University of Thessaly
Volos, February, 2020

Gaze Estimation Algorithms using low cost Cameras

Αλγόριθμοι Αναγνώρισης Κατεύθυνσης Βλέμματος με Χρήση Καμερών Χαμηλού Κόστους

Christos Axelos

Supervisor: Gerasimos Potamianos

Committee Members: Nikolaos Bellas, Antonios Argyriou

Diploma Thesis

Department of Electrical and Computer Engineering

University of Thesssaly

Volos, Greece

This thesis was submitted for the acquisition of the Diploma of the School of Electrical and Computer Engineering.

Declaration of Authorship

I, Christos Axelos, declare that this diploma thesis entitled "Gaze Estimation Algorithms using Low-Cost Cameras" and the work presented in it are my own. I certify that:

- This work was carried out entirely or mainly during my candidacy for the diploma at this University.
- Where any part of this thesis has previously been submitted for a degree or other qualification at this or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have cited the work of others, the source is always given. Except for such citations, this thesis is entirely my own work.
- I have acknowledged all major sources of assistance.
- Where the work is based on work performed jointly with others, I have clearly stated what was done by others and what I contributed myself.

University of Thessaly

Greece, February 2020

Christos Axelos

Abstract

Deep learning and particularly neural networks have offered a new point of view in problem solving. These techniques have been heavily adopted by many applications used in daily life or industry. *Gaze estimation* belongs to this category of applications. Since neural networks have been used in order to solve problems related to gaze estimation, they continuously provide solutions that outperform the previous ones.

This thesis proposes a neural network architecture based on the popular *residual networks* (ResNets), a novel convolutional network introduced in ILSVRC [1] (2015). Specifically, the proposed method is a ResNet-20 network, which achieves competitive performance compared to the literature. This architecture achieves desirable performance regardless of the environmental conditions (in-the-wild) or the facial characteristics and it can also operate well without any calibration techniques.

Finally, this solution can substitute the use of expensive, special hardware when high accuracy is not necessary. As a result, reducing the production cost can make these applications accessible not only to specialized users, but to everyone with a laptop and a web camera.

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor, Prof. Gerasimos Potamianos, for his guidance throughout the preparation of this thesis. His suggestions and approach helped me successfully complete the work and gain a stronger academic foundation.

I would also like to thank my co-supervisors, Prof. Nikolaos Bellas and Prof. Antonios Argyriou, as well as the other professors of the department who supported me throughout my studies.

Furthermore, I am deeply grateful to my parents and my sister, who stood by me and supported me in every way during my academic journey.

Finally, I owe a big thank you to my friends and fellow students, who provided useful technical advice and ideas that helped me complete this thesis.

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1

Introduction

1.1 Gaze Direction Estimation

We define as *Gaze Direction Estimation* the process of determining **where** a person is looking. This process is used in various applications related to human–computer interaction [2], virtual reality, the video game industry, and attention analysis [3]. To record human eye activity, a camera-based application is required.

As described in [4], gaze estimators can be classified into two major categories, depending on the **position of the camera in space**. In the first category, we have the *near-eye cameras*, which are attached to head-mounted devices, such as virtual reality headsets. They can capture high-resolution images because of their close proximity to the eyes. In the second category, we have the *remote image estimators*. Here, images are captured by a camera fixed at a specific position in space—for example, a laptop webcam. The captured images usually have lower resolution due to the larger distance between the eye and the camera lens. However, in this case the user is not required to wear any device on the head. In this thesis, we adopt the *remote image estimator* approach using a *standard computer camera*.

Another key criterion for classifying gaze direction estimators is whether or not they require *person-specific camera calibration* each time a new user employs the application. Adjusting the internal parameters of the camera for each user can increase system accuracy, but this process is time-consuming and requires distinguishing between users' faces, thus

increasing complexity. For this reason, in this thesis we avoid per-user calibration, instead setting *fixed approximate values* for the internal camera parameters from the outset.

Finally, gaze estimators can also be divided into two additional categories. The first category includes the *specialized hardware–based estimators* (e.g., high-resolution cameras or infrared trackers). These can achieve high accuracy but are expensive and inaccessible to many people. On the other hand, we have the *general-purpose software-based estimators*, which avoid specialized devices and instead use the resources of a standard computer. Although their accuracy is typically lower, advances in computer vision and artificial intelligence now provide increasingly reliable solutions. In this thesis, we follow the second approach, avoiding specialized hardware.

1.2 Our approach

Previous works on gaze direction recognition have shown that the use of neural networks can result in a considerable increase in performance compared to similar works that used traditional methods, such as the methods of Adaptive Linear Regression (ALR) [5], Support Vector Regression (SVR) [6], k-Nearest Neighbors (kNN) and Regression Forests (RF) [7]. For example in [8], the use of a Multimodal Convolutional Network improved performance by about **6%** compared to the best performance of Regression Forests [7], while in [9] the use of a VGG-16 network increased performance by **18%** on real-world data and by **19%** on the GazeNet dataset compared to Regression Forests [7]. A similar approach with even better performance was proposed in [10]. However, in that work, the authors used the variant of *fully pre-activated ResNets* (fully preactivated ResNets), focusing the implementation on mobile devices.

Observing these results from the literature as well as the fact that the continuously improved architectures of neural networks seem quite promising, it was considered good to turn in this direction. For this reason, in this thesis we propose the use of the **ResNet** architecture [11], which has dominated the ILSVRC [1] competition in 2015 and is considered one of the best-performing architectures to date.

1.3 Contribution of the Diploma Thesis

This diploma thesis presents the **ResNet-20** architecture, a convolutional neural network architecture based on the *ResNet* (Residual Network) [11]. The ResNet network features shortcut paths (residual connections), an approach that, until now, has not been applied in the literature for the problem of gaze direction estimation (gaze tracking) in space. The performance of the proposed method is very close to that of the most recent works in the literature.

Additionally, we provide an implementation related to how the information of the gaze direction can be utilized so that a computer can detect the point on the screen at which a user is looking (gaze tracking).

1.4 Structure of the Thesis

The remainder of the thesis is organized as follows:

- In Chapter 2 we describe in detail the process that generates the input of the ResNet-20 network.
- In Chapter 3 we describe the proposed ResNet-20 network.
- In Chapter 4 we describe an application that exploits gaze-direction information.
- In Chapter 5 we evaluate the performance of ResNet-20 and compare it with the literature.
- Finally, in Chapter 6 we summarize and provide notes that extend our methodology.

Chapter 2

Input Data Preprocessing

In this chapter, we explain in detail the preprocessing of the data that serve as input to the proposed ResNet-20 network. The process begins with the capture of a real-time image from the computer's camera and ends with a final transformation that prepares the data for network input.

Before describing the preprocessing steps, we must first define the three coordinate systems used for feature extraction and show how to transform between them, following the definitions in [7].

2.1 Definition of Coordinate Systems

2.1.1 World Coordinates

The first coordinate system is the 3D world coordinate system. The origin of its axes is a 3-dimensional point located on the object of interest. In our case, the object is the human face, and the reference point is defined as the midpoint between the two eyes.

2.1.2 Camera Coordinates

The second system is the 3D camera coordinate system, which uses the camera as its reference point. A point in world coordinates can be transformed into camera coordinates if we know the extrinsic camera parameters: the *rotation matrix* (R) and the *translation*

vector (t) that relate the object to the camera.

For example, if a point has world coordinates (U,V,W), rotation matrix $\boldsymbol{R}$ and translation vector $\boldsymbol{t}$, then it's camera coordinates (X,Y,Z) are given by (2.1).

$$\begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \end{bmatrix} = \boldsymbol{R} \begin{bmatrix} \mathrm{U} \\ \mathrm{V} \\ \mathrm{W} \end{bmatrix} + \boldsymbol{t} \tag{2.1}$$

2.1.3 Screen Coordinates

Lastly, the third system is the 2D screen coordinate system. We can easily obtain screen coordinates from camera coordinates, if we know the *intrinsic camera parameters*, namely the focal lengths (f_x, f_y) and the optical centers (c_x, c_y).

For example, having obtained from (2.1) the camera coordinates (X, Y, Z) and since we know the focal lengths (f_x, f_y) and the optical centers (c_x, c_y), then the projected point in the screen coordinate system is given by the equation (2.2).

$$\begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ 1 \end{bmatrix} = \mathbf{s} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \end{bmatrix} \tag{2.2}$$

The coefficient **s** (scale) denotes a scale factor that represents the depth of an image. The equation (2.3) shows us how we can transform the world coordinates (right side) into the camera coordinates (left side), provided that we know the rotation matrix and the translation vector.

$$\mathbf{s} \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} \mathrm{U} \\ \mathrm{V} \\ \mathrm{W} \\ 1 \end{bmatrix} \tag{2.3}$$

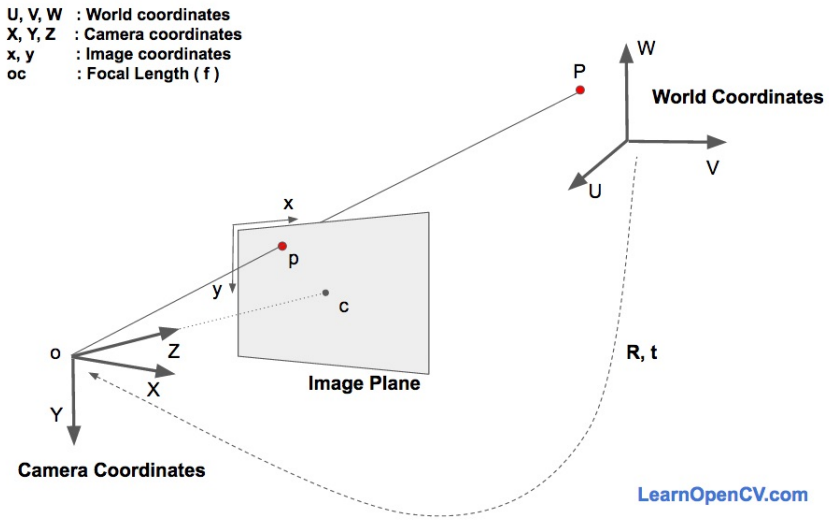

Figure 2.1: The 3 coordinate systems discussed in the section 2.1. (Figure from [12]).

Figure 2.1 shows us the relationship between these 3 coordinate systems.

2.2 Feature extraction

Feature extraction follows these steps. First, we detect whether the captured image contains a human face. If a face is detected, we then identify *6 key facial landmarks* using pre-trained models. Next, we try to extract the head pose by mapping these six landmarks to an already known 3D face model in world coordinates. Finally, we preprocess the captured face image and head pose to ensure they are compatible with the databases we will use.

2.2.1 Face detection

Initially, we capture an image from the computer's camera. Then, we use a pre-trained face detector from the Dlib tool [13]. This face detector is based on the HoG algorithm [14].

Once we confirm that there is indeed a face in the image, we attempt to extract information regarding the position of the head in three-dimensional space.

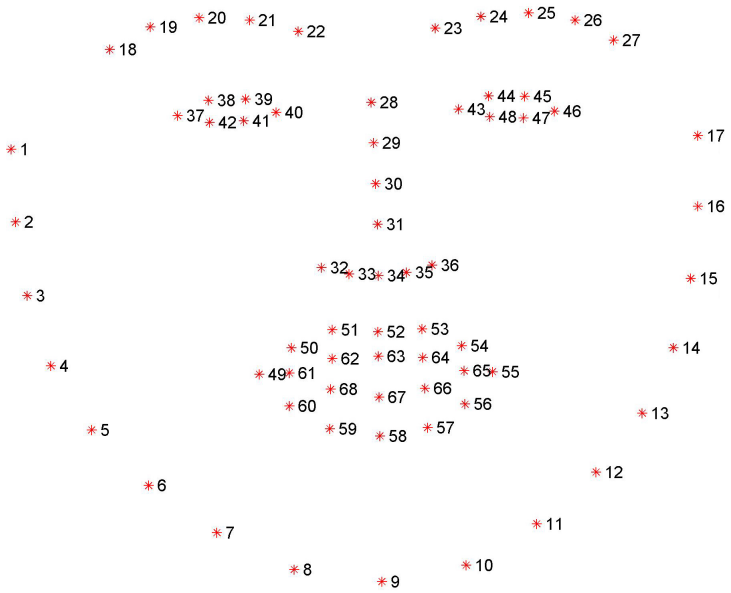

Figure 2.2: The image shows the 68 points detected by the algorithm [15]. We will need 6 of them, points 37, 40, 43, 46, 49, and 55. (Image from[15]).

2.2.2 Detection of key points in the face image

The next step is to detect 6 characteristic points that appear on the two-dimensional image of the face. We follow the same process as in [7] and calculate 6 precise points, as measurements showed that this number is the minimum required for the satisfactory extraction of the head's position. These points are the 2 corners of each eye and the 2 corners of the mouth, which are shown in figure 2.2. The detection of these points was done using a trained 68-point face model (landmark detector) provided in [15] and is based on AOMs (Active Orientation Models) [16].

2.2.3 Head pose detection

The calculation of $\boldsymbol{R}$ is done as follows. First, we estimate the internal parameters of the camera (f_x, f_y, c_x, c_y) as done in [8]. Then, from (2.3) we observe that if we know the appropriate $\boldsymbol{R}$ $\boldsymbol{t}$, we can *project* the 3D face model of 6 points from the world coordinates onto the image coordinates. Next, we attempt to minimize the *reprojection error* created between the 6 projected points and the 6 detected points using the algorithm in [15] so that the 6 points of the model align with those detected. The goal is to find the appropriate $\boldsymbol{R}$ $\boldsymbol{t}$ so as to minimize the projection error. At this point, we apply the Levenberg–Marquardt optimization [17] to compute the optimal $\boldsymbol{R}$ and $\boldsymbol{t}$. This entire

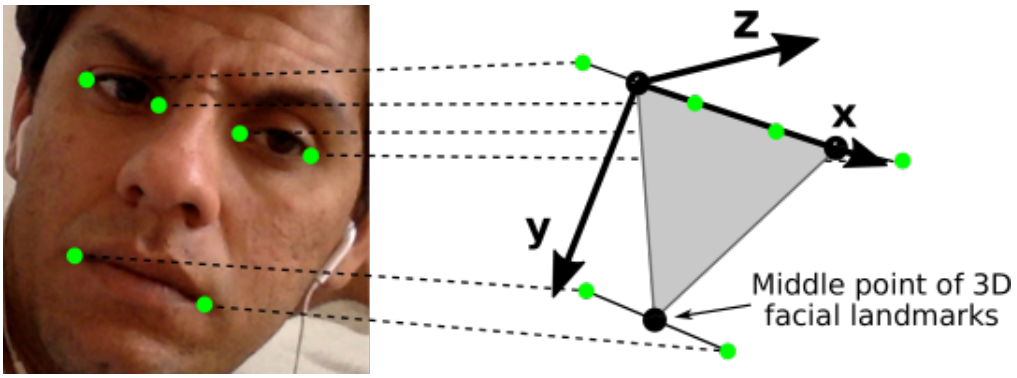

Figure 2.3: On the left are the 6 points detected by the point-detection algorithm in the image. On the right is the face model that has undergone translation and rotation so that the reprojection error is minimized. The horizontal line expresses the plane formed by the eyes and the mouth, while the vertical line (not shown in the figure) provides information about the gaze. (Image from [9]).

process is carried out through the function `solvepnp()` of OpenCV [18].

Once we compute $\boldsymbol{R}\ \ \boldsymbol{t}$, we project the 6-point face model into the camera coordinates according to equation (2.3). Then, we compute the midpoint of each eye and the midpoint of the mouth in the camera coordinates. Finally, we can represent the gaze as the 2D vector $(\hat{p}_{hor}, \hat{p}_{vert})$, where $\hat{p}_{hor}$, $\hat{p}_{vert}$ are the horizontal and vertical angles respectively, which are formed between the following lines. On the vertical line in the plane formed by the centers of the eyes and the mouth, and on the line connecting the face to the camera. This is illustrated more clearly on the right side of 2.3.

2.3 Data transformation in the normalized camera system

At this point, we transform the input image and the gaze, following the same procedure as in [19]. In summary, the goal of this transformation is to convert the data from the original camera system to the *normalized camera system*, as it is called by the authors of [19]. It has been shown that with this method the differences caused by the gaze are better eliminated than when using the raw data. A more detailed discussion of this transformation can be found in [19].

After the transformation is complete, we crop the original image so that we isolate the image of the right or left eye. Figure 2.4 shows how this transformation is carried out.

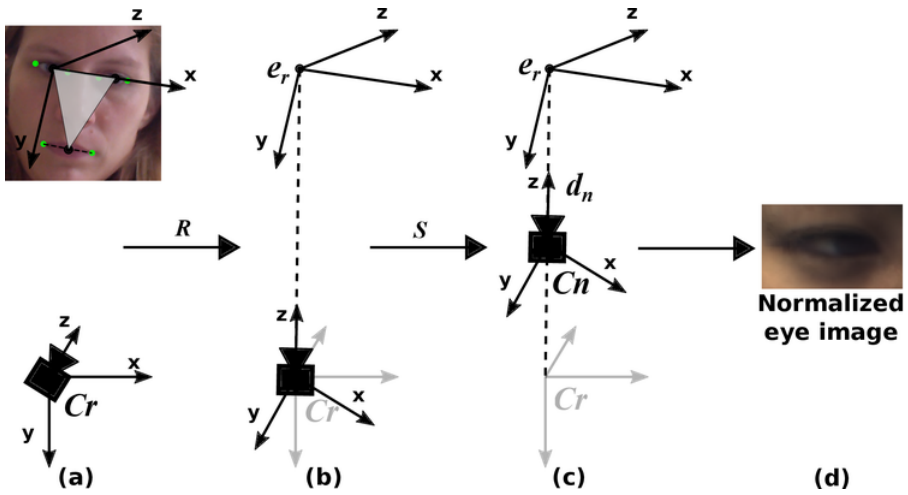

Figure 2.4: Conversion of the input image (a) into its normalized form (d). The transformation applied removes rotations and keeps only the eye region. A rotation (b) and scaling (c) are applied, and finally (d) cropping is performed. The matrix $\boldsymbol{C_r}$ contains the internal parameters of the camera, the matrix $\boldsymbol{R}$ is used as a transformation matrix of the original rotation matrix into the camera coordinate system. The matrix $\boldsymbol{C_n}$ serves a similar function but for the internal parameter matrix $\boldsymbol{C_r}$ of the camera, while the matrix $\boldsymbol{S}$ performs enlargement of the image. (Image from [19]).

Apart from the image, we also observe the effect of this transformation not only on the gaze but also on the direction of the gaze. These changes are due to the parameters $\boldsymbol{C_r, R, S, C_n}$ of figure 2.4.

After the end of the transformation, we use the transformed versions of the eye image and the head pose as input to the proposed ResNet-20 network. As output from this network, we obtain the normalized gaze direction. This process is described in detail in Chapter 3.

Chapter 3

The proposed ResNet-20 network

After having obtained from chapter 2 the normalized head pose and eye image, we attempt with these two variables to predict the *normalized direction* of gaze, that is, the direction of gaze in the coordinate system of the image as described in Section 2.3. To achieve this, we employ a convolutional neural network (CNN) of the *Residual Network* (ResNet) family [11]. To be more specific, we use a **ResNet-20** architecture with specific modifications to the original network, in order to suit our gaze estimation problem. The number 20 represents the total *number of convolutional layers* of the model.

3.1 Brief Theoretical Background

The ResNet network [11] is a convolutional neural network architecture proposed in ILSVRC [20] in 2015. It became known because it successfully addressed to a considerable degree the problem of *gradient vanishing* when training deep convolutional layers (vanishing gradients problem). Furthermore, it has been proven that their performance is similar to other networks, but with much less convolutional layers needed (less depth).

The differences of these networks compared to the well-known architectures are as follows. Firstly, the convolutional layer is replaced with the *basic block*. The basic block consists of two triplets of operations, where each triplet consists of a 3×3 convolution, a batch normalization [21], and an activation function ReLU. Often, instead of two 3×3 convolutions, the option with the 3 convolutions (1×1, 3×3, 1×1) is chosen for better time performance (bottleneck option).

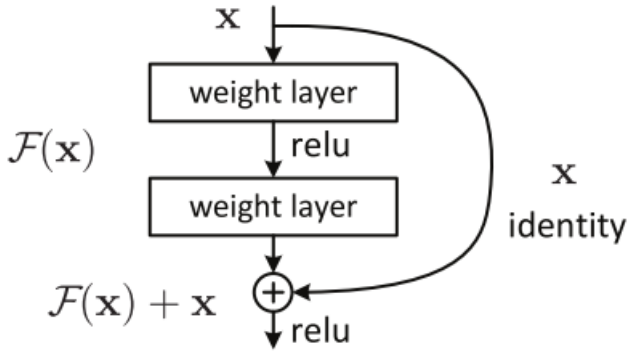

Figure 3.1: In the image, each box (weight layer) consists of a 3×3 convolution and a batch normalization [21]. The curved line represents the *shortcut path*.
. (Image from [11]).

In addition, the concept of the *shortcut path* is introduced. These shortcuts bypass convolutional layers and the identity function is applied to them. Then they are added to the output of the *basic block*. If the number of channels of the shortcut and the basic block differ, then instead of the identity connection the *residual block* is applied. The residual blocks consist of a 1×1 convolution and a batch normalization [21].

In figure 3.2 we see the basic block as well as the shortcut path.

3.2 Parameter calculation of ResNet-20

The **ResNet-20** network was derived after experimenting with the basic parameters of the ResNet networks. Using some initial values for the parameters, we attempted to change these values one by one. As soon as we found the optimal value for one parameter, that is, the value that produced the smallest mean error, we fixed that value and attempted to change the next parameter.

We tried to change the parameters in a specific order. Priority was given to the parameters that affect the initial stages of the algorithm, such as the parameters of the initial convolution of the network (gate block). Then we dealt with the parameters of the remaining convolutional layers of the network and left for last the parameters related to the fully connected layers. For each convolutional layer we added, we checked which values of the layer's parameters gave better performance. If the current layer's parameter values did not yield better performance, we stopped adding layers. Otherwise, we kept

the current layer and its best parameters and repeated the same process for the next convolutional layer. In a similar way, we dealt with the parameters of the fully connected networks, such as the number of layers and neurons.

The parameters of the network on which we experimented are as follows:

- **Network version ResNet**. We selected the basic version of the algorithm, while we also tested the versions *ReLU before addition* and f*ull pre-activation* [3].
- **Filter size of the initial convolution** (gate block convolution). We tested the values 3, 5, 7, 9. The optimal value chosen was **7**.
- **Number of output channels of the initial convolution** (number of output channels of gate block). We tested the values 16, 32, 64, 128 and the optimal value chosen was **64**.
- **Number of convolutional and fully connected layers**. The number of these layers was computed using the method described in the second paragraph of section 3.2. Specifically, we used **1** initial convolution, **16** convolutional layers and **3** fully connected layers in the main path of the network. In addition, at **4** points in the network there are four shortcut convolutions applied to the shortcut paths.
- **Output channel values per layer**. We experimented with various values between 16 and 1024 for the convolutional layers and between 128 and 2048 for the fully connected layers. The best output channel values selected were **64, 128, 256, 512** for the convolutional layers and **512** for the fully connected layers.

3.3 Operation of the ResNet-20 network

Schematically, the network is shown in 3.2. Initially, the image with dimensions 60×36 passes through a convolutional layer (yellow color in Figure 3.2) in order to increase the number of output channels. Then, we apply 16 convolutional layers (green color in Figure 3.2). In some of these layers (dark green color in 3.2) the number of channels increases from 64 to 512. Moreover, at four points of the network there are four shortcut connections (olive color in Figure 3.2) that are applied to the residual paths (shortcuts).

Next, three fully-connected networks are applied (turquoise color in Figure 3.2), where

between the first and the second we insert the vector of poses $(\hat{p}_{hor}, \hat{p}_{vert})$ that we calculated in section 2.2.3. The third fully-connected network produces as output the two-dimensional direction of the gaze in the normalized camera system.

Finally, we apply the transformation mentioned in section 2.3 but in reverse, that is, from the normalized coordinate system to the synthetic coordinate system, as done in [19]. The result of this transformation is the acquisition of the angles $(\hat{\phi}, \hat{\theta})$, which express the horizontal and vertical angle of the gaze.

3.4 Software for network training and evaluation

For the training of the proposed network, we used the PyTorch software [22], which is one of the fastest-growing deep learning platforms today. The reason we specifically chose PyTorch is the fact that it combines the simplicity offered by the Keras platform [23], while at the same time providing us with the full range of TensorFlow options [24].

Before using PyTorch, we attempted to build the network using Keras [23]. However, it turned out to be very difficult to implement the proposed network with Keras, as several complex procedures were required to construct a network with multiple input types (images and head-pose diagrams). This problem was solved by using PyTorch.

Finally, the training of the network was carried out using Google Colab [1], which provides free access to considerable computational power.

[1] https://colab.research.google.com/

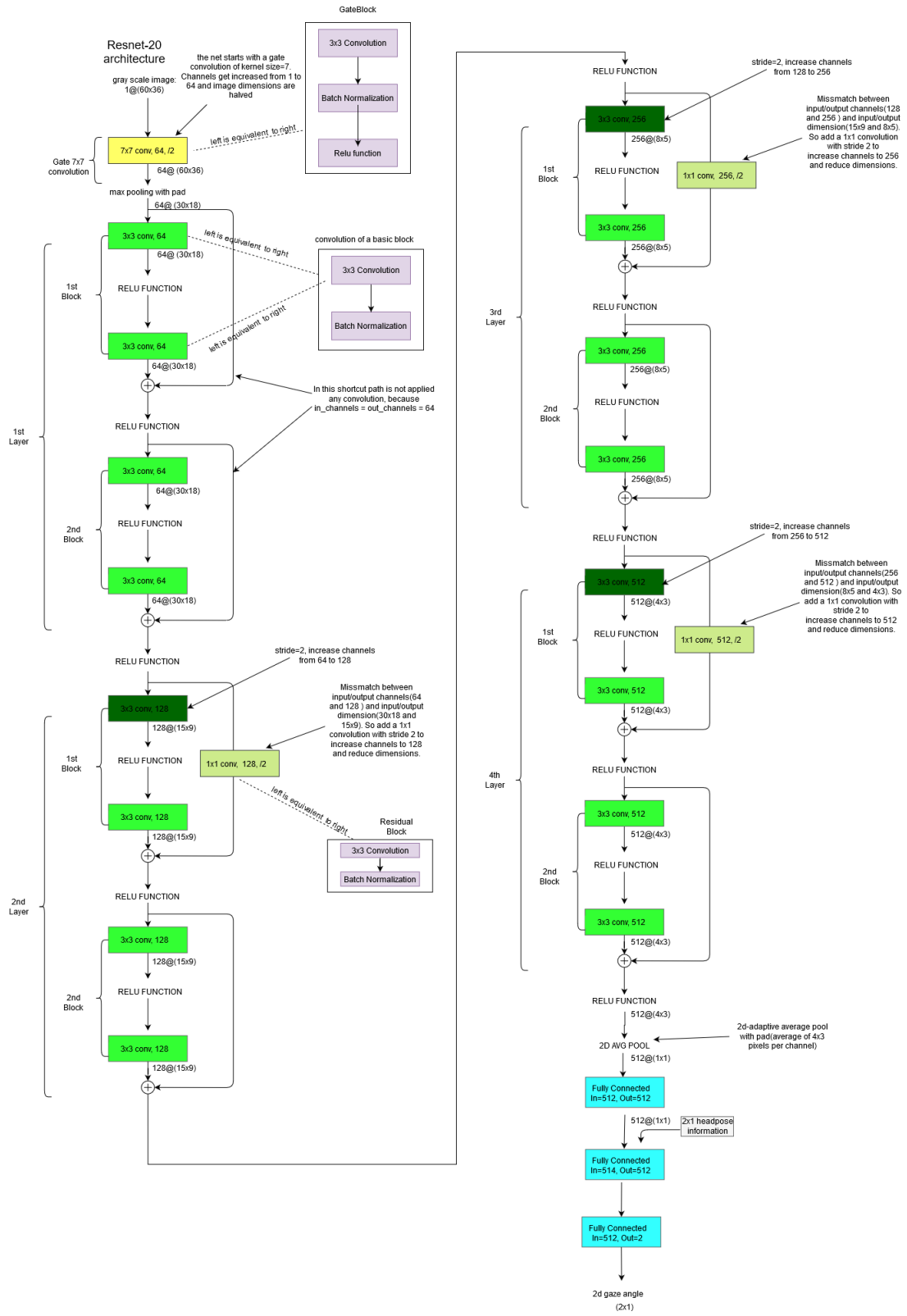

Figure 3.2: On the left, the first 9 convolutional layers of the ResNet-20 network are shown. On the right, the last 11 convolutional layers of the ResNet-20 network are shown.

.

Chapter 4

Human–Computer Interaction through Gaze Direction

In this chapter, we will describe in detail a *real-time* application, where we will make use of the gaze direction obtained from chapter 3.3. Specifically, we will describe a method regarding how the computer can recognize where exactly on the screen a user is looking. For the implementation of the application, we will need the user's head position relative to the screen, the gaze direction, the dimensions of the screen in millimeters, as well as the resolution of the screen in pixels. Finally, we will present the results and verify that the proposed algorithm of chapter 3 can indeed be used under real conditions. Therefore, we consider this application as an additional way to verify the accuracy of the proposed algorithm.

4.1 Schematic representation of the application

At this point, we recall the $(\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\theta}})$, the two-dimensional vector of the gaze direction calculated in chapter 3.3, as well as the vector $\boldsymbol{t}$, that is, the vector of the head position in three-dimensional space relative to the camera, which we calculated in section 2.2.3. By applying trigonometry, we can calculate the point on the screen where the user is looking. This is shown in detail in figure 4.1.

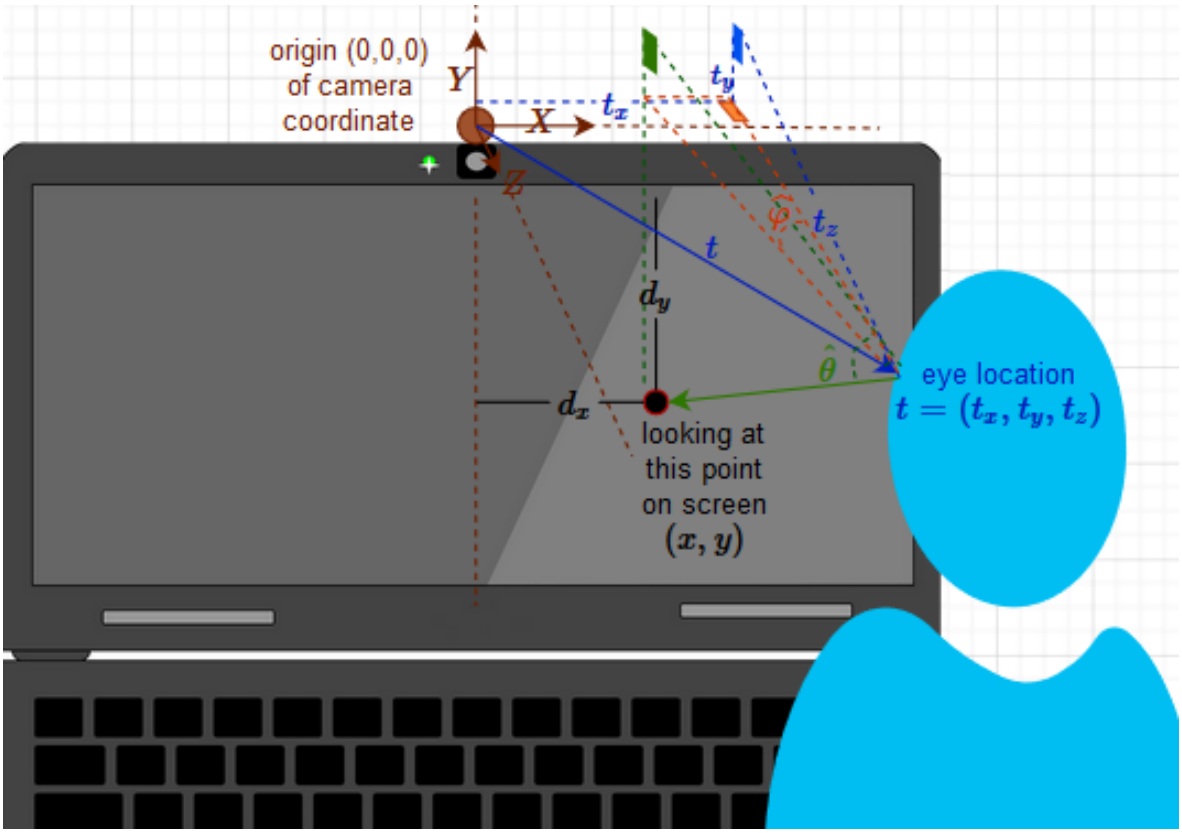

Figure 4.1: Example of detecting a point on the screen given the translation vector $\boldsymbol{t}$, the gaze angles $\hat{\boldsymbol{\phi}}$, $\hat{\boldsymbol{\theta}}$, and the desired point $\boldsymbol{(x, y)}$ on the screen.

The *gray* color of the image denotes the camera coordinate system. The *blue* vector indicates the position of the eye relative to this system. The *green* vector indicates the direction of the gaze. The *orange* angle $\hat{\boldsymbol{\phi}}$ indicates the horizontal angle of the gaze direction, and the *green* angle $\hat{\boldsymbol{\theta}}$ indicates the vertical angle of the gaze, while the intersecting lines that define these angles are also visible. The d_x denotes how far to the right or left the point $\boldsymbol{(x, y)}$ on the screen, where the user is looking, is located relative to the camera, while d_y denotes how far above or below the camera the user is looking.

4.2 Detection of a point of interest on the screen

Before we calculate the point $\boldsymbol{(x, y)}$ of the screen in pixels, we calculate the displacements d_x, d_y (in millimeters) using the tangent of $\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\theta}}$ and the vector $\boldsymbol{t}$ in millimeters. Once we find (d_x, d_y), we convert them from millimeters to pixels. To achieve this, we approximate

the ratio of pixels per millimeter. Specifically, we measured with a ruler the horizontal and vertical dimension of the screen (in millimeters), while at the same time we find the resolution of the screen in pixels. If the screen has dimensions $m_x \times m_y$ millimeters and $p_x \times p_y$ pixels, then the ratios we want are

$$mmPerPixel_x = \frac{m_x}{p_x} \tag{4.1}$$

for the horizontal case and

$$mmPerPixel_y = \frac{m_y}{p_y} \tag{4.2}$$

for the vertical case. For example, if the dimensions are 3400mm×1900mm and the resolution is 1240×780 pixels, then the ratios of millimeters per pixel are approximately 0.3647 horizontally and 0.4105 vertically. Next, using formulas (4.3) and (4.4) we calculate the values of (d_x, d_y) in pixels using the coefficients computed from formulas (4.1) and (4.2).

$$d_x(pixels) = \frac{d_x(mm)}{mmPerPixel_x} \tag{4.3}$$

$$d_y(pixels) = \frac{d_y(mm)}{mmPerPixel_y} \tag{4.4}$$

Finally, after calculating the (d_x, d_y) in pixels, we look for the coordinates $\boldsymbol{(x, y)}$ that indicate the central pixel of the area of the screen the user is looking at. For the horizontal pixels we have

$$x = \frac{MAX\_WIDTH}{2} + d_x \tag{4.5}$$

since d_x denotes the displacement from the middle of the screen, while for the vertical pixels we have

$$y = d_y \tag{4.6}$$

since d_y denotes the vertical displacement from the height of the camera. In algorithms 1, 2 the calculations of $\boldsymbol{(x, y)}$ are shown in detail.

Algorithm 1 Calculation of the horizontal pixel x at which the user is looking

Outputs: x
Inputs: ϕ, t_x, t_z
Parameters: MID_HOR_PIXEL $= 620$ `// middle horizontal pixel`
Parameters: MM_PER_PIXEL $= 0.3647$ `// millimetres per pixel`
if $(t_x < 0) \wedge (\phi > 0)$ **then**
  $d_x \leftarrow -t_x + t_z \tan(\phi), \quad x \leftarrow \text{MID\_HOR\_PIXEL} - \dfrac{d_x}{\text{MM\_PER\_PIXEL}}$
else if $(t_x < 0) \wedge (\phi < 0)$ **then**
  $d_x \leftarrow t_z \tan(\phi), \quad x \leftarrow \text{MID\_HOR\_PIXEL} + \dfrac{t_x - d_x}{\text{MM\_PER\_PIXEL}}$
else if $(t_x > 0) \wedge (\phi > 0)$ **then**
  $d_x \leftarrow -t_z \tan(\phi), \quad x \leftarrow \text{MID\_HOR\_PIXEL} + \dfrac{t_x + d_x}{\text{MM\_PER\_PIXEL}}$
else if $(t_x > 0) \wedge (\phi < 0)$ **then**
  $d_x \leftarrow t_x - t_z \tan(\phi), \quad x \leftarrow \text{MID\_HOR\_PIXEL} + \dfrac{d_x}{\text{MM\_PER\_PIXEL}}$

Algorithm 2 Calculation of the vertical pixel y at which the user is looking

Outputs: y
Inputs: θ, t_y, t_z
Parameters: MM_PER_PIXEL $= 0.4105$ `// millimetres per pixel`
if $(t_y < 0) \wedge (\theta > 0)$ **then**
  $d_y \leftarrow t_z \tan(\theta) - t_y, \quad y \leftarrow \dfrac{d_y}{\text{MM\_PER\_PIXEL}}$
else if $(t_y < 0) \wedge (\theta < 0)$ **then**
  $d_y \leftarrow -t_z \tan(-\theta), \quad y \leftarrow \dfrac{d_y}{\text{MM\_PER\_PIXEL}}$
else if $(t_y > 0) \wedge (\theta > 0)$ **then**
  $d_y \leftarrow -t_y - t_z \tan(-\theta), \quad y \leftarrow \dfrac{d_y}{\text{MM\_PER\_PIXEL}}$
else if $(t_y > 0) \wedge (\theta < 0)$ **then**
  $d_y \leftarrow -t_y + t_z \tan(-\theta), \quad y \leftarrow \dfrac{d_y}{\text{MM\_PER\_PIXEL}}$

4.3 Software of our application

All the stages of the above application were implemented in the C++ programming language, except for the training of the proposed network which was done in Python. The C++ language was chosen because, apart from the accuracy of the measurements, a basic criterion was also the execution speed.

Initially, for the preprocessing of the data mentioned in Chapter 2, the tools OpenCV [18] and Dlib [13] were used. Subsequently, through the programming interface[1] of PyTorch

[1] https://pytorch.org/tutorials/advanced/cpp_export.html

(C++ API), we loaded the trained model into the application. Finally, we used the SDL[2] library to create a graphical environment that displays the point at which the user is looking on the screen.

The code of the above application is available on Github[3].

[2]https://www.libsdl.org/download-2.0.php
[3]https://github.com/caxelos/MPIIGaze/blob/hpc-code/webcam_face_pose_ex.cpp

Chapter 5

Experiments and Results

In this chapter we will evaluate the ResNet-20 algorithm that we analyzed in Chapter 3 and compare our proposed solution with the most recent approaches in the literature. We will follow the same evaluation procedure as in [9] and use the MPIIGaze [9] and UT Multiview [7] datasets. From MPIIGaze, 45,000 random samples are taken, 3,000 from each participant (15 participants). From UT Multiview, 64,000 random samples are taken, 1,280 from each participant (50 participants).

5.1 MPIIGaze and UT Multiview Datasets

Below we describe in detail the data collection procedure used for the training and evaluation stages. Note that the data of the two datasets are not the final data we used, as we applied the preprocessing described in section 2.3.

5.1.1 MPIIGaze

The MPIIGaze database [9] is based on data collected under non-laboratory conditions (in-the-wild). Unlike UT Multiview, the aim here is to collect data under different conditions of use. For this reason, data were collected over several months from 15 participants while they were using their personal laptops, in order to cover the different conditions of use (locations, lighting, time of day, etc.). The participants' heads were not in fixed position. The laptops were equipped with built-in webcams. In each image the position of the eyes

was manually annotated. In total, the MPIIGaze database contains more than 200,000 images, where the gaze directions cover the horizontal range of angles approximately [-60°, +60°] and the vertical range of angles approximately [-18°, +18°].

5.1.2 UT Multiview

The UT Multiview database [7] is based on data acquired in laboratory conditions. It contains data collected from 50 participants (35 men, 15 women), aged around 20 to 40 years. During the data collection, the participants' heads were fixed on a chin rest and they looked at a fixed distance of 60 centimeters from an LCD screen. Around the screen there were approximately 1.3 megapixel PointGrey Flea3 USB3.0 cameras, which were synchronized by a computer so as to capture simultaneous photographs (Images 1, 2 from [7]). Before taking the photographs, an intrinsic and extrinsic calibration of the cameras was performed for each participant in the UT Multiview (calibration of intrinsic and extrinsic parameters).

5.2 Evaluation Methods

We are going to perform 3 types of evaluations. In the first one, training is done on 64,000 random samples from the UT Multiview dataset [7] and evaluation on 45,000 random samples from the MPIIGaze dataset (cross-dataset evaluation). In the second, we use only the MPIIGaze dataset [9] and examine each participant separately from the network trained on data from the remaining 14 participants, with 3,000 samples per participant (leave-one-person-out). Finally, we perform a third evaluation similar to the second, except that this time the training data come only from a **single person** (person-specific validation). Only for this final scenario, we increase by 1000 the samples per person in order to balance the fact that we have less training data (4,000 samples per person).

For evaluation, we compute the mean euclidean error of all test samples, as shown in equation (5.1):

$$mean\ test\ error = \frac{\sum_{i=1}^{N} \sqrt{(\hat{\phi}_{pred}^{(i)} - \hat{\phi}_{target}^{(i)})^2 + (\hat{\theta}_{pred}^{(i)} - \hat{\theta}_{target}^{(i)})^2}}{N}, \tag{5.1}$$

hyperparameters	Cross-dataset Evaluation	Leave-one-person-out Evaluation	Person-specific Evaluation
Batch size	32	16	4
Epochs	40	25	30
Initial learning rate	0.0001	0.0001	0.0001
Every 10 epochs:	learning rate/10	learning rate/10	learning rate/10
Momentum:	0.9	0.9	0.9
Optimizer:	Nesterov	Nesterov	Nesterov

Table 5.1: Experimenting on various hyperparameters of the proposed ResNet-20.

methods	mean error	standard deviation
Regression Forests [7] (2014)	15.4°	4.5°
Mnist Net [8] (2015)	13.9°	2.5°
GazeNet [9] (2018)	**9.8°**	2.4°
ResNet-20 (ours)	12.45°	**2.21°**

Table 5.2: Training of models on UT Multiview and evaluation on MPIIGaze (cross-dataset evaluation). The values represent the mean error and the standard deviation among the 15 participants of MPIIGaze.

where $\hat{\boldsymbol{\phi}}_{\boldsymbol{pred}}$, $\hat{\boldsymbol{\theta}}_{\boldsymbol{pred}}$ are the predicted gaze angles and $\hat{\boldsymbol{\phi}}_{\boldsymbol{target}}$, $\hat{\boldsymbol{\theta}}_{\boldsymbol{target}}$ are the ground truth angles, and N is the total number of evaluation samples.

Finally, the training hyperparameters used in all three evaluation scenarios mentioned appear in Table 5.1. After measurements, in the case of the cross-dataset evaluation, the optimal value of the batch size is 32, because there is a large amount of training data in this case. In the case of the *leave-one-person-out* evaluation, the optimal batch size was computed to be 16, while in the case of the *person-specific* evaluation it was set to 4, since in this case the training data were much fewer compared to the other two evaluations. Finally, the number of epochs in each evaluation scenario differs. In all evaluation cases, training continued until no improvement in the mean test error was observed. The mean error is calculated from the last 5 epochs from the point where no improvement is observed.

methods	mean error	standard deviation
Regression Forests [7] (2014)	6.7°	0.7°
Mnist Net [8] (2015)	6.3°	0.6°
GazeNet [9] (2018)	**5.5°**	**0.5°**
ResNet-20 (ours)	6.95°	0.65°

Table 5.3: Training and evaluation of models on MPIIGaze, where each time we evaluate one person on a network trained with data from the remaining fourteen people (leave-one-person-out evaluation). The values represent the mean error and standard deviation across the 15 participants of MPIIGaze.

5.2.1 Training on UT Multiview and evaluation on MPIIGaze

First, we will train the model with 64,000 samples from UT Multiview and test the accuracy of the model with 45,000 samples from MPIIGaze (cross-dataset evaluation). This case represents the most difficult scenario with respect to prediction accuracy. Table 5.2 shows the mean error of all MPIIGaze participants, while Figure 5.1 shows the mean error of each participant in MPIIGaze. While the mean error of our algorithm (12.45°) is smaller than that of most methods in the literature, it is **21.3%** larger than that of GazeNet [9] (9.8°).

5.2.2 Evaluation per participant on MPIIGaze

Next, we will perform measurements on MPIIGaze using *leave-one-person-out* evaluation with 45,000 samples. At each step, one participant will be evaluated and the network will be trained with data from the rest 13 participants. The aggregate results are shown in Table 5.3. In this case our algorithm does not achieve the desired results, as the mean error is quite high (6.95°) compared to the corresponding errors of the algorithms in the literature.

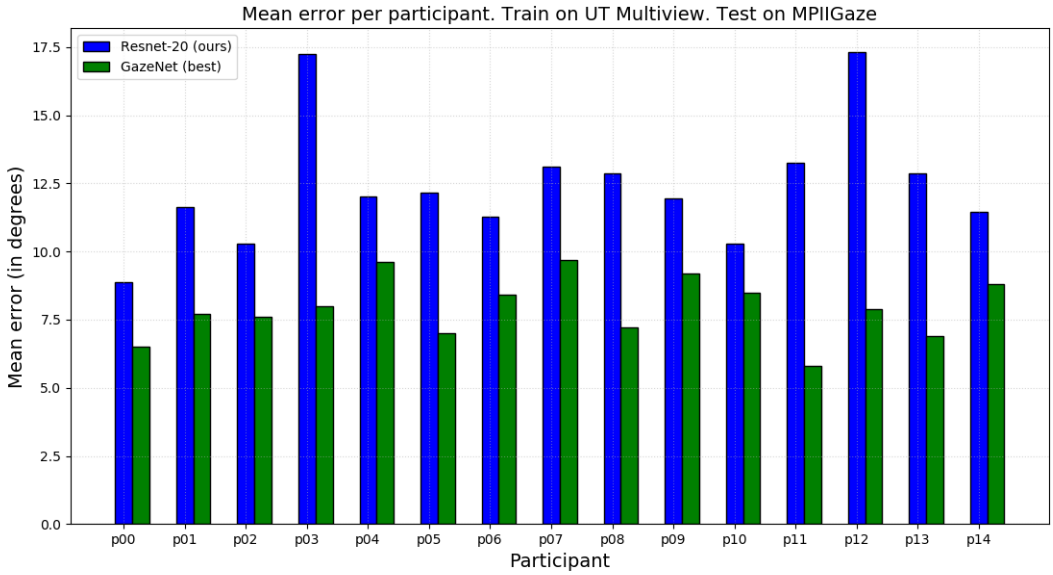

Figure 5.1: Training of models on UT Multiview and evaluation on MPIIGaze (cross-dataset evaluation). The values express the mean error of each of the 15 MPIIGaze participants.

5.2.3 Evaluation and training per participant on MPIIGaze

Finally, we perform measurements on MPIIGaze as in section 5.2.2, with the difference that in this case the training data come from the same person being evaluated and not from the remaining subjects (*person-specific*). Specifically, we use *5-fold cross validation*, with **80%** of the data serving as training data and **20%** as evaluation data. The only change compared to the process in figure 5.2 is that in this case the prediction takes place without difficulty, since both the training and the testing are done with data from the same person. As a result, the proposed method shows clearly better performance than the state-of-the-art method (GazeNet [9]) for each individual.

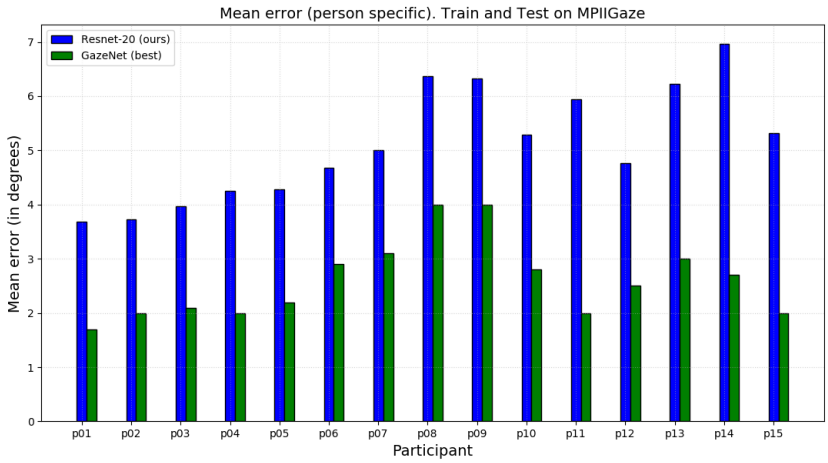

Figure 5.2: Εκπαίδευση και αξιολόγηση μοντέλων με δεδομένα από το ίδιο άτομο της βάσης MPIIGaze (person-specific evaluation). Οι τιμές εκφράζουν το μέσο σφάλμα καθενός από τους 15 συμμετέχοντες του MPIIGaze.

5.3 Conclusions

The initial conclusions were the observation of clearly better performance compared to the proposed architectures of the bibliography, due to the unique production of the ResNet network, as well as their ability to largely address the problem of vanishing gradients that causes the known cumulative errors (vanishing problem). Also, none of the three evaluation methods fell short of the best performance of the bibliography, while in the *cross-dataset* evaluation we had quite good performance.

The three evaluation methods in section 5.2 showed that the performance of the proposed network depends on the type of evaluation we follow. The proposed ResNet-20 method has better results in the first scenario (cross-dataset), where training and evaluation are performed on different data bases, while in the *leave-one-person-out* and *person-specific* evaluation scenarios, the performance was not as expected.

One of the possible reasons for the better performance in this case is the large amount of training data available in this scenario (64,000 samples) compared to the other two scenarios (42,000 samples in the leave-one-person-out scenario and 1,250 samples per person in the person-specific scenario). The architecture we used has greater depth compared to the other architectures in the literature. Therefore, it is very likely that more data are needed to achieve better results.

Chapter 6

Conclusion and Future Work

In our work we tested the ResNet-20 method, which resulted after experiments on various parameters of the ResNet network. Although the proposed network did not achieve the desired performance in the leave-one-person-out and person-specific evaluations, it had a remarkable performance in the cross-dataset evaluation, being the network with the second smallest mean error in the literature, namely 12.45°(see Table 5.2). Of course, there are still quite a few factors or techniques that we did not test within the framework of this thesis, which however could improve performance. Below we present some of our thoughts on this matter.

First, we could experiment with some different network architectures or some other machine learning algorithm. A quite worthwhile candidate seems to be the *ResNetXt network* [25], which borrows elements from both ResNet and VGG [26]. In this network, a main module can create parallel sub-modules which either add up or merge among themselves (cardinality increase). Then follows the network-in-neuron approach, where the classic convolution is replaced by a set of input-output layers inside each layer of the known bottleneck module of the ResNet networks [11], which is applied to every sub-module. Finally, the network uses filters that share the same intermediate dimensions, something that also happens in the VGG network.

Another change that appears quite interesting is the conversion of the problem from a *regression problem* to a *classification problem*. This can be done through a neural network that performs classification, and in the final layer does not simply call the softmax, which

discards quite a bit of information about the final prediction, but instead uses the *ordinal classification* technique [27], which treats a regression problem as a classification problem. Such a network is presented in [28]. For example, instead of the network's output being a single angle, we can consider the output as a vector of neighboring angle ranges. The goal of the network is to predict in which range of angles the target angle belongs.

Finally, regarding the input data, we can carry out additional changes, which very often yield better results. For example, the selection of some other preprocessing of the data than the one done in section 2.3 could be tested, so that the data are more evenly distributed.

Bibliography

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "ImageNet Large Scale Visual Recognition Challenge", *CoRR*, vol. abs/1409.0575, 2014. arXiv: 1409.0575. [Online]. Available: http://arxiv.org/abs/1409.0575.

[2] P. Majaranta and A. Bulling, "Eye Tracking and Eye-Based Human-Computer Interaction", in *Advances in Physiological Computing*, ser. Human-Computer Interaction Series, S. H. Fairclough and K. Gilleade, Eds., London: Springer, 2014, ch. 3, pp. 39–65, ISBN: 978-1-4471-6391-6. DOI: 10.1007/978-1-4471-6392-3_3.

[3] Y. Sugano, X. Zhang, and A. Bulling, "AggreGaze: Collective Estimation of Audience Attention on Public Displays", in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ser. UIST '16, Tokyo, Japan: Association for Computing Machinery, 2016, pp. 821–831, ISBN: 9781450341899. DOI: 10.1145/2984511.2984536. [Online]. Available: https://doi.org/10.1145/2984511.2984536.

[4] J. Kim, M. Stengel, A. Majercik, S. De Mello, D. Dunn, S. Laine, M. McGuire, and D. Luebke, "NVGaze: An Anatomically-Informed Dataset for Low-Latency, Near-Eye Gaze Estimation", in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19, Glasgow, Scotland UK: Association for Computing Machinery, 2019, ISBN: 9781450359702. DOI: 10.1145/3290605.3300780. [Online]. Available: https://doi.org/10.1145/3290605.3300780.

[5] K. A. Funes Mora and J. Odobez, "Person independent 3D gaze estimation from remote RGB-D cameras", in *2013 IEEE International Conference on Image Processing*, Sep. 2013, pp. 2787–2791. DOI: 10.1109/ICIP.2013.6738574.

[6] T. Schneider, B. Schauerte, and R. Stiefelhagen, "Manifold Alignment for Person Independent Appearance-Based Gaze Estimation", in *2014 22nd International*

Conference on Pattern Recognition, Aug. 2014, pp. 1167–1172. DOI: 10.1109/ICPR.2014.210.

[7] Y. Sugano, Y. Matsushita, and Y. Sato, "Learning-by-Synthesis for Appearance-Based 3D Gaze Estimation", in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 1821–1828. DOI: 10.1109/CVPR.2014.235.

[8] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling, "Appearance-based gaze estimation in the wild", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 4511–4520. DOI: 10.1109/CVPR.2015.7299081.

[9] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling, "MPIIGaze: Real-World Dataset and Deep Appearance-Based Gaze Estimation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 1, pp. 162–175, Jan. 2019, ISSN: 1939-3539. DOI: 10.1109/tpami.2017.2778103. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2017.2778103.

[10] E. T. Wong, S. Yean, Q. Hu, B. S. Lee, J. Liu, and R. Deepu, "Gaze Estimation Using Residual Neural Network", in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Mar. 2019, pp. 411–414. DOI: 10.1109/PERCOMW.2019.8730846.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[12] S. Mallick, *Head Pose Estimation using OpenCV and Dlib*, https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/.

[13] D. E. King, "Dlib-Ml: A Machine Learning Toolkit", *J. Mach. Learn. Res.*, vol. 10, pp. 1755–1758, Dec. 2009, ISSN: 1532-4435.

[14] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, Jun. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[15] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic, "300 Faces in-the-Wild Challenge: The First Facial Landmark Localization Challenge", in *2013 IEEE International Conference on Computer Vision Workshops*, Dec. 2013, pp. 397–403. DOI: 10.1109/ICCVW.2013.59.

[16] G. Tzimiropoulos, J. Alabort-i-Medina, S. P. Zafeiriou, and M. Pantic, "Active Orientation Models for Face Alignment In-the-Wild", *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 12, pp. 2024–2034, Dec. 2014, ISSN: 1556-6021. DOI: 10.1109/TIFS.2014.2361018.

[17] M. Lourakis, "A Brief Description of the Levenberg-Marquardt Algorithm Implemened by Levmar", *A Brief Description of the Levenberg-Marquardt Algorithm Implemented by Levmar*, vol. 4, Jan. 2005.

[18] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[19] X. Zhang, Y. Sugano, and A. Bulling, "Revisiting Data Normalization for Appearance-Based Gaze Estimation", in *Proceedings of the 2018 ACM Symposium on Eye Tracking Research Applications*, ser. ETRA '18, Warsaw, Poland: Association for Computing Machinery, 2018, ISBN: 9781450357067. DOI: 10.1145/3204493.3204548. [Online]. Available: https://doi.org/10.1145/3204493.3204548.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[21] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *CoRR*, vol. abs/1502.03167, 2015.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[23] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[24] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning", in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[25] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 5987–5995. DOI: 10.1109/CVPR.2017.634.

[26] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size", in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, Nov. 2015, pp. 730–734. DOI: 10.1109/ACPR.2015.7486599.

[27] E. Frank and M. Hall, "A Simple Approach to Ordinal Classification", vol. 2167, Aug. 2001, pp. 145–156. DOI: 10.1007/3-540-44795-4_13.

[28] J. Cheng, Z. Wang, and G. Pollastri, "A neural network approach to ordinal regression", in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, Jun. 2008, pp. 1279–1284. DOI: 10.1109/IJCNN.2008.4633963.