AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

FACULTY OF ELECTRICAL ENGINEERING, AUTOMATICS, COMPUTER SCIENCE AND BIOMEDICAL ENGINEERING

Projekt dyplomowy

Rozpoznawanie wieku oraz płci osób na fotografiach z wykorzystaniem uczenia głębokiego

Recognizing the age and sex of people in photos using deep learning

Author:	*Jan Zasadny*
Degree programme:	*Computer Science*
Supervisor:	*dr hab. Adrian Horzyk, prof. AGH*

Kraków, 2020

Serdecznie dziękuję Panu Promotorowi dr. hab Adrianowi Horzykowi za pomoc w pisaniu pracy inżynierskiej i poświęcony mi czas.

Contents

1. Introduction

Computer vision is an interdisciplinary scientific field that deals with extracting and analysing pieces of information from images and videos using computers. It is concerned with enabling humans to gain a better understanding of the world through automation and refinement of various processes employed by the human brain in order to extract useful information from visual data.

One of the most important visual data instances that we encounter in daily life are human faces. By taking a look at another human's face, we are able to identify them and, to a varying degree, estimate many different characteristics of that person, such as their age, sex, ethnicity or even current mental state. The information about another person's age and sex allows us to, for example, determine whether we should use words such as "Sir" or "Madam" while addressing them or to estimate which conversational topics they are most likely to be interested in. An 8 year old girl is significantly less likely to be interested in politics than a 35 year old man.

The process of recognizing a human's age or sex consists of 4 steps which human brains and computers alike need to complete.

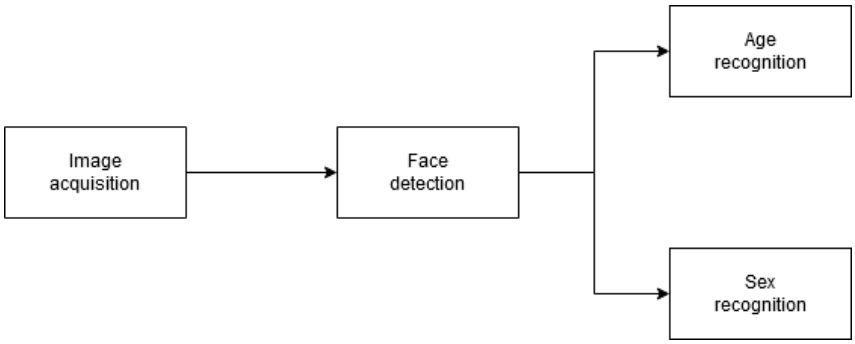


Figure 1.1. Steps of human age and sex recognition

The first step is image acquisition. It is completed with the use of a camera, which records a number of image frames, which are then saved in the computer's memory in the form of digital data. Next, the image is used as an input by a face detection algorithm. There exist many face detection algorithms with varying speeds and rates of accuracy. These methods and differences between them will be described in section 3.2. The final two steps are age recognition and sex recognition and, unlike the previous two, they can be completed in any order. Just like in the case of face detection, these can be achieved using different methods, which will be described in sections 3.3 and 3.4.

1.1. Statement of the problem

The main problem of the thesis is to design and implement a system for detecting faces of people on photographs and detecting their sex and age. Convolutional Neural Networks (CNNs) will be the main tool used in the proposed solution.

1.2. Scope of thesis

The paper is comprised of two parts: Theoretical introduction and solution to the problem. The first theoretical chapter includes an overview of mechanisms behind neural networks and their architectures. The second theoretical chapter includes a basic explanation of the face detection, age recognition and sex recognition problems and different methods for solving them. In Chapter 4, a proposed solution for the age and sex recognition problem is presented. The project has been developed using the Python language, using Keras and TensorFlow deep learning libraries. The following chapter contains an accuracy analysis of the developed program and it's comparison to similar solutions presented in recent articles.

2. Neural networks

This engineering thesis aims to utilise Convolutional Neural Networks for age and sex recognition. Several basic concepts and their respective modelling tools are examined throughout this chapter. Its initial segment describes and formally defines various mechanisms governing neural networks, such as their construction and mechanisms. In the second section, a showcase of basic neural network architectures is presented, as well as a comparison of their defining features.

2.1. Basic information

2.1.1. Definition of a neural network

Of all the numerous proposed definitions for neural networks, from my point of view S. Haykin's one represents it most accurately:

> "A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use." [1]

2.1.2. Architecture

A neural network is build of small building blocks called "neurons". They are represented by circles on the graph. Basic types of neural networks are comprised of three types of layers, namely: Input layer, hidden layer(s) and an output layer. Neurons are connected, one layer to another. Connections between them are represented by lines on the graph [1].

2.1.2.1. Neuron

Neurons are the fundamental building blocks of a neural network and it's processing units [1]. The model of Figure 2.1 demonstrates the elements of a neuron. These elements are:

1. Input - Input values are either outputs, passed from a previous layer of neurons or outside of the network, in case of the input layer [1].

2. Weights - A weight w_n is used to define how important an input x_n is supposed to be in relation to other inputs at a given moment. These weights change as the neural network learns and the end

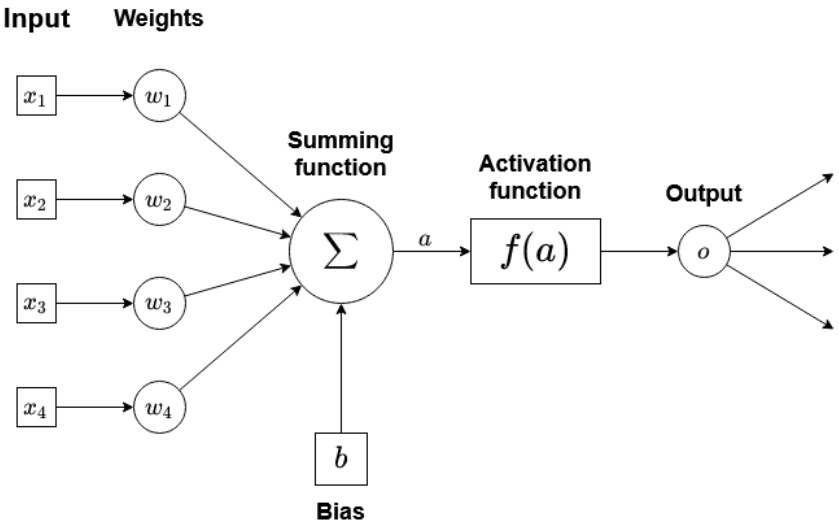


Figure 2.1. A diagram demonstrating the elements of a neuron

result outputted by the network depends on their values. Weight values can be either positive or negative [1].

3. Bias - Bias b is another component of a neuron besides weights that changes during the learning process. It is a single value which is inputted into the summing function and it may either increase or lower it's output depending on whether it's positive or negative. Biases tend to have a big impact on the results returned by the neural network [1].

4. Summing function - It is defined by the following equation:

$$a = \sum_{j=1}^{n} w_j x_j + b$$

where n is the number of inputs, $x_1, x_2, ..., x_n$ are the inputs, $w_1, w_2, ..., w_n$ are the weights of the inputs and b is the bias [1].

Summing function simply sums all the outputs (and the bias value) of the previous layer.

5. Activation function - It is a function, which attempts to introduce non-linearity to the network. The reason why this is important will be presented further in this chapter. [1]. Activation functions transform a value passed to them, so that the result is a part of a specific interval, such as [0,1], [-1,1] or [0,+∞) [5]. Types of activation functions and their ways of application will be presented in section 2.1.2.

2.1.2.2. Layers

The model of Figure 2.2 demonstrates the three types of layers in a neural network. These layers are:

1. Input layer - It is responsible for accepting single inputs and passing them forward. It is a passive layer, meaning that it will not modify the input that it receives, instead, it will multiply it and then

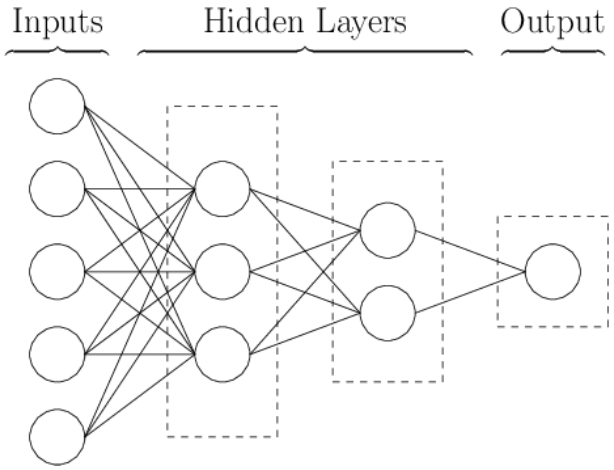


Figure 2.2. An example of a neural network's architecture [2].

forward it to the next layer of neurons, a hidden layer. Unlike neurons from the hidden and output layers, those in the input layer do not change their weights over the course of learning, neither do they fit the result of the summing function to a specific interval. They always simply pass their inputs forward, with no modification [1].

2. Hidden layer - A neural network may contain one or many of these layers. Each hidden layer is responsible for discerning a different feature of analysed data, with subsequent layers being responsible for features of increasing complexity [1]. For example, in a neural network designed for detecting human faces on a picture, one of the initial layers could, over the course of learning, begin to specialise in detecting circles on an image, one of the further layers could detect eyes and another layer located further in the net could detect faces on an image. Hidden layers of the neural network achieve this ability to categorise and detect features through the process of backpropagation, which will be discussed further in this chapter.

3. Output layer - It is the final layer of a network, responsible for transferring the final result outside of the network [1].

2.1.2.3. Activation functions

Selection of activation functions is one of the most important parts of designing a neural network. Different activation functions are suitable for different problems. Altering the types of activation functions present in network's layers has a big impact on the manner, in which the network processes data. [1].

In order for an activation function to be effective, it should introduce a degree of non-linearity to the network. This is important because real life patterns and objects are usually not linear, in the sense that they do not always appear exactly the same, just similar enough for humans to categorise them as the same type of object. For example, an apple can be either green, red or a combination of these colours;

it can be smaller or bigger than a fist. By introducing non-linear transformations to our network, we can train it to put objects which are only a little different into the same categories, just as humans would [1].

While an activation function can technically be any function that accepts a single value and returns a single value, in practice, there's a specific selection of functions which provide best results and are most commonly used. These include:

1. Sigmoid - It is a function defined by the following equation:

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

Sigmoid takes the input and transforms it in such a way that the resulting value is within a [0,1] interval. Very high and very low values of x result in similar output values [1]. This property has both advantageous and disadvantageous aspects. While the function enables very clear predictions of the end value, due to the fact that all the values above 2 or below -2 will be very close to 1 or 0 respectively, it also has the effect of slowing down the learning rate of the network if the input value of the function is either very large or very low. This is caused by the fact that small changes of outputs cause the gradient to be small, directly negatively impacting the network's ability to train. This problem is also known as the "Vanishing gradient problem" [3]. Sigmoids can be used in both single- and multi-label classification problems, because the outputs of a sigmoid layer can be interpreted as probabilities of each label to be correct, though, sigmoid functions are most commonly used in multi-label classification. In the case of a single-label classification problem, we would choose only one label with the highest output value, while in the case of a multi-label classification problem, we would choose all the labels with output values above a certain threshold.

2. Tanh - It is a function defined by the following equation:

$$g(x) = \frac{2}{1 + e^{-2x}} - 1$$

Tanh is a scaled sigmoid function, meaning that it is characterised by the same advantages and disadvantages. It can also be used in the same ways as a sigmoid. One of the key differences of tanh, is that it returns a value in the range of [-1,1] instead of [0,1]. Tanh is also suspectible to the vanishing gradient problem [5].

3. ReLU - It is a function defined by the following equation:

$$h(x) = max(0, x)$$

ReLU, the rectified linear unit function returns values in the range of [0,$+\infty$). It returns the original value, if it's positive or 0, if it is equal or lower than 0. It is a non-linear function, despite the fact that it behaves linearly for values above 0. This property of ReLU makes it a very popular choice

in many different neural network models because it is able to return a full range of positive values, without suffering from the inability to train, which regular linear functions posses. It does not suffer from the vanishing gradient problem, which is a big advantage, especially in large Deep Neural Networks, though it suffers from another problem, namely, "The dying ReLU problem". This issue is caused by ReLU always returning 0 for negative inputs, which causes the gradient to become 0. A situation like this may cause neurons, which use ReLU, to stop responding to changes in the input and become passive [5]. This problem can be mitigated by using a modified version of ReLU, which does not return values of 0 for negative numbers, but instead returns values very close to 0, for example $0.01x$. This modified version of ReLU is called "Leaky ReLU" [4].

4. Softmax - It is a function defined by the following equation:

$$s(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

where k is a total number of softmax layer inputs, x_i is the currently evaluated input value and $x_1, x_2, ..., x_j$ are all the input values of the layer.

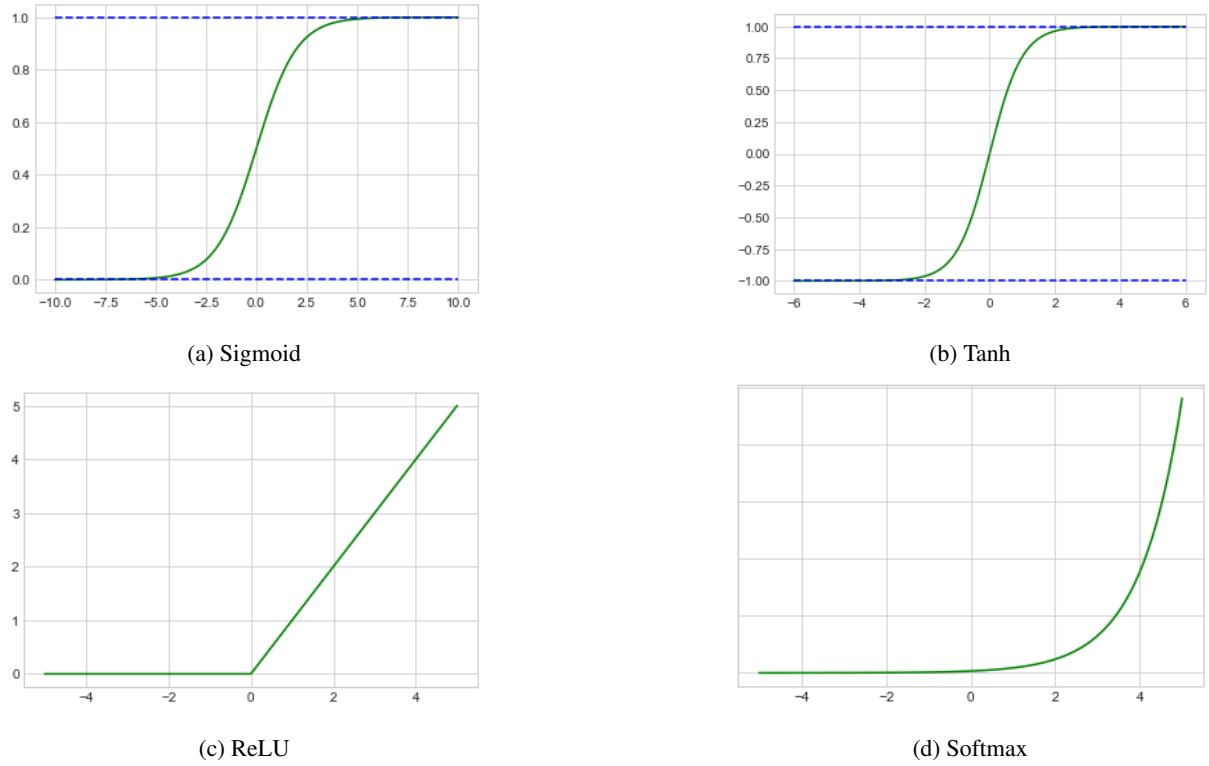
Softmax determines the probabilities to be true for each label in a layer for given data. This is accomplished by calculating an exponent of each value and then normalising it, so that the sum of all probabilities is equal to zero. It normally implemented as a last hidden layer of a Deep Neural Network, right before the output layer. Softmax is commonly used in single-label classification problems, as it returns the probabilities of each label to be the only correct one for the given data. Unlike sigmoid, it should not be used for multi-label classification problems. "Any time we wish to represent a probability distribution over a discrete variable with n possible values, we may use the softmax function." [5]

2.1.3. Backpropagation

Backpropagation, short for "backwards propagation" is a mechanism, which allows neural networks to train. As the data flows through the network, a process of "gradient descent" attempts to discern the local minimum of the network's error function. In order for backpropagation to succeed and successfully train the network, a large amount of data is required. [5]. Backpropagation is repeated multiple times, depending on the parameters of a network, using random batches of input-output pairs. The algorithm is carried out in steps, which are described below.

2.1.3.1. Forward pass

In the beginning, random weights and biases are assigned to the network. This causes it to return erroneous results. In order to correct these weights and biases, we need to obtain the values of activations a_j^k and outputs o_j^k for each neuron, which will in later allow us to determine values of error. This



(a) Sigmoid

(b) Tanh

(c) ReLU

(d) Softmax

Figure 2.3. Graphs of popular activation functions

phase is completed by simply inputting a single instance of data into the network and saving necessary information as it is evaluated by each layer. [5].

2.1.3.2. Backward pass

After the forward step is complete, we may use the information gathered during it to calculate the errors of each neuron. This process begins at the output layer and continues backwards, until the input layer is reached. [5]. As this process is fairly complex, it will be presented in substeps:

1. First, we calculate the final output error using a selected error function. A number of different error functions can be used depending on the problem. In the case of a mean squared error:

$$E(o, \hat{o}) = \frac{\sum_{i=1}^{N} (\hat{o}_i - o_i)^2}{2N}$$

where o is the expected result, $\hat{o}$ is the calculated result and N is the number of elements in the output vector [6].

2. Next, we need to calculate partial derivatives of the error function in respect to weight, for every weight in each layer, starting from the last layer. Utilising the chain rule, we can determine the following equation:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ij}^k}$$

where a_j^k is the activation value of a neuron j in layer k and w_{ij}^k is the weight of a connection between a neuron j in layer k and a neuron i in layer $k-1$ [6].

Through this equation we can determine that a partial derivative of the error function in respect to weight is a product of two components:

(a) Partial derivative of the error function in respect to activation, which is often called the error δ_j^k [6]:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k}$$

(b) Partial derivative of the activation in respect to weight, which can be further calculated :

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\left(\sum_{h=1}^{N^{k-1}} w_{hj}^k o_h^{k-1}\right) + b^k\right) = o_i^{k-1}$$

where N^{k-1} is the number of neurons in layer $k-1$ and b_k is the bias of layer k [6].

3. In the next step, we need to determine the value of error δ_j^k for neurons in a given layer.

In the case of the output layer, this value is:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \frac{\partial}{\partial a_j^k}\left(\frac{1}{2N}\sum_{i=1}^{N^k}(f(a_i^k) - o_i)^2\right) = \frac{1}{N}(f(a_j^k) - o_j)f'(a_j^k)$$

where $f(\circ)$ is the activation function of the output layer [6].

In the case of hidden layers, through the use of chain rule for multivariate functions, we can determine this value to be:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{i=1}^{N}\frac{\partial E}{\partial a_i^{k+1}}\frac{\partial a_i^{k+1}}{\partial a_j^k} = \sum_{i=1}^{N}\delta_i^{k+1}\frac{\partial}{\partial a_j^k}\left(\left(\sum_{h=1}^{N^{k+1}} w_{hi}^{k+1} f^k(a_h^k)\right) + b^{k+1}\right) =$$

$$= \sum_{i=1}^{N}\delta_i^{k+1} w_{ji}^{k+1} f^{k'}(a_j^k) = f^{k'}(a_j^k)\sum_{i=1}^{N}\delta_i^{k+1} w_{ji}^{k+1}$$

where $f^k(\circ)$ is the activation function of the hidden layer k [6].

After these calculations, we can determine that the derivative of the error function in respect to weight can be described as:

(a) In the case of the output layer [6]:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{1}{N}(f(a_j^k) - o_j)f'(a_j^k)o_i^{k-1}$$

(b) In the case of the hidden layers [6]:

$$\frac{\partial E}{\partial w_{ij}^k} = f^{k'}(a_j^k)o_i^{k-1}\sum_{i=1}^{N}\delta_i^{k+1}w_{ji}^{k+1}$$

4. We repeat the process of calculating the derivative of the error function in respect to weight for all weights in the network, for each element in the batch, and then calculate the average for each weight. The old weight is then updated by subtracting the calculated gradient, multiplied by the network's learning rate, from it .

$$\Delta w_{ij}^k = -\alpha\frac{1}{B}\sum_{p=1}^{B}\frac{\partial E_p}{\partial w_{ij}^k}$$

where α is the learning rate of a network, p is the input, output pair index and B is the batch size [6].

The above steps are repeated for each batch of input-output pairs, until a satisfactory low error rate is achieved, or until a given number of iterations has completed [6].

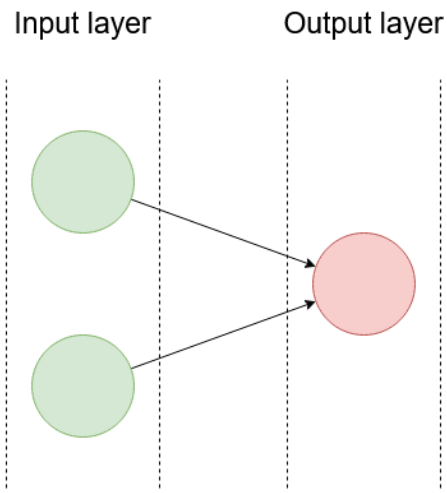
2.2. Select neural network architectures

2.2.1. Feed forward

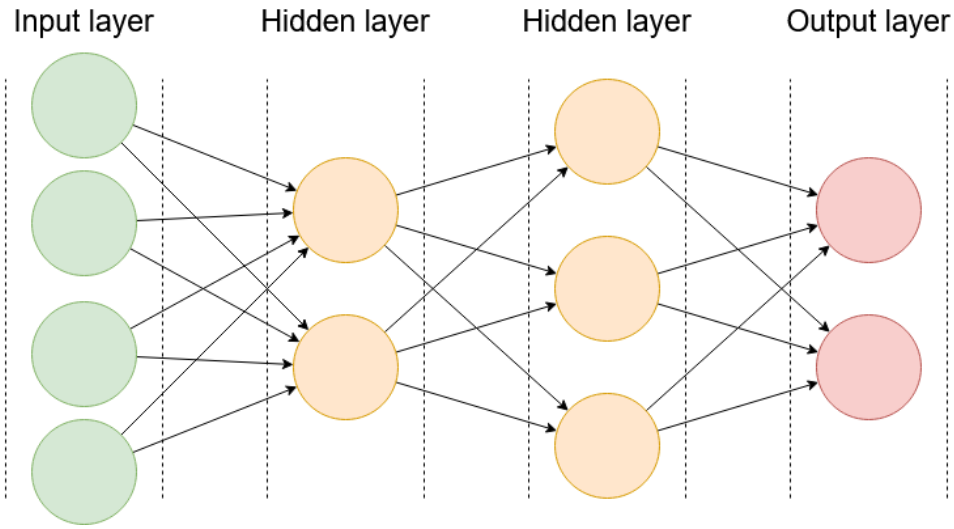
In chapter 2.1.2, a basic model of a Feed Forward Neural Network's architecture has been presented. It is the oldest and most basic type of a neural network. Its characteristics include:

1. Neuronal connections do not form cycles.
2. Data moves only with a forward direction, from input layer to output layer.
3. Neighbouring layers are fully connected, meaning that every neuron in layer k has a connection to every neuron in layer $k + 1$.
4. There may exist any number of hidden layers in the network.

A Feed Forward Neural Network without hidden layers is known as a "Single-layer perceptron", while a Feed Forward Neural Network with one or more hidden layers is known as a "Multi-layer perceptron". Feed Forward Neural Networks may be used for tasks such as pattern recognition, speech recognition or hand-writing recognition [1].



(a) Single layer perceptron



(b) Multi layer perceptron

Figure 2.4. Feed Forward Neural Networks

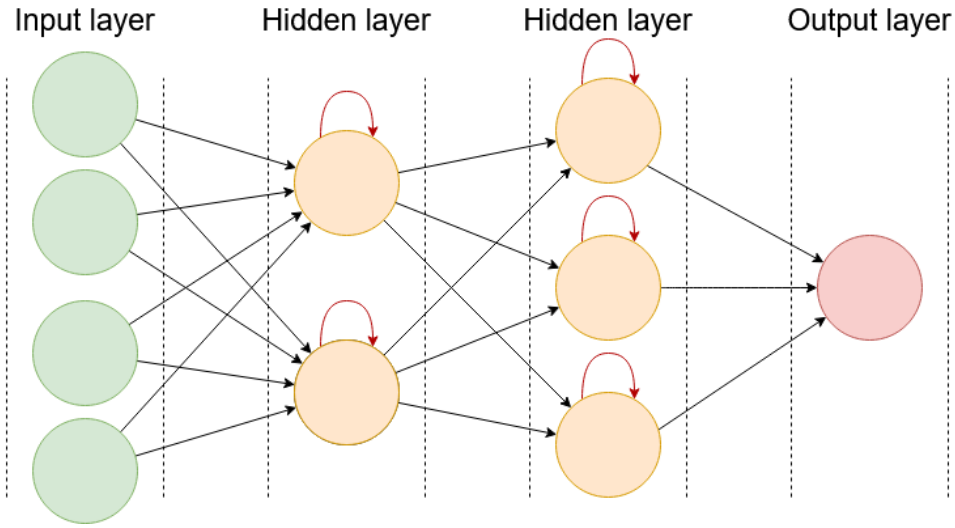


Figure 2.5. Recurrent Neural Network

2.2.2. Recurrent

Recurrent Neural Networks (RNN) are characterised by their ability to access previous predictions. This ability is useful when we're dealing with sequential data, such as written text or speech recordings. In such cases, in order for the network to make an accurate prediction, it needs to know the context in which new data appeared. RNNs are rather similar to Feed forward networks, although they posses a few important differences. Characteristics of a Recurrent Neural Network include:

1. There exist loops in the network, which allow it to store infromation.

2. All layers posses the same weights and biases, but they still change in the process of training

Recurrent Neural Networks are commonly used for tasks such as text translation, speech recognition or robot control [1]. There is a special architecture of a Recurrent Neural Network, called Long Short Term Memory Network (LSTM), which uses a more advanced memory system. LSTMs are capable of processing sequential data with memory gaps [31].

2.2.3. Convolutional

A Convolutional Neural Network (CNN) is a variant of a Feed forward network with added convolutional layers. It is a network architecture specialised for classification of images and object recognition. Convolutional Neural Networks are able to accept images as input and then, through the use of filters which are implemented inside so-called "convolutional layers", successfully perceive dependencies between pixels within an image and use these dependencies during classification. A CNN is mostly an extended version of a Feed forward network. Its special characteristics include:

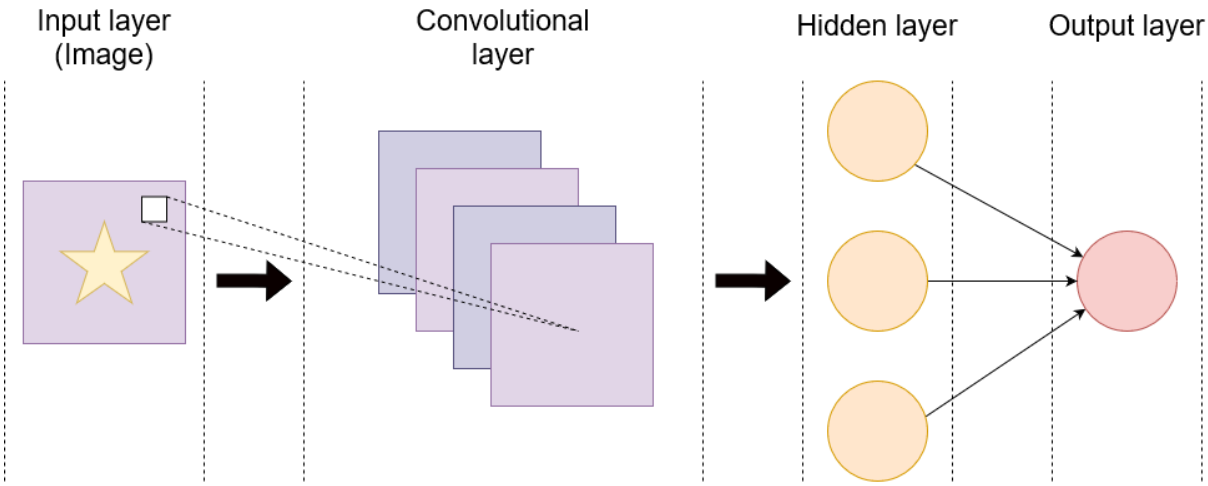


Figure 2.6. Convolutional Neural Network

1. CNNs use convolutional layers, which contain filters that can reduce the input image into a form which is easier for the network to process, while also preserving the dependencies between specific areas on an image.

2. Unlike in Feed forward networks, not all layers are necessarily fully connected. Neurons in convolutional layers usually connect to only a small part of neighbouring convolutional or input layers, which they use as input for their filters.

Convolutional Neural Networks are commonly used for computer vision related problems, such as face detection/recognition, object classification, video analysis or image up-scaling [1]. A Convolutional Neural Network is also used in this paper as a base for a sex and age recognition system in chapter 4. A deeper analysis of used CNN design is also included in this chapter.

3. Age and sex recognition

As was mentioned in the introduction chapter, age and sex recognition is a process, which consists of multiple steps. This chapter aims to present an overview of this process, explain possible applications of face detection, as well as age and sex recognition, identify problems and limitations which may occur during each step and showcase example methods and tools which may be used for their completion.

3.1. Face detection

Face detection is a process of identifying areas of an image, which contain human faces. Depending on the used algorithms and their implementation, a face detection system may be able to detect one or multiple faces on an image.

It is a fairly complicated process, compared to other object detection problems, due to the fact that human faces vary widely in appearance. People's faces vary in skin colour, hair colour, size and shape of facial features, such as nose, eyes, ears, etc. Some people wear glasses or piercings, some others wear intense make-up or even masks. A photographed person may be expressing a wide variety of emotions, which changes the alignment of their facial features; they may also be facing the camera in any angle. The presence of all these potential features and variables must be accounted for by the face detection system, in order for it to work properly. The quality of the input image is also an important factor. Images that have too low of a resolution or that are heavily distorted may be difficult or impossible for a face detection system to successfully evaluate.

In recent years, there has been an increased interest in the issue of face detection, primarily due to the fact that face detection algorithms are often used as a base for other systems, which use faces as their input data. These systems will be further discussed in subsections 3.2 and 3.3.

3.1.1. Operation process

Even though there exist many different face detection algorithms, they share a basic procedure:

1. Extract an image from specified location - User provides the algorithm with location of a picture, which will be further evaluated by the algorithm.

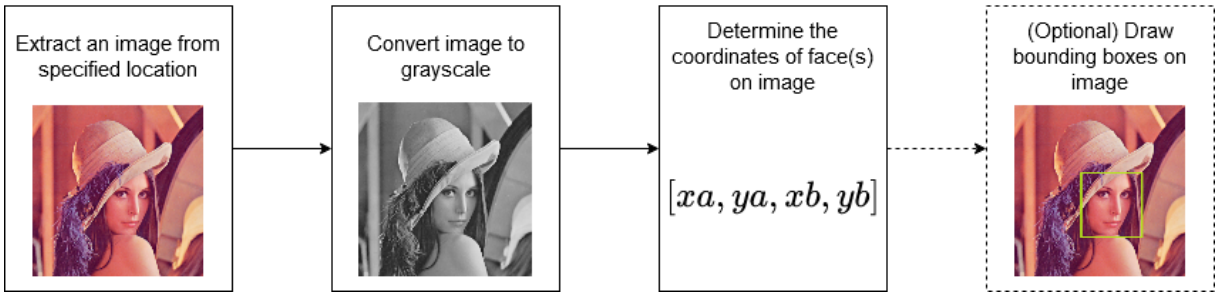


Figure 3.1. Steps of a face detection algorithm

2. Convert image to grayscale - Due to the fact that face detection algorithms do not require colour channels to work, by converting an image from RGB to grayscale, we can reduce the size of data that we will operate on, which may increase efficiency, without negatively impacting the algorithm's accuracy.

3. Determine the coordinates of face(s) on the image - This step is carried out in a different manner, depending on the chosen algorithm. In every case, at this point the algorithm should provide us with coordinates of a face or multiple faces, if it detected any.

4. (Optional) Draw bounding boxes on image - For the sake of visualisation, we may choose to display an image with highlighted locations of detected faces.

3.1.2. Classification of face detection algorithms

A common classification of face detection algorithms divides them into four distinct groups [9]:

1. Knowledge-based - Knowledge-based methods possess data of typical locations of facial features in relations to others. As these typical locations are usually directly coded into the program, by it's author and not collected through a process of learning, they tend to only accurately describe the most standard archetypes of human faces. Because of that, these algorithms don't tend to achieve good results and aren't widely used [9].

2. Template-matching - Template-matching methods extract smaller parts of an image and then compare them to a set of templates defined in the system's code. If a sub-image is similar enough to one of the face templates, the algorithm identifies it to be a face. Unfortunately, these methods are not accurate enough compared to different types of solutions [9].

3. Feature-based - These methods attempt to detect faces on images through the use of filters. These filters are used to collect information about the location of facial features, from which a statistical model is built. After processing a large enough amount of faces, a statistical model constructed in this manner may start achieving good results on previously unseen data. [9].

4. Appearance-based - By combining the usage of face templates utilised by the template-matching category of face detection methods with various machine learning techniques, appearance-based methods are able to achieve the best results out of all the algorithm categories. Generally, methods in this group attempt to create accurate templates of human faces by processing large amounts of training data, instead of using pre-defined templates. [9]. Examples of such methods include:

   (a) Eigenfaces - The Eigenfaces are a special type of face templates, created through the process known as Principal Component Analysis (PCA). These templates are created by processing many different photos of the same person, and they are supposed to only match that specific person's face. After the templates are created, the algorithm is able to detect a given face on new photos. A clear disadvantage of this method, though, is its inability to recognise faces which do not match the data supplied to the algorithm [10].

Figure 3.2. Example eigenfaces [10].

   (b) Neural networks - The neural network method, which is one of the main topics of this paper, utilises different artificial neural network architectures to detect faces. In the past, researchers successfully used hierarchical, associative and probabilistic decision-based neural networks, but currently the most popular and accurate approach is to use deep convolutional neural networks. The details of implementation of this specific type of neural network architecture, with the goal of face detection, will be discussed in chapter 4.3. Neural networks, despite being disadvantaged by the requirement of a large amount of training images, after being successfully trained, may detect any person's face and not just the ones which were used during training, unlike systems which utilise the eigenfaces approach [5].

   (c) Support Vector Machines - SVMs are a class of linear classifiers. Their training process, known as "structural risk minimization", utilises support vectors. Support vectors are the edge values in the training set, that is, values which are classified to be a certain object, despite being very similar to another object. For example, if our goal were to create a classifier, which is able to accurately detect apples and oranges on a picture, the support vectors would be apples that look the most like oranges and oranges that look the most like apples. The training algorithm attempts to find a separating hyperplane, which has the maximum distance

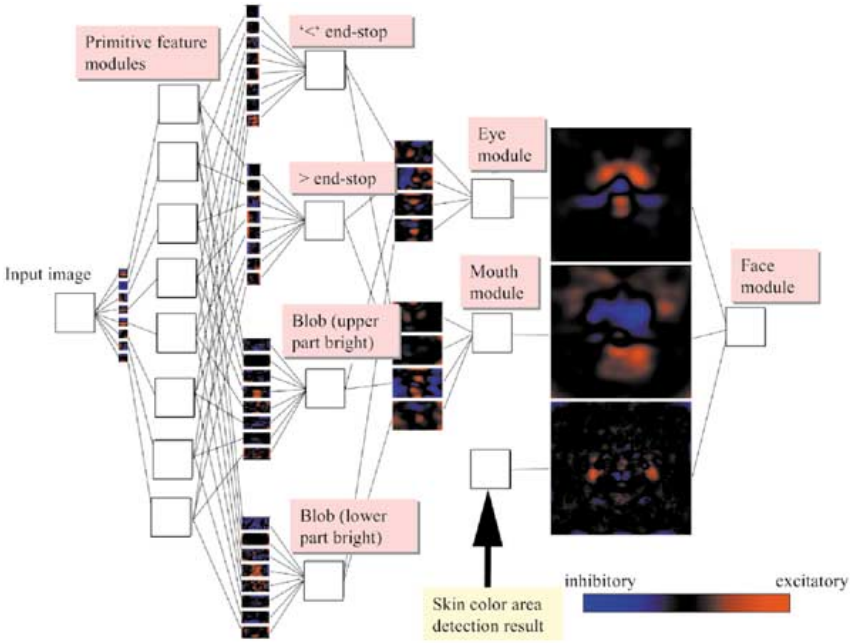


Figure 3.3. An example of a face detection CNN architecture [11].

to these support vectors. After the training is complete, the resulting hyperplane is the one, for which the classification error is expected to be minimal. In the case of face detection, we would create a SVM using a "one-vs-all" approach, meaning that we would only consider two classes of objects: faces and non-faces [32].

(d) Naive Bayes Classifiers - Naive Bayes Classifiers can be used to detect faces on an image, by determining the probabilities of a face being present in specific areas of an image. First, the system is trained to recognise which patterns of pixels on an image have a high probability of being a fragment of a face. After the training is complete, we may input an image into the system. NBC will then divide the image into many small subregions, and determine the probability for each subregion to be a fragment of a face. The neighbouring subregions are then merged together. If the probability ratio of the combined area is higher than the value of prior probabilities, then a face is detected in that area [33].

(e) Viola-Jones Algorithm - Being one of the most popular face detection solutions, Viola-Jones possesses extremely fast face detection speed. Unfortunately, this comes at a significant cost of being one of the slowest face-detection algorithms to train. Viola-Jones utilises groups of weak classifiers. These classifiers form a cascade, which the input image is passed to, and evaluated by. The input image is passed through each classifier in the cluster, one after another. The exceptionally fast speed of this solution makes it an excellent choice for systems, in which face detection speed is the most important factor, such as cameras. [12].

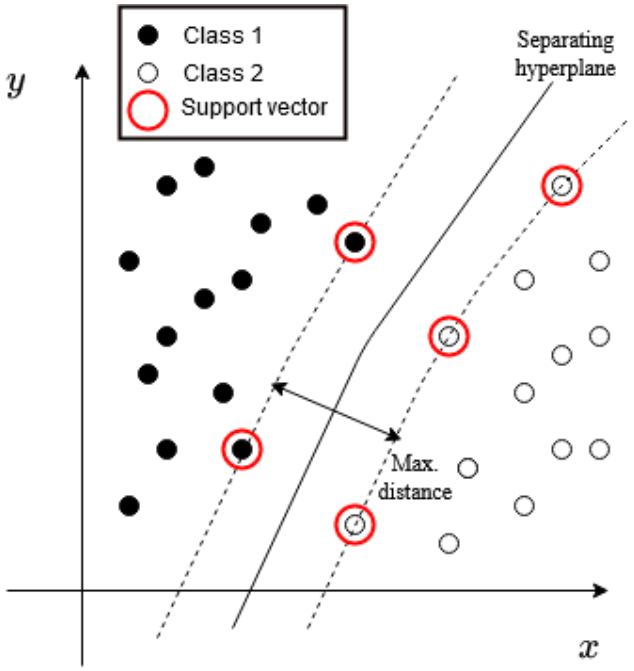


Figure 3.4. SVM diagram

3.1.3. Applications

Face detection is most often used in combination with other algorithms, which require the coordinates of a face on an image to function properly. Such applications of face detection include:

1. Photography - Real-time face detection systems are commonly used in photography. The ability to detect faces on a viewed scene in order to automatically focus the camera on a person is an ordinary feature of most cameras nowadays. Some modern cameras also possess the ability to detect smiles on a photo, in order to take a picture at the best possible moment [14].

2. Emotion recognition - A different application of face detection, is emotion recognition. It is a process of recognising the emotional state of a person, using their facial expressions [15]. People have varying predispositions for emotional recognition, for example people who are on the autism spectrum may have a hard time recognising how other people feel based on their facial expressions. Real-time emotion recognition systems may help them in that task and make them able to interact with others more comfortably.

3. Computer animation - Facial feature detection is used in the field of computer animation. Animation studios use facial tracking systems to record data about actors' facial features when they speak or express emotions. This data can later be used to create realistic-looking characters in animated videos or video games. There are a few approaches to this problem. One of them includes sticking or painting special markers on an actors face, in key positions, which the system is programmed to detect. As the actor speaks or moves, the markers move as well. Later on, the recorded video

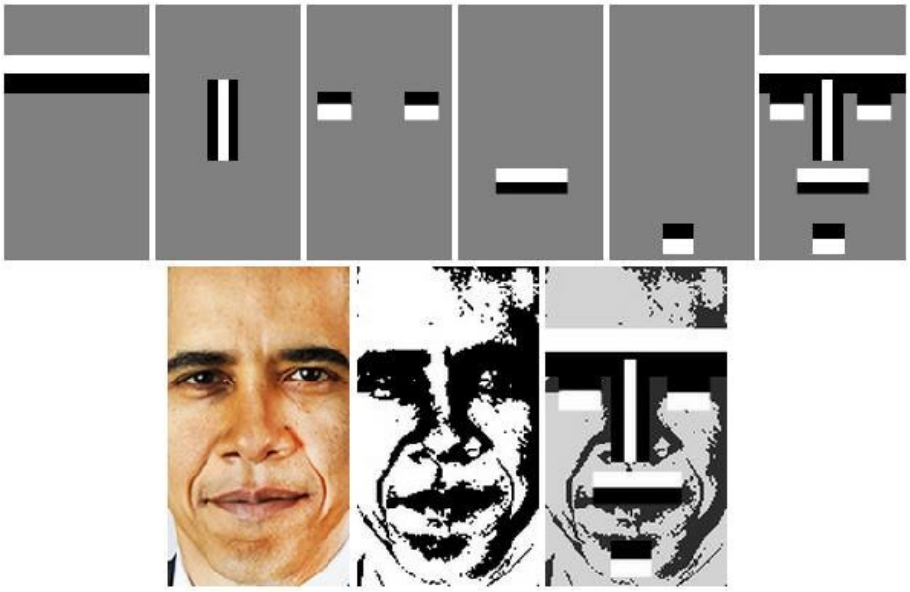


Figure 3.5. An illustration Haar-like features on an example photo. [13].

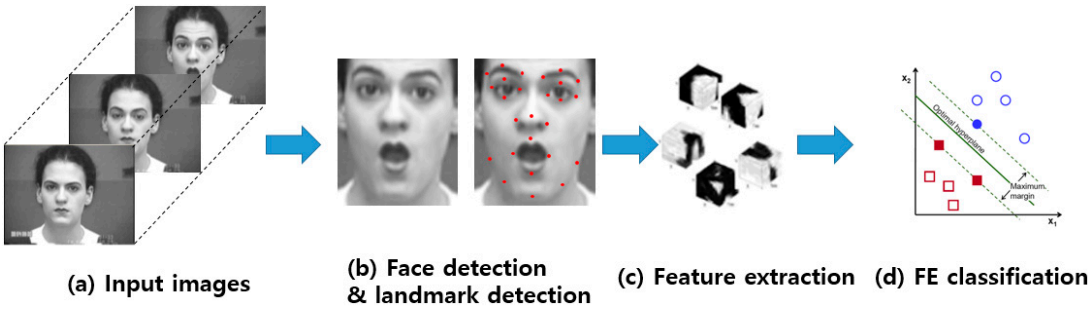


Figure 3.6. Recognising emotion through the usage of classifiers [16].

can be analysed by the feature detection system, which translates the changing position of markers to a data format that can be later used in animation software. Recently, more advanced face tracking systems have been created, which allow for face tracking and animation in real time, without the use of any markers. An example of software utilising this technology is FaceRig, which is commonly used by video streamers and creators to assume a fictional persona.

4. Face recognition - Facial recognition is a broad topic and it has gained a lot of attention in the recent years [18]. There are many algorithms that allow for quick and efficient facial recognition. Facial recognition is most often used as a part of systems which solve more complex problems. It is commonly used by smartphone manufacturers to implement user verification systems, one of which is FaceID for iPhone devices. Another use of facial recognition, which is gaining popularity, is classroom attendance tracking. Cameras placed in classrooms are used to detect faces of present students and automatically determine which are absent [19]. Facial recognition is also used for access control in high security areas. In order for a system like that to work properly, a database of faces of authorised people is created. Cameras are mounted next to doors, which allow the system

to detect faces of any people trying to enter and, based on whether their faces exist in the database and have proper permissions, it can allow them to pass or not.

5. Biological feature recognition - Face detection can be used to determine various biological features in people. Such features include ethnicity, sex, age or even illnesses that a person is afflicted with. The topics are further discussed in the following sections of this chapter.

3.2. Sex recognition

Sex recognition is a process of estimating a person's biological sex using available data. Different types of data may be used for this purpose, such as: voice recordings, samples of text written by the tested person or a face photo. In this section, examples of sex recognition methods are presented.

3.2.1. Methods of sex recognition

3.2.1.1. Support vector machines

One of the first attempts at sex recognition using photos was made by Moghaddam and Yuang [21]. During their research, they have used SVMs on images from the FERET database. Using this method, an error rate of 3.4% has been achieved.

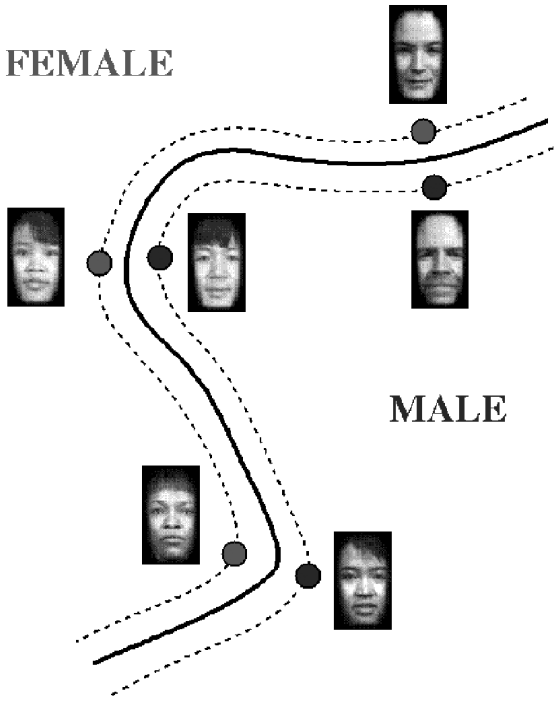


Figure 3.7. Diagram of SVM used by Moghaddam and Yuang [21].

(a) Male (b) Female

Figure 3.8. Illustration of male and female visual traits detected by the Bayesian Classifier in [22].

3.2.1.2. Naive Bayes Classifiers

In 2007, a new method for sex recognition has been created by Toews and Arbel [22] utilising Naive Bayes Classifiers. Their paper presented a model of a Bayesian Classifier trained to identify the model features indicative of visual traits. This model has achieved an error rate of 16.3% on images from the FERET database.

3.2.1.3. Convolutional Neural Network

More recently, Levi and Hassner [20] have created a system for age and sex recognition using CNN. They have created a single neural network model, which is capable of recognising both sex and age. It is a relatively simple model, which is intentional, as the biggest obstacle in the age and sex recognition problem is the risk of overfitting. The used model is shown on Figure 3.9. In this paper, an attempt is made to create a different CNN model for age and sex recognition with comparable or better accuracy. The details of this model's implementation are contained in Chapter 4, while the comparison to other models is in the Chapter 5.

3.3. Age recognition

Age recognition is a problem of recognising which age group a person belongs to, based on their characteristics, in this case, facial features. Automatic discerning of a person's age based on a photo of their face is a difficult task because of a high degree of variability. People age at different speeds and the rate at which they age is often determined by factors that are very hard to account for, such as genetic makeup, environment or diet. Because of that, scientific works which focus on the problem of

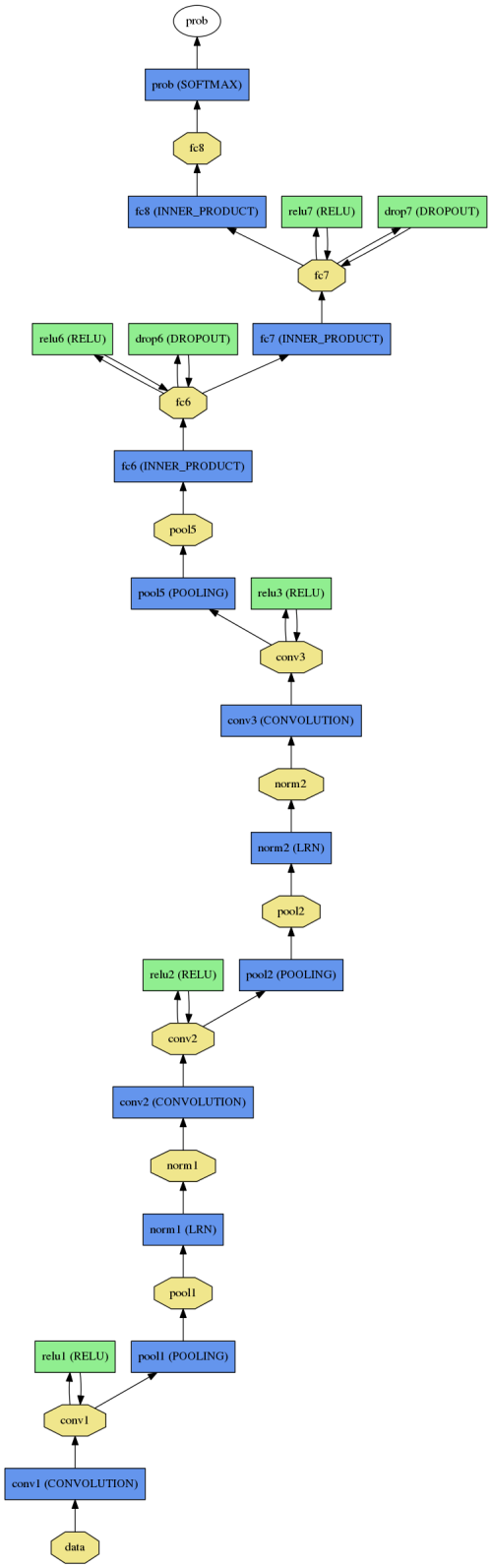


Figure 3.9. Model of a Convolutional Neural Network for age and sex recognition in [20].

age recognition usually aim to construct a method for accurately assigning facial photographs to arbitrary age groups. Attempting to recognise the exact age of a person based on a photo of their face would yield too big of an error value to be useful, due to the aforementioned obstacles. Examples of scientific works discussing methods of age recognition include:

3.3.1. Methods of age recognition

3.3.1.1. Support Vector Machines

A method for age recognition utilising SVMs has been created Guo et al [24]. Authors of the paper have used a Locally Adjusted Robust Regressor (LARR), which is a modified version of the SVM approach, to achieve a mean error of 7.16 years on images from the FG-Net database.

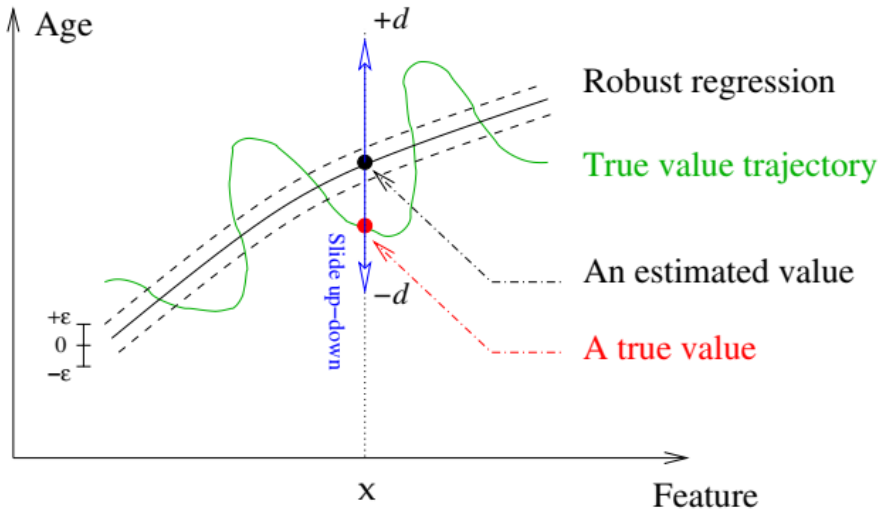


Figure 3.10. An illustration of the idea behind LARR in [24].

3.3.1.2. Gabor Features + Fuzzy LDA

Gabor features are a result of using Gabor filters on an image. A technique utilising Gabor features in conjunction with a modified version of Linear Discriminant Analysis has been showcased in [25]. LDA has the ability to assign more than one age group to a single image, in cases where the image does not provide enough certainty about which age group it belongs to.

3.3.1.3. Convolutional Neural Network

For more information about the usage of CNNs for age recognition, please refer to section 3.2.1.3

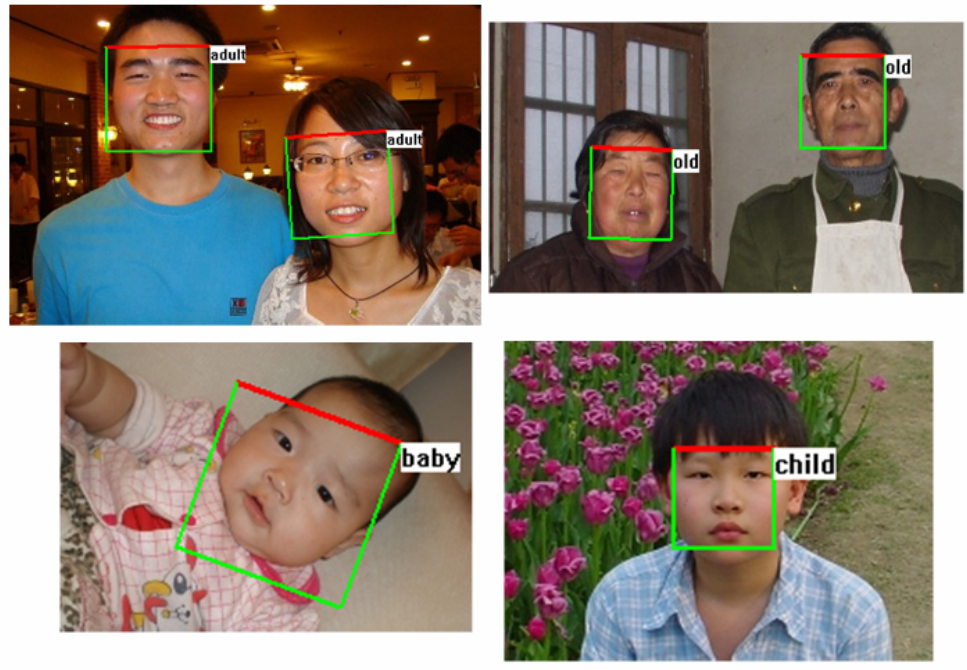


Figure 3.11. Result of an age detection system utilising Gabor Features and Fuzzy LDA in [25].

4. Proposed solution for the age and sex recognition problem

4.1. Data and tools used in the project

The project is implemented in the Python language, using Jupyter Notebook. Python has been chosen due to abundance of libraries written for it, which are specialised for data analysis and neural network modelling.

Jupyter Notebook has been chosen as the development application, due to it's readability and ability to run specific blocks of code independently to others, which is useful in the case of neural network modelling. Jupyter Notebook is a good choice for applications which do not contain a large amount of code, which is the case with this project.

4.1.1. Dataset

The dataset which is used for network training is the "Adience dataset of unfiltered faces for gender of age classification" [26]. It is a dataset which contains photos of faces in varying environments, backgrounds, angles and light settings. This variety makes it ideal for training a network, which is meant to be able to process photos in any settings.

The photographies in this dataset are paired with a file containing information about the sex and age group of the photographed people. The file also contains some irrelevant information, which is cut during the dataset processing step in the program.

Figure 3.12 presents example photographs contained in this dataset.

4.1.2. Keras

Keras is an interface to TensorFlow, which is a python deep learning library. It allows the user to build a neural network using modules, such as customisable neural layers, activation functions, objectives and optimizers. It provides an intuitive way for using TensorFlow.

It is a very commonly used deep learning software, with a large community dedicated to it. A large number of articles and tutorials about neural network modelling can be found on the internet. Intuitiveness of use and multitude of resources are the reason for which Keras has been chosen as the tool for modelling neural networks in this paper.

Figure 4.1. Examples of photographs contained in the Adience dataset

Apart from implementing a multitude of tools for creation and training of neural networks, it also provides a range of functions for processing data, so that it may be used to train a network. Such functions are used multiple times throughout the project and include `to_categorical()` and `img_to_array()` [34].

4.1.3. Other notable libraries used in the application

4.1.3.1. Pandas

Pandas is a very popular data-manipulation library, due to its abundance of useful and fast functions, which are suitable for processing large datasets. In this project, a couple of functions implemented by Pandas are used, such as `read_csv()` or `concat()` [36].

4.1.3.2. Numpy

Numpy contains a number of efficient and fast data structures, which are preferable to Python's built-in arrays, such as the Dataframe. It also adds a large amount of functions for creating and operating on these structures. In this project Dataframe is used for storing large amounts of data used for training the neural networks [37].

4.1.3.3. OpenCV

OpenCV is an open-source library which contains tools for computer vision, machine learning and image processing. It is available for languages such as Python, Java or C++. It contains useful functions for processing images, so that they may be accepted by neural networks created in this project [38].

4.1.3.4. MTCNN

MTCNN is an implementation of the Multitask Cascaded Convolutional Networks for Keras in Python. It will be used to detect faces in the project. A very useful function of this library, is it's ability to detect facial keypoints, such as location of eyes, nose and corners of the mouth [39].

4.2. Training implementation

4.2.1. Dataset processing

The first step is extracting data from the Adience dataset text files and saving them in a Pandas [36] dataframe.

```
adience_path="C:/Users/QCXM38/Desktop/adience/"
adience_df=pd.read_csv(adience_path+"fold_0_data.txt", sep="\t", header=0)

for i in range(1,5):
    this_df=pd.read_csv(adience_path+"fold_"+str(i)+"_data.txt", sep="\t", header=0)
    adience_df=pd.concat([adience_df,this_df])
```

By analysing the dataset, it can be determined that some columns are irrelevant for this task. This data is removed, in order to save memory. It can also be determined that some entries in the set have age brackets assigned to them, which are not a part of the previously defined group. Some entries have "unknown" as their value in the "gender" column, making them useless for this project. All of the entries which do not possess desired values in the "age" and "gender" columns are removed.

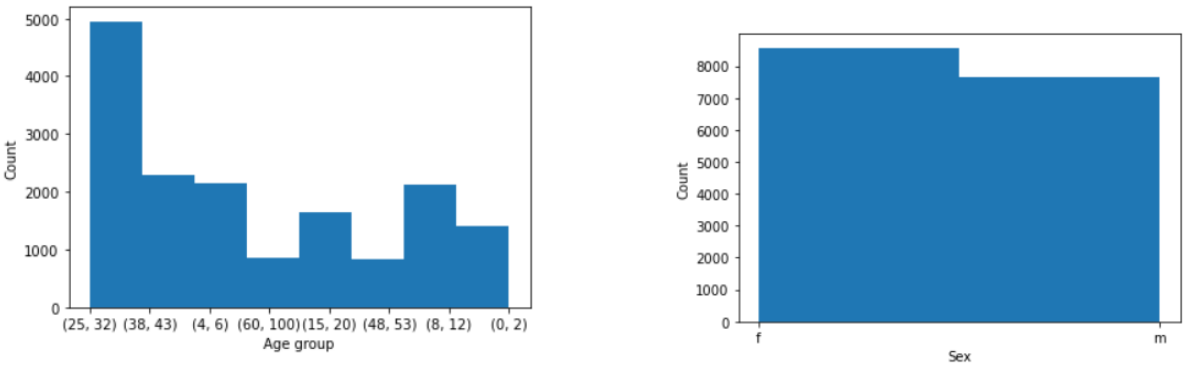
```
adience_df=adience_df.drop(columns= ["x","y","dx","dy","tilt_ang","fiducial_yaw_angle",
"fiducial_score"])
age_brackets=["(0, 2)","(4, 6)","(8, 12)","(15, 20)","(25, 32)","(38, 43)",
"(48, 53)","(60, 100)"]
adience_df = adience_df[adience_df['age'].isin(age_brackets)]
adience_df = adience_df[adience_df['gender'].isin(["f","m"])]
```

The resulting structure of the dataframe is presented on Figure 4.2

A histogram presented on Figure 4.3 reveals that the age group (25,32) is the most common, while age groups (60,100) and (48,53) are significantly less common than others. We can assume that the resulting network will most likely be significantly better at correctly recognising the age of people from

	user_id	original_image	face_id	age	gender
2733	9017386@N06	8578335271_e8afb6b51e_o.jpg	210	(0, 2)	m
1897	28754132@N06	11547428766_6f1c75a32e_o.jpg	609	(8, 12)	m
1106	20254529@N04	9405704984_0cd1df6d83_o.jpg	13	(25, 32)	f
1021	10792106@N03	11080608713_3dce2200a8_o.jpg	511	(15, 20)	m
240	115321157@N03	12112800114_33ca8b373e_o.jpg	1747	(25, 32)	m

Figure 4.2. A sample of 5 records from a cleaned dataframe



(a) Age histogram

(b) Sex histogram

Figure 4.3. Histograms presenting the age and sex distribution of samples inside the cleaned dataframe

the (25,32) age group, than people from the (60,100) and (48,53) age groups. A histogram presenting the distribution of sexes in the dataset shows that there is not a significant difference in the number of male and female records.

As the next step, an image for every entry in the dataframe is extracted from the dataset, converted into an array of numbers and inserted into the dataframe.

```
def image_to_array(row):
    user_id=row["user_id"]
    image_name=row["original_image"]
    face_id=row["face_id"]

    image_path=adience_path+"faces/faces/"+user_id+"/coarse_tilt_aligned_face."
    +str(face_id)+"."+image_name

    image = tf.keras.preprocessing.image.load_img(image_path, grayscale=False,
    target_size=(227,227))
    array = tf.keras.preprocessing.image.img_to_array(image).reshape(1, -1)[0]
    return array
```

```
adience_df['image_pixels'] = adience_df.apply(image_to_array, axis=1)
```

Afterwards, two new columns are added into the dataframe. These columns contain numbers, which translate the row's age group and sex values to their indexes in `["(0, 2)","(4, 6)","(8, 12)","(15, 20)","(25, 32)","(38, 43)", "(48, 53)","(60, 100)"]` and `["f","m"]` arrays. Those two columns are then supplied to the `keras.utils.to_categorical` function, which creates another pair of columns which contain binary class matrices.

```
target_s=keras.utils.to_categorical(adience_df['gender_int'],2)
target_a=keras.utils.to_categorical(adience_df['age_int'],8)
```

After the data is properly processed, the entire dataframe is split into testing and validation subsets, with the test size of 0.3.

```
train_s_x, test_s_x, train_s_y, test_s_y = train_test_split(features, target_s,
test_size=0.30)
train_a_x, test_a_x, train_a_y, test_a_y = train_test_split(features, target_a,
test_size=0.30)
```

Libraries used in the code in this subsection: Pandas [36], [34].

4.2.2. Neural network model implementation

Four neural network models are implemented and trained as a part of the program. These are later compared in terms of accuracy and the best one is chosen to be a part of the finished system.

4.2.2.1. Model 1 - Conv, pool, conv, pool

The first model draws inspiration from the classic LeNet-5 model [29]. It consists of multiple convolutional and pooling layers appearing after one another. A major difference between this model and LeNet-5, is the usage of ReLU activation functions, instead of Tanh.

```
if selected_name=="model1":
    selected_model = Sequential()

    selected_model.add(Convolution2D(32, (3, 3), activation='relu',padding='same',
    input_shape=(227,227,3)))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(64, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(128, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(256, (3, 3), activation='relu',padding='same'))
```

```
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(512, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))


    selected_model.add(Convolution2D(4096, (7, 7), activation='relu',padding='same'))

    selected_model.add(Convolution2D(2048, (1, 1), activation='relu',padding='same'))
```

4.2.2.2. Model 2 - Conv, pool, conv, pool with added dropout layers

The second model is the same as the first, except it has added dropout layers, in order to prevent potential overfitting.

```
if selected_name=="model2":
    selected_model = Sequential()

    selected_model.add(Convolution2D(32, (3, 3), activation='relu',padding='same',
    input_shape=(227,227,3)))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(64, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(128, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(256, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(512, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))
    selected_model.add(Dropout(0.2))

    selected_model.add(Convolution2D(4096, (7, 7), activation='relu',padding='same'))
    selected_model.add(Dropout(0.2))

    selected_model.add(Convolution2D(2048, (1, 1), activation='relu',padding='same'))
```

4.2.2.3. Model 3 - Conv, conv, pool

The third model is inspired by the VGG-16 network [30]. Like VGG-16, it consists of sets of several convolutional layers (in this case two) followed by a pooling layer. This type of network architecture is expected to have higher accuracy, but a much longer training time compared to the first 2 models.

```
if selected_name=="model3":
        selected_model = Sequential()

    selected_model.add(Convolution2D(16, (3, 3),
    activation='relu',padding='same',input_shape=(227,227,3)))
    selected_model.add(Convolution2D(32, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(64, (3, 3), activation='relu',padding='same'))
    selected_model.add(Convolution2D(128, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(256, (3, 3), activation='relu',padding='same'))
    selected_model.add(Convolution2D(512, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(2048, (7, 7), activation='relu',padding='same'))
```

4.2.2.4. Model 4 - Conv, conv, pool with added dropout layers

The last model consists of sets of two convolutional layers followed by a pooling layer, but with added dropout layers:

```
if selected_name=="model4":
    selected_model = Sequential()

    selected_model.add(Convolution2D(16, (3, 3),
    activation='relu',padding='same',input_shape=(227,227,3)))
    selected_model.add(Convolution2D(32, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(64, (3, 3), activation='relu',padding='same'))
    selected_model.add(Convolution2D(128, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))

    selected_model.add(Convolution2D(256, (3, 3), activation='relu',padding='same'))
    selected_model.add(Convolution2D(512, (3, 3), activation='relu',padding='same'))
    selected_model.add(MaxPooling2D((2,2), strides=(2,2)))
    selected_model.add(Dropout(0.2))

    selected_model.add(Convolution2D(2048, (7, 7), activation='relu',padding='same'))
    selected_model.add(Dropout(0.2))
```

After the model has been selected, a fully-connected output layer is added, with its size depending on whether the network is meant to be used for recognising sex or age. The output layer for the sex-recognising network has two output nodes, one for each sex, while the age-recognising network has eight output nodes, one for each of the previously specified age groups.

```
if selected_mode=="sex":
    selected_model.add(Convolution2D(1, (1, 1)))

    selected_model.add(Flatten())
    selected_model.add(Dense(2, activation='softmax'))
if selected_mode=="age":
    selected_model.add(Convolution2D(1, (1, 1)))

    selected_model.add(Flatten())
    selected_model.add(Dense(8, activation='softmax'))
```

Next, the model is compiled, and a checkpointer is created. The loss function of the model is Categorical Crossentropy, which is a suitable choice for a multi-classification network, such as this one. Checkpointer is able to monitor the neural network as it is being trained and it will save the best state of the model into a separate file. The metric, which the checkpointer will monitor, is the network's accuracy on validation data.

```
selected_model.compile(loss='categorical_crossentropy',
optimizer=keras.optimizers.Adam(), metrics=['accuracy'])

checkpointer = ModelCheckpoint(
    filepath='sex_recognition_'+selected_name+'.hdf5',
    monitor = "val_accuracy",
    verbose=1,
    save_best_only=True,
    mode = 'auto'
)
```

All of the aforementioned models have been constructed using the Keras library [34].

4.2.3. Network training

During the next step, all of the previously defined networks have been trained over twenty epochs, using batches of 256 inputs using the Keras [34] `fit()` function .

```
history = selected_model.fit(
    train_a_x,
    train_a_y,
    batch_size=256,
    epochs=20,
```

```
    validation_data=(test_a_x, test_a_y),
    callbacks=[checkpointer]
)
```

The sex recognition networks have been trained before the age recognition networks. Afterwards, the trained sex recognition networks were loaded into the program again, including their weights. From each of these networks, the output layer has been removed and a new one was added, allowing the already trained sex recognition network to be partially re-purposed for the task of age recognition.

```
selected_model.load_weights('sex_recognition_'+selected_name+'.hdf5')
selected_model.pop()
```

Afterwards, a specific number of the top layers in the network is made to be un-trainable. The reason for this, is that we can assume that by being already trained with a large amount of data, those top layers have gained a decent level of ability to detect low-level features on the image, which enable the network to perform sex recognition. It is assumed that the ability to detect those same low-level features will enable the resulting age-recognition network to also perform well. By re-using the trained weights on top layers of the network, we also greatly reduce the training time. The amount of top layers which are modified to be un-trainable varies depending on the architecture.

```
for layer in selected_model.layers[:-7]:
    layer.trainable=False
selected_model.add(Dense(8, activation='softmax'))
selected_model.summary()
```

The results of the network training are presented on figure 4.4.

	Model1	Model2	Model3	Model4
Age prediction accuracy	50.21	57.9	55.77	52.35
Sex prediction accuracy	85.06	91.02	89.84	85.19
Age prediction loss	1.2496	0.6103	0.728	0.074
Sex prediction loss	0.2742	0.1294	0.1294	0.2574

Figure 4.4. Results of network training.

4.3. System implementation

After the training has been completed, an automatic age and sex recognition system, which additionally utilises a face detection network, has been created. This program is also written in the Jupyter Notebook.

4.3.1. Network initialisation

At the start of the program, the user has the option to manually choose networks used in the age and sex recognition. Model 2 is the recommended one, as it has achieved the highest accuracy on validation data in both tasks.

```
age_model_name="model2"
sex_model_name="model2"
```

Additionally , the pre-trained model of MTCNN [39] is loaded.

```
detector = MTCNN()
```

4.3.2. Face detection and image modification

Figure 4.5. Source image.

First, MTCNN [39] is used to detect faces and their keypoints on a previously defined image.

```
img = cv2.imread(image_path)

faces = detector.detect_faces(img)
```

Next, the coordinates of detected faces are used to cut fragments of the source image containing faces. An additional border of 40 pixels is added, in order to include the hair of a detected person, as the images which the networks were trained on also contained hair. The result of this operation is presented on figure 4.6.

```
for face in faces:
    x, y, w, h = face['box']
    x1, y1 = x + w, y + h
    face_cut = img[(y-40):(y1+40), (x-40):(x1+40)]
```

The cut face is then aligned, so that eyes of the detected person are on the same level, using the `face_align()` function.

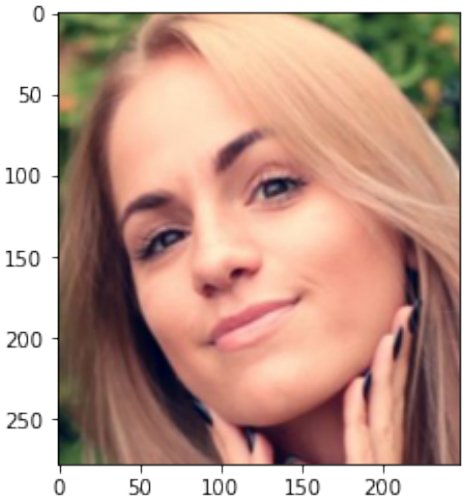


Figure 4.6. A face detected and cut from the source image.

```
def face_align(image,left_eye,right_eye):
    if left_eye[1]< right_eye[1]:
       point = (right_eye[0], left_eye[1])
       clockwise = True
    else:
       point = (left_eye[0], right_eye[1])
       clockwise = False
    a = distance(left_eye, point)
    b = distance(right_eye, left_eye)
    c = distance(right_eye, point)
    cos_a = (b*b + c*c - a*a)/(2*b*c)
    angle = np.arccos(cos_a)
    angle = (angle * 180) / math.pi
    pil_image = Image.fromarray(image)
    if clockwise:
        angle = 90 - angle
        return np.array(pil_image.rotate(-1 * angle))
    else:
        return np.array(pil_image.rotate(angle))

face_aligned=face_align(face_cut,face['keypoints']['right_eye'],
face['keypoints']['left_eye'])
face_aligned=Image.fromarray(face_aligned)
```

After the aligning, the picture is resized, so that it matches the size of the sex and age recognition networks' input layers.

```
face_aligned=face_aligned.resize((227, 227))
face_aligned=np.array(face_aligned)
```

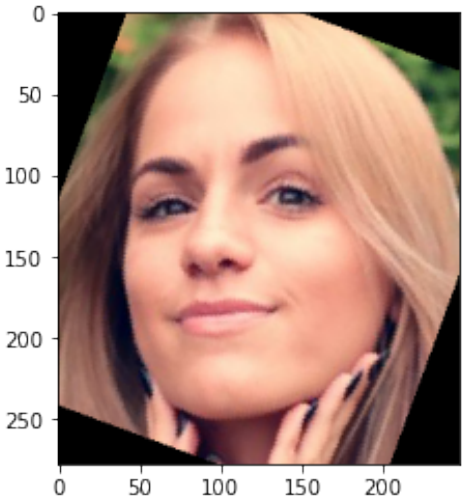


Figure 4.7. Detected face after aligning.

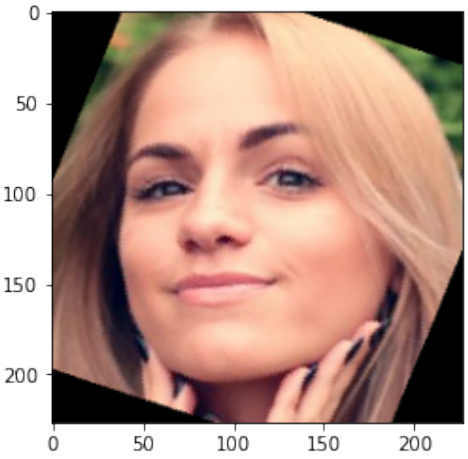


Figure 4.8. Detected face after resizing.

Finally, the last data conversions are made and afterwards, the picture is ready to be evaluated by the age and sex recognition models.

```
test_img = np.expand_dims(face_aligned, axis = 0)
test_img = test_img.astype(np.float32)
test_images.append(test_img)
face_coords.append([x,y,x1,y1])
```

4.3.3. Age and sex detection

Once the cut pictures have been modified, the entire source photo is displayed.

```
plt.figure()
fig,ax = plt.subplots(1, figsize=(12,6))
ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

Every cut fragment containing a face is then evaluated by both networks.

```
for i in range(0,len(test_images)):
    prediction_age=age_model.predict_classes(test_images[i])
    prediction_sex=sex_model.predict_classes(test_images[i])
    predicted_age_class=age_classes[prediction_age[0]]
    predicted_sex_class=sex_classes[prediction_sex[0]]
```

A bounding box is drawn around every detected face and labels predicted by the networks are displayed above them.

```
    x,y,x1,y1=face_coords[i]

    rect = patches.Rectangle((x,y),x1-x,y1-y,linewidth=2,edgecolor='r',facecolor='none')
    ax.add_patch(rect)
    plt.text(x+10,y-10,predicted_age_class+",␣"+predicted_sex_class,c="r", weight='bold')

plt.show()
```

The end result of the program is shown on figure 4.9.

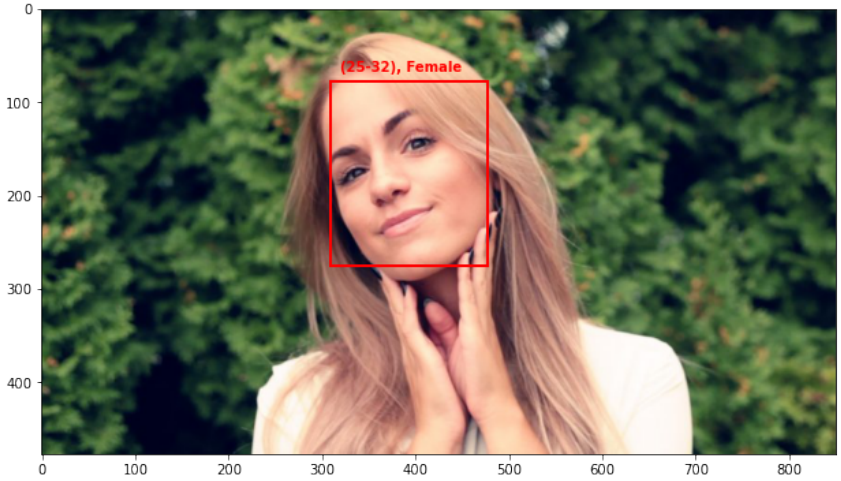


Figure 4.9. End result of the program

5. Discussion

5.1. Accuracy analysis

All of the trained networks have achieved satisfactory results. The best accuracy has been achieved by the second model. A significant boost of over 7% and over 5% in age and sex recognition accuracy, has been achieved by adding two drop-out layers with 20% drop rate. As expected, this has prevented the network from over-fitting prematurely.

In the case of the 4th model, adding drop-out layers has resulted in a lowered performance, in comparison to model 3. The lower accuracy of model 4 was not caused by over-fitting, but most likely by low training time. It can be assumed, that this network would have achieved better results, if it had been trained for more than twenty epochs, unlike other networks, which have already started severely over-fitting by the twentieth epoch.

A sample of outputs returned by the program is presented on figures 5.1-5.4.

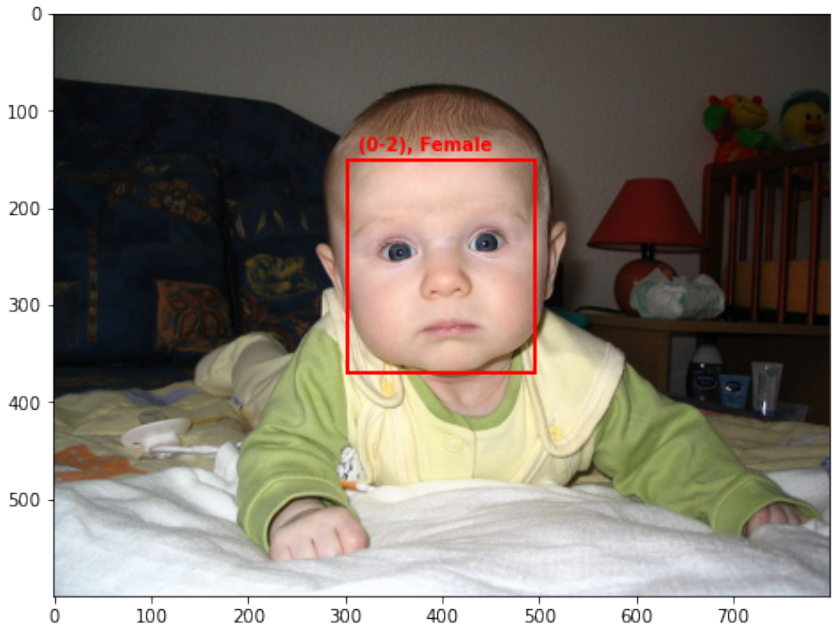


Figure 5.1. Example of a correct output.

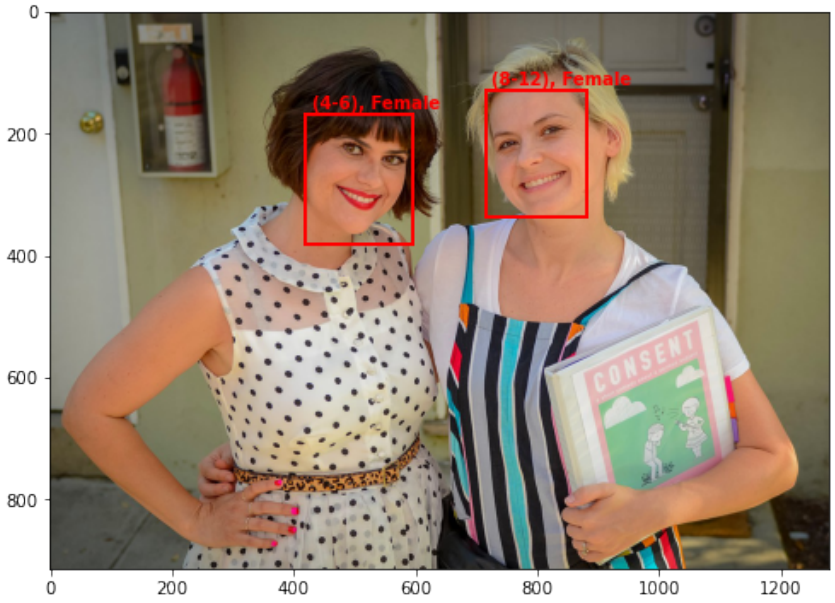


Figure 5.2. Example of a partially correct output. (Sexes of detected people are recognised correctly, while age groups are incorrect.)

Figure 5.3. Example of a partially correct output. (Some of the detected faces have both labels correct, some have both incorrect.)

5.2. Potential improvements

Created system would most likely become more accurate if the following changes were made:

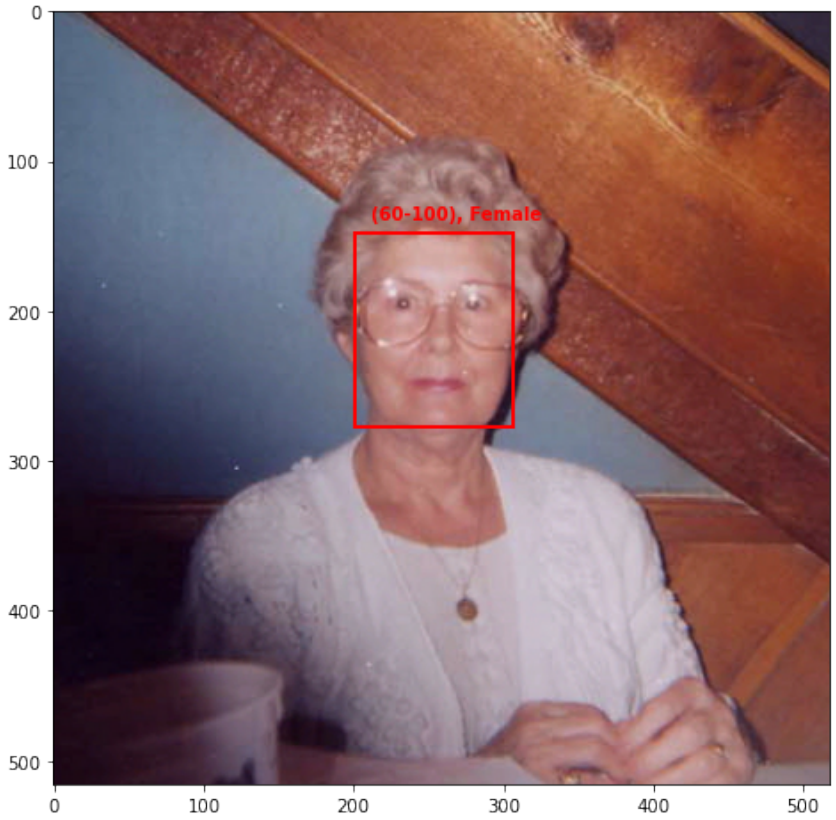


Figure 5.4. Example of a correct output.

1. Both model 2 and 4 could most likely benefit from longer training time. If a much longer training time was used, changing the drop-out layers' drop rate from 20% to 40% or 50% would further reduce the risk of model over-fitting, thus increasing the potential maximum accuracy a network can achieve.

2. A greater amount of convolutional layers would most likely result in better accuracy in all of the models, although it would also greatly increase training time per data sample.

3. Data used for training the network could be augmented, increasing the number of training samples, which would potentially improve the accuracy of all the models.

5.3. Comparison to similar solutions

An accuracy comparison to other scientific papers has been presented on figure 5.5. Due to the fact, that the age recognition system presented in this paper was utilising specifically defined age-groups, a comparison of its accuracy with solutions, which utilise different age groups or attempt to recognise the exact age of a photographed person would not provide much insight.

Models 2, 3 and 4 have achieved better age recognition accuracy, than a similar solution utilising CNNs presented in [20]. Age and sex recognition networks presented in [20] were trained on the same dataset, therefore it is the most fitting target for comparison.

All of the models trained in this paper have achieved considerably worse sex recognition accuracy than a solution presented in [27]. In this paper, a classifier trained on 4 million images has achieved an accuracy rate of 96.8%, which is the highest sex recognition accuracy achieved so far.

A different classifier presented in [28], utilising Precise Patch Histogram, has achieved accuracy of 79.53% at sex recognition, which is lower by at least 4.53% compared to all of the models presented in this paper.

	Model1	Model2	Model3	Model4	Best from [20]	Best from [27]	Best from [28]
Age prediction accuracy	50.21	57.9	55.77	52.35	50.7	N/A	N/A
Sex prediction accuracy	85.06	91.02	89.84	85.19	86.8	96.8	79.53

Figure 5.5. Prediction accuracy comparison.

6. Summary

The goal of this paper was to create an age and sex recognition system, using deep learning. This goal has been fulfilled, and the created program has achieved good results in comparison to systems presented in other papers. Notably, it has managed to achieve better results in both age and sex recognition than a different age and sex recognition system, which has been trained on the same dataset.

A total of 4 models has been trained, with the best one achieving 57.9% accuracy in age prediction and 91.02% accuracy in sex prediction. A comparison of models presented in this paper and models presented in other scientific papers is shown on figure 5.5.

Additionally, an introduction to the theory of neural networks has been presented, as well as an overview of face detection, sex recognition and age recognition methods and their real-life usages.

While convolutional neural networks can achieve very good results at sex recognition based on face photos, age recognition seems to be a problem which is too difficult to reliably solve using that method. This can be explained by the fact that different people age at different rates. This aging rate depends on data which may be impossible to extract from a portrait. Such data includes dietary habits, tendency to smoke cigarettes or drink alcohol and living environment. People may also attempt to conceal their age using make-up or plastic surgeries.

Nevertheless, it has been assumed that improvements could be made, in order to raise the accuracy of the system. One of such improvements is using a larger labeled data set or using techniques of data augmentation in order to produce more training data. Another way to improve the networks, could be raising the training time, as two of the networks trained in this paper, did not yet start over-fitting during their training, so it may be assumed that additional training time could improve them.

Evidently, usage of machine learning techniques in the field of computer vision is gaining a lot of popularity in recent years and with growing computational power and increasing ability of companies to collect data about the world and people, this trend can be expected to continue.

Bibliography

[1] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. 2nd edition. Upper Saddle River, NJ: Prentice Hall, 1999.

[2] Lucas Cinelli, Gabriel Chaves, and Markus Lima. "Vessel Classification through Convolutional Neural Networks using Passive Sonar Spectrogram Images". In: May 2018. DOI: *10.14209/sbrt.2018.340*.

[3] *The vanishing gradient problem. https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484*.

[4] Bing Xu et al. *Empirical Evaluation of Rectified Activations in Convolutional Network*. 2015.

[5] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. *http://www.deeplearningbook.org*. Cambridge, MA, USA: MIT Press, 2016.

[6] *Backpropagation*. *https://brilliant.org/wiki/backpropagation/*.

[7] Kenji Suzuki. *ARTIFICIAL NEURAL NETWORKS – ARCHITECTURES AND APPLICATIONS*. Jan. 2013. ISBN: 978-953-51-0935-8.

[8] Paul Viola and Michael Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". In: vol. 1. Feb. 2001, pp. I–511. ISBN: 0-7695-1272-0. DOI: *10.1109/CVPR.2001.990517*.

[9] Ming-Hsuan Yang, David Kriegman, and Narendra Ahuja. "Detecting Faces in Images: A Survey". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24 (Feb. 2002), pp. 34 –58. DOI: *10.1109/34.982883*.

[10] Mejda Chihaoui et al. "A Survey of 2D Face Recognition Techniques". In: *Computers* 5 (Sept. 2016). DOI: *10.3390/computers5040021*.

[11] Masakazu Matsugu et al. "Subject independent facial expression recognition with robust face detection using a convolutional neural network". In: *Neural Networks* 16.5-6 (2003), pp. 555–559.

[12] Yi-Qing Wang. "An analysis of the Viola-Jones face detection algorithm". In: *Image Processing On Line* 4 (2014), pp. 128–148.

[13] K. Kadir et al. “A comparative study between LBP and Haar-like features for Face Detection using OpenCV”. In: *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)* (2014), pp. 335–339.

[14] Jacob Whitehill et al. “Toward practical smile detection”. In: *IEEE transactions on pattern analysis and machine intelligence* 31.11 (2009), pp. 2106–2111.

[15] Beat Fasel and Juergen Luettin. “Automatic facial expression analysis: a survey”. In: *Pattern recognition* 36.1 (2003), pp. 259–275.

[16] Byoung Chul Ko. “A brief review of facial emotion recognition based on visual information”. In: *sensors* 18.2 (2018), p. 401.

[17] Kresimir Delac and Mislav Grgic. “A survey of biometric recognition methods”. In: *Proceedings. Elmar-2004. 46th International Symposium on Electronics in Marine*. IEEE. 2004, pp. 184–193.

[18] Anil K Jain and Stan Z Li. *Handbook of face recognition*. Vol. 1. Springer, 2011.

[19] Shireesha Chintalapati and MV Raghunadh. “Automated attendance management system based on face recognition algorithms”. In: *2013 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE. 2013, pp. 1–5.

[20] Gil Levi and Tal Hassner. “Age and gender classification using convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2015, pp. 34–42.

[21] Baback Moghaddam and Ming-Hsuan Yang. “Learning gender with support faces”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.5 (2002), pp. 707–711.

[22] Matthew Toews and Tal Arbel. “Detection, localization, and sex classification of faces from arbitrary viewpoints and under occlusion”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.9 (2008), pp. 1567–1581.

[23] Yun Fu, Guodong Guo, and Thomas S Huang. “Age synthesis and estimation via faces: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* 32.11 (2010), pp. 1955–1976.

[24] Guodong Guo et al. “Locally adjusted robust regression for human age estimation”. In: *2008 IEEE Workshop on Applications of Computer Vision*. IEEE. 2008, pp. 1–6.

[25] Feng Gao and Haizhou Ai. “Face age classification on consumer images with gabor feature and fuzzy lda method”. In: *International Conference on Biometrics*. Springer. 2009, pp. 132–141.

[26] Eran Eidinger, Roee Enbar, and Tal Hassner. “Age and gender estimation of unfiltered faces”. In: *IEEE Transactions on Information Forensics and Security* 9.12 (2014), pp. 2170–2179.

[27] Sen Jia and Nello Cristianini. “Learning to classify gender from four million images”. In: *Pattern recognition letters* 58 (2015), pp. 35–41.

[28] Juan Bekios-Calfa, José M Buenaposada, and Luis Baumela. “Robust gender recognition by exploiting facial attributes dependencies”. In: *Pattern recognition letters* 36 (2014), pp. 228–234.

[29] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[30] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[31] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[32] Marti A. Hearst et al. “Support vector machines”. In: *IEEE Intelligent Systems and their applications* 13.4 (1998), pp. 18–28.

[33] Duy Nguyen et al. “Real-time face detection and lip feature extraction using field-programmable gate arrays”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 36.4 (2006), pp. 902–912.

[34] *Keras - Deep learning API. https://keras.io/*.

[35] *Tensorflow - An end-to-end open source machine learning platform. https://www.tensorflow.org/*.

[36] *Pandas - Open source data analysis and manipulation tool. https://pandas.pydata.org/*.

[37] *Numpy - Scientific computing package. https://numpy.org/*.

[38] *OpenCV - Computer Vision library. https://opencv.org/*.

[39] *MTCNN - Implementation of MTCNN face detection network in Python for Keras. https://github.com/ipazc/mtcnn*.