes it possible to better deal with network overfitting in comparison with standard regularization methods.

4.5. Normalizing layer

On this layer, the standard normalization of the inputs occurs (the sample mean of their values is subtracted, and the result is divided by the root of the sample variance). Sample values are calculated taking into account the values at the inputs of this layer at previous training iterations. This approach allows you to increase the speed of network learning and improve the final result.

4.6. Fully connected layer

This layer is a conventional multilayer perceptron, the purpose of which is classification. It simulates a complex nonlinear function, optimization of which improves the recognition quality.

The neurons of each map of the previous subsample layer are associated with one neuron of the hidden layer. Thus, the number of neurons in the hidden layer is equal to the number of maps in the subsample layer.

As in conventional neural networks, in a fully connected layer, neurons are connected to all activations in the previous layer. Their activations can be calculated by multiplying matrices and applying an offset.

The difference between fully connected and convolutional layers is that the neurons of the convolutional layer

1) are connected only to the local input area;
2) can share parameters.

4.7. Choice of training criterion

CNN training is an iterative process. Each iteration calculates the network outputs for one (or more) samples in the training set, and adjusts the network weights to reduce the error between the actual network output. $(y_{i,p}^{L})(i = 1,..., N^{L})$ and the target output for a given sample $d_{i,p}$. Therefore, training is reduced to minimizing some functionality.

In practice, a quadratic function, cross entropy, or some combined functional is used as a criterion for the error function.

A backpropagation neural network (BPNN) is a multi-layer feedforward neural network that uses a supervised training algorithm known as an error backpropagation algorithm. Errors accumulated on the output layer are propagated back to the network to correct the weights. A conventional MP, which consists of three types of layers: input, hidden and output has no backward computation other than the operations used in training. All operations are performed in the forward direction during simulation.

5. Training of convolutional neural network

CNN training is similar to training of any direct propagation ANN and consists in correcting its weight parameters based on minimizing some selected loss function, which are usually used as a quadratic function or cross entropy.

To train convolutional neural networks, both the standard backpropagation method and its various modifications can be used. The derivation of the backpropagation algorithm for CNN training is discussed in sufficient detail in [14]. This method is based on the stochastic gradient descent algorithm (SGD).

In practice, the most common neural network learning algorithm is usually used, based on the gradient descent method (the backpropagation error method) and its modifications: SGD, Adam, AdaGrad, AdaDelta, etc. [15].

6. Training neurons of the output (fully connected) layer

To train the neurons of this layer, a gradient backpropagation algorithm is used

$$\theta(k+1) = \theta(k) - \eta \nabla_{\theta} J(\theta(k)), \tag{3}$$

where θ – network parameters (elements of weight matrices, displacements, angles of inclination of activation functions, etc.); $J(\theta(k))$ – objective function (loss function E or C); η – the parameter of the learning rate.

6.1. Training neurons when choosing a quadratic function E

If the quadratic function E is chosen as the network loss function, and the sigmoidal function is chosen as the activation function of neurons, then to adjust the weights matrix of the L-th layer following equations are used

$$\nabla_{w_{ki}^L} J(w_{ki}^L) = \frac{\partial E}{\partial w_{ki}^L} = \frac{\partial E}{\partial y_i^L}\frac{\partial y_i^L}{\partial x_i^L}\frac{\partial x_i^L}{\partial w_{ki}^L} = \delta_i^L \cdot \frac{\partial x_i^L}{\partial w_{ki}^L} = \delta_i^L \cdot y_k^{L-1}, \tag{4}$$
$$\forall i \in (0,...,N^L), \forall k \in (0,...,N^{L-1}),$$

where

$$\delta_i^L = \frac{\partial E}{\partial y_i^L}\frac{\partial y_i^L}{\partial x_i^L};$$
$$\frac{\partial E}{\partial y_i^L} = y_i^L - d_i; \tag{5}$$
$$\frac{\partial y_i^L}{\partial x_i^L} = y_i^L(1 - y_i^L);$$

d_i – required value of the i-th output.

If, however, as the activation function of neurons SoftMax is selected, then in (4) one should take $\frac{\partial y_i^L}{\partial x_i^L} = y_i^L(1 - y_i^L)$, and besides, $\frac{\partial y_i^L}{\partial x_j^L} = -y_i^L y_j^L$, because $\frac{\partial E}{\partial x_i^L} = \sum_{j=1}^{N^L} \frac{\partial E}{\partial y_j^L}\frac{\partial y_j^L}{\partial x_i^L}, i = (1,...,N^L)$.

Similarly, one can get the procedure for adjusting the offsets for neurons of a fully connected layer

$$\frac{\partial E}{\partial b_i^L} = \frac{\partial E}{\partial y_i^L}\frac{\partial y_i^L}{\partial x_i^L}\frac{\partial x_i^L}{\partial b_i^L} = \delta_i^L \cdot \frac{\partial x_i^L}{\partial b_i^L} = \delta_i^L. \tag{6}$$

When training neurons of other (hidden, l-x, $l = 1,...,L-1$) layers it is needed to calculate gradients

$$\frac{\partial E}{\partial y_k^l} = \sum_{i=1}^{N^{L-1}} \delta_i^l \cdot \frac{\partial x_i^l}{\partial y_k^{l-1}} = \sum_{i=1}^{N^{L-1}} \delta_i^{l+1} \cdot w_{ki}^{l+1}, \tag{7}$$
$$\forall i \in (1,...,N^L) \forall k \in (1,...,N^{L-1}).$$

Here w_{ij}^{l+1} – the weight of the connection between the neuron i of the current (hidden) layer and neuron j of the next layer.

6.2. Training neurons on selection cross entropy function as a loss function

When choosing the cross entropy as the loss function, the training procedure for the output (fully connected) layer of the CNN will take the form

$$\theta(k+1) = \theta(k) - \eta \nabla_\theta C(\theta(k)), \tag{8}$$

where $\theta = (w, b)$.

Calculating the partial derivatives with respect to the tunable parameters, we have (here it is taken into account that for the output layer $y_i^L = f_i^L$)

$$\nabla_w C = \frac{\partial C}{\partial w_{ik}^L} = \frac{\partial C}{\partial y_i^L}\frac{\partial y_i^L}{\partial w_{ik}^L} = f_k^{L-1}(f_k^{L-1} - d_i); \tag{9}$$

$$\nabla_b C = \frac{\partial C}{\partial b_i^L} = \frac{\partial C}{\partial y_i^L}\frac{\partial y_i^L}{\partial b_i^L} = f_i^L - d_i; \tag{10}$$

$$\nabla_f C = \frac{\partial C}{\partial f_j^L} = -\frac{\partial}{\partial f_j^L}\left(\sum_{i=1}^n d_i \ln f_i^L\right) = -\frac{d_j}{f_j^L} = f_i^L - d_i. \tag{11}$$

When calculating the gradient, the following derivatives are used:

$$\frac{\partial f_i^L}{\partial x_j^L} = \begin{cases} \dfrac{e^{x_j^L}}{\sum_{k=1}^n e^{x_k^L}} - \left(\dfrac{e^{x_j^L}}{\sum_{k=1}^n e^{x_k^L}}\right)^2 \; i = j; \\ \dfrac{e^{x_i^L} e^{x_j^L}}{\left(\sum_{k=1}^n e^{x_k^L}\right)^2} i \neq j; \end{cases} = \begin{cases} f_i^L(1 - f_i^L), i = j; \\ -f_i^L f_j^L \end{cases} \tag{12}$$

$$\frac{\partial C}{\partial x_i^L} = -\frac{\partial}{\partial x_i^L}\left(\sum_{j=1}^n d_j \ln f_i^L\right) = \sum_{j=1}^n \frac{\partial C}{\partial f_j^L}\frac{\partial f_j^L}{\partial x_i^L} = y_i^L - d_i^L. \tag{13}$$

6.3. Training neurons in the subsampling layer

The peculiarity of this layer is that it sits in front of both fully connected and convolutional layers. In the first case, it has the same neurons and connections as a fully connected one. Therefore δ is calculated in the same way as in a hidden layer. If the subsampling layer is in front of convolutional layer, δ is computed by the inverse convolution method using the rotation by 180^0 $\left(rot_{180^o}\left\{\delta_{ij}^l\right\}\right)$ [16–18].

6.4. Training of neurons of the convolutional layer

The backpropagation algorithm is also used to train the neurons of this layer.

Core is a filter that slides over the entire image and finds its features anywhere, i.e. provides invariance to displacements.

Formula for updating the convolution kernel w_{ab}^l has the form [16–18]

$$\frac{\partial E}{\partial w_{ab}^l} = \sum_i \sum_j \frac{\partial E}{\partial y_{ij}^l}\frac{\partial y_{ij}^l}{\partial x_{ij}^l} \cdot y_{(is-a)(js-b)}^{l-1}, \forall a \in (-\infty, ..., +\infty) \forall b \in (-\infty, ..., +\infty). \tag{14}$$

Only one offset can be used for one feature map b^l, which is “connected” with all the elements of

this map. Accordingly, when correcting the value of this offset, all values from the map obtained during the back propagation of the error should be taken into account. In this case (when using one displacement for one feature map) we have

$$\frac{\partial E}{\partial b^l} = \sum_i \sum_j \frac{\partial E}{\partial y_{ij}^l} \frac{\partial y_{ij}^l}{\partial x_{ij}^l} \frac{\partial x_{ij}^l}{\partial b^l} = \sum_i \sum_j \frac{\partial E}{\partial y_{ij}^l} \frac{\partial y_{ij}^l}{\partial x_{ij}^l}. \quad (15)$$

As an alternative, one can take as many offsets b_{ij}^l for a separate feature map, as many elements are in this map, but in this case there will be too many displacement parameters (more than the parameters of the convolution kernels themselves). For this case $\frac{\partial E}{\partial b_{ij}^l} = \frac{\partial E}{\partial x_{ij}^l}$ ...

To train neurons in the convolution layer, the gradient procedure uses the derivative $\frac{\partial E}{\partial y_{ij}^{l-1}}$ calculated as follows:

$$\frac{\partial E}{\partial y_{ij}^{l-1}} = \sum_{i'} \sum_{j'} \frac{\partial E}{\partial y_{i'j'}^l} \frac{\partial y_{i'j'}^l}{\partial x_{i'j'}^l} \frac{\partial x_{i'j'}^l}{\partial y_{ij}^{l-1}} = \sum_{i'} \sum_{j'} \frac{\partial E}{\partial y_{i'j'}^l} \frac{\partial y_{i'j'}^l}{\partial x_{i'j'}^l} \cdot w_{(i's-i)(j's-j)}^l. \quad (16)$$

7. Histogram of oriented gradients (HOG)

Histogram of Oriented Gradients (HOG) – is a method of information evaluation of special points of the image, based on the calculation of the number of directional gradients in the space of these points. The basic idea of the algorithm is the assumption that the appearance and shape of the object in the image area can be described by the distribution of intensity gradients or the direction of the edges [19]. Such a description is carried out by dividing the image into cells and constructing histograms of directional gradients of cell pixels. The result of the algorithm is a descriptor, which includes a combination of the obtained histograms.

HOG algorithm contains following stages:

1. Gradient calculation.
2. Calculation of the histograms of the image cells.
3. Forming and normalizing descriptor blocks.
4. The final step is to classify the HOG descriptors using a training system with a teacher.

The result of the classifier's work is two object images with positive and negative weights of reference vectors. Positive weights mean that the features belong to the target class, and negative weights mean that the features belong to the background.

8. SSD network

The SSD method is based on such CNN architectures as Faster R-CNN and YOLO, however, the authors of [20] took into account their shortcomings, thanks to which SSD could achieve new peaks of accuracy and speed. The method is based on the Feed-forward convolutional network, which creates a finite set of limiting rectangular frames and quantitative estimates of the presence of objects of various classes in these frames, after which it produces the suppression of maximums ("Non-maximum suppression predictors"). The structure of the network is shown in Fig. 1 and can be divided into four functional parts:

1. Network input. It accepts a three-channel (color) image with a size of 300x300.

2. Backbone network. Some standard architecture for image classification is usually used as a backbone network. In this case it is VGG-16. However, the fully connected layer of VGG-16 is not used in SSD. This architecture is described in detail in [20].

3. Layer of additional features. These are convolutional layers and subsampling layers that generate feature maps of different resolution for detecting objects of different scales.

4. At this stage operations are performed to merge the outputs of various layers and suppress maximums to obtain the output of the network.

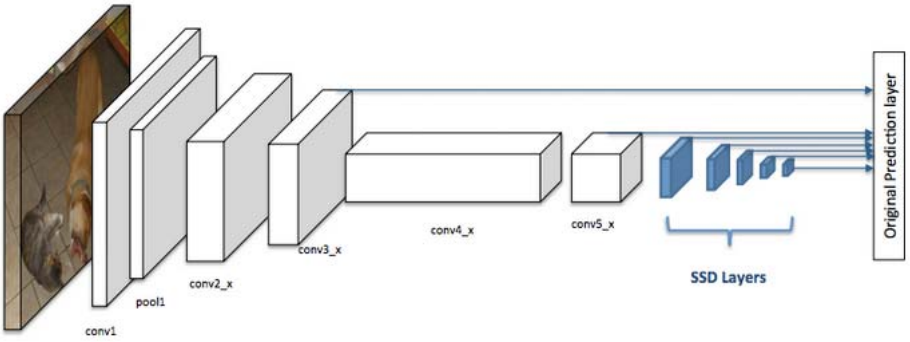

Figure 1: The architecture of the SSD network [20]

The SSD method has the following key features:

1. Detection of objects of various scales. Several convolutional layers of different sizes are sequentially added to the base network, which make it possible to obtain maps of signs of different resolutions and, accordingly, to predict objects at different scales (unlike, for example, YOLO, which receives only one map).

2. Convolutional predictors for detection. Each feature layer generates a finite set of assumptions about the class and position of the object using a set of convolutional filters. Each such convolutional filter has a kernel that determines the probability of belonging to the default frame offset class itself.

3. Default bounding frames. The default set of frames is linked to the cards of signs of different resolutions described in point 1: each cell of the card of signs corresponds to a frame from the set by default. For each cell of the feature map, estimates are determined according to the class of objects, as well as four offsets of the default frame relative to its initial position.

Combining these properties allowed efficiently sampling many different forms of the resulting bounding frames, which had a positive impact on the speed of the network.

9. MTCNN network

Of the neural network approaches in face detection, Multi-task Cascaded CNN (MTCNN) is especially significant. This network is described in details in [21].

10. RetinaFace Network

The architecture of the RetinaFace network [22] is shown in Fig. 2. This network is a single-stage face detector that detects faces through collaborative, supervised and self-guided multi-tasking learning. The network has several features that improve the ArcFace network's face detection performance and surpass most network architectures in the vast majority of tests for such systems:

- RetinaFace uses five additional key points on faces to dramatically improve facial recognition results;
- RetinaFace has a self-checking grid decoder branch to predict per-pixel information about 3D face shapes in parallel with existing controlled branches;
- By utilizing lightweight backbones, RetinaFace delivers performance that allows it to be run on the CPU, while most similar systems achieve such performance solely using the GPU.

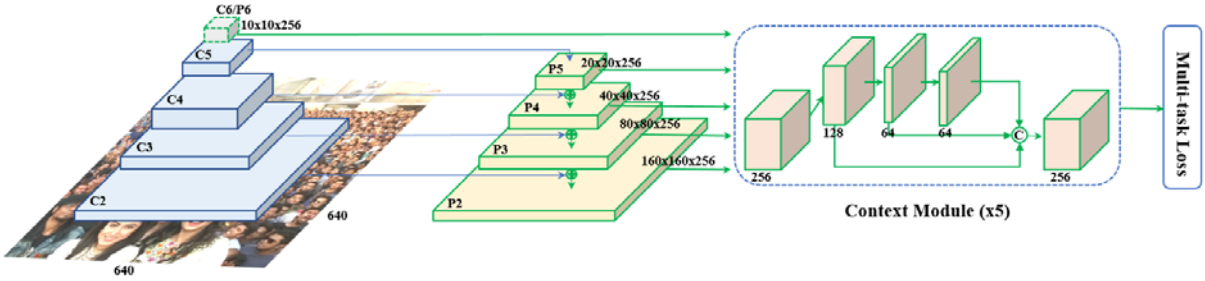

Figure 2: The architecture of the RetinaFace network [22]

11. Simulation

The face recognition system consists of the following steps:

1) Locate all the faces in the image.

It applies the location method to the given image and returns a list of bounding boxes of human faces at the image. The HOG method and CNN based methods (SSD, MTCNN, RetinaNet) are used as face detectors. HOG is less accurate than convolution networks models, but much faster on CPUs.

2) Face encoding (features extracting).

The method performs face landmark estimation and extract specific points from the found face. This will make it a lot easier to compare faces and not worry about the problem when faces turned in different directions look totally different for a vision system.

Found specific points are passed to the pre-trained CNN model, which generates a 128- or 256-dimension face encoding for each face at the image.

3) Matching the encoded faces with a database of known people.

Finally, the method compares the list of face encodings against to the candidate encoding to see if they match. Given the list of face encodings, compare them with the known face encoding and get the Euclidean distance for each comparison face. The distance indicates how similar the faces are. After that, return a list of True/False values indicating which known face encodings match the face encoding to check.

Experiment 1. A comparison was made between different face detectors in combination with different parameters of the RNET feature encoder. So the following detectors were used:

1. The Retina MNET model, which is a lightweight model for finding facial boundaries with fast coordinate regression. Given the found face boundaries, it returns 2d-106 face markers. A 192x192 image was fed to the network input.

2. The MTCNN model used to find facial boundaries and regression of facial markers. Given the found face boundaries, it returns 5 exact markers.

3. The SSD model, which is the fastest neural network model for object detection.

4. The RNET encoder accepts the locations of the faces, separates the face by the found coordinates. Using the resulting face landmarks (eye sockets – 2, nose tip – 1, lip edges – 2) aligns the picture to obtain a frontal view of the face. Then the result is reduced to 112x112 pixels, after which it is processed by the network. At the output of the network, we get a vector of face features with dimension 512.

To compare and identify the obtained representation of the encoder's face with the entire base of faces, a "cosine coefficient" is used – a binary similarity measure.

The results of the system operation on a test set of 500 images with various combinations of detection and recognition methods are shown in Table 1. Some selected images are presented in Fig. 3–4.

Experiment 2. Experiments were carried out to recognize faces in images with one face, a group of faces and in the presence of non-gaussian noise (salt-and-pepper). RetinaFace MNET network and the arc512d recognition method were used as a detector.

Some selected results are shown in Fig. 5.

The recognition accuracy for 500 test images was 94.7%, which indicates a high noise canceling ability of the method in question.

The results for other combinations of detectors and recognition methods are presented in Table 2.

Table 1
System performance results

Method	HOG	SSD	MTCNN	RetinaFace
fr128d	86.67%	90.67%	92.67%	96%
arc512d	87%	93.33%	95.67%	96.67%

Table 2
System performance results with noise presents

Method	HOG	SSD	MTCNN	RetinaFace
fr128d	73.46%	86.23%	90.65%	91.25%
arc512d	75.23%	88.54%	91.88%	94.70%

Figure 3: Results of recognition of single objects

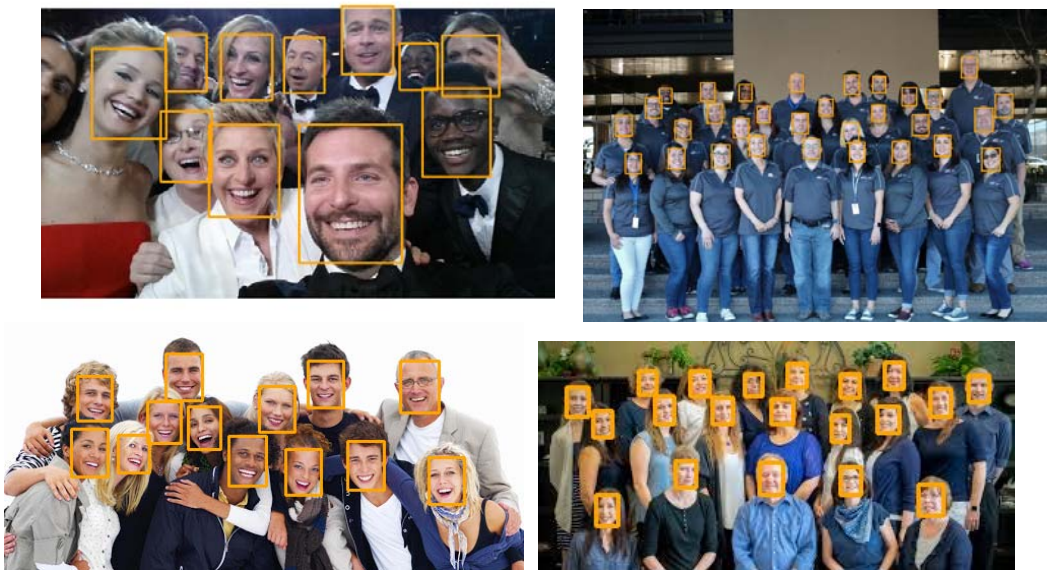

Figure 4: Results of recognition of a group of objects

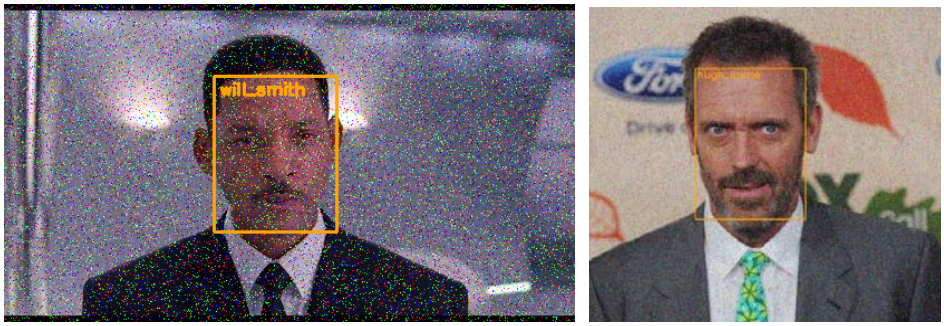

Figure 5: Results of recognition of noisy images

12. Conclusions

Analysis of the obtained results allows us to draw the following conclusions:

1. Increasing the number of more boxes leads to more accurate detection, while reducing its speed
2. The presence of MultiBox on several layers also leads to better detection, due to the fact that the detector works with several resolutions at the same time;
3. 80% of the time is required to perform computations on the VGG-16 core network. This means that with a faster and equally accurate network, the performance of an SSD can be significantly improved;
4. SSD mixes objects with similar categories (for example, animals). This is probably because cue points are common across multiple classes
5. The worst performance of the SSD network is observed for small objects, as they may not appear on all object maps. Some increase in network performance can be achieved by increasing the resolution of the input image.
6. The SSD network is quite effective for solving the problem of face detection both in the absence of interference and in the presence of non-gaussian interference.

13. Acknowledgements

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

14.References

[1] B. Yang, J. Yan, Z. Lei, S. Li, Aggregate channel eatures for multi-view face detection, in IEEE International Joint Conference on Biometrics, 2014, pp. 1-8.

[2] S. Zhang, et al., Detecting Face with Densely Connected Face Proposal Network, 2017, pp. 3-12.

[3] R. Ranjan, et al., Deep Learning for Understanding Faces: Machines May Be Just as Good, or Better, than Humans. Signal Processing Magazine, IEEE, 2018.35 (1), p. 66-83.

[4] Y. LeCun et al., Handwritten digit recognition with a back-propagation network, 1990.

[5] H.A. Rowley, S. Baluja, T. Kanade, Neural network-based face detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1998.20 (1), p. 23-38.

[6] S. Yang, et al., WIDER FACE: A Face Detection Benchmark, 2015.

[7] I. Kemelmacher, et al., The MegaFace Benchmark: 1 Million Faces for Recognition at Scale, 2016, pp. 4873-4882.

[8] G.B. Huang, et al., Labeled Faces in the Wild: A Database forStudying Face Recognition in Unconstrained Environments, in Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition, Marseille, France, 2008.

[9] S. Wu, et al., Funnel-structured cascade for multi-view face detection with alignment-awareness. Neurocomputing, v. 221, 2017, p. 138-145.

[10] A. Krizhevsky, I. Sutskever, G. Hinton., ImageNet classification with deep convolutional neural networks. Communications of the ACM, 2017.60 (6), p. 84-90.

[11] A. Parashar, et al., SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. ACM SIGARCH Computer Architecture News, v.45 (2), 2017, pp. 27-40.

[12] J. Deng, J. Guo, S. Zafeiriou. ArcFace: Additive Angular Margin Loss for Deep Face Recognition, 2018.

[13] Z. Zhang., Derivation of Backpropagation in Convolutional Neural Network (CNN), Technical Report. University of Tennessee, Knoxvill, TN, October 18, 2016. URL: http://web.eecs.utk.edu/~zzhang61/docs/reports/2016.pdf

[14] Y. LeCun, Y. Bengio, Convolutional networks for images, speech, and timeseries, The Handbook of Brain Theory and Neural Networks, 1995, pp. 255-258.
O. Rudenko, O. Bezsonov, K. Oliinyk, First-Order Optimization (Training) Algorithms in Deep LearningProceedings of the 4th International Conference on Computational Linguistics and Intelligent Systems (COLINS 2020). Volume I: Main Conference Lviv, Ukraine, April 23-24, 2020, pp. 921-935.

[15] Deep Learning for Image Processing Applications, Ed. Hemanth DY, Estrella VV, 2017, 273 p. URL: http://ebooks.iospress.nl/volume/deep-learning-for-image-processing-applications

[16] Deep learning tutorial, Stanford University. Autoencoders. URL: http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

[17] Convolutional network in python. Part 2. Derivation of formulas for training the model. URL: https://habr.com/ru/company/ods/blog/344116/

[18] P.Viola, M.J. Jones, and D.Snow, Detecting pedestrians using patterns of motion and appearance, The 9-th ICCV, Nice, France, v.1, 2003 , pp. 734-741.

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, SSD: Single Shot MultiBox Detector, 2016. URL: https://arxiv.org/abs/1512.02325.

[20] K. Simonyan, A.Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. URL: https://arxiv.org/abs/1409.1556v6

[21] K. Zhang, Z. Zhang, Z. Li, Y. Qiao, Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. URL: https://arxiv.org/pdf/1604.02878

[22] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, S. Zafeiriou, RetinaFace: Single-stage Dense Face Localisation in the Wild. URL: https://arxiv.org/abs/1905.00641