Universitat Rovira i Virgili

Daniel-Markus Koeckerbauer

Recognising people by means of clothing or other non-invasive factors

Final Master's Project, None-confidential Version

Directed by Dr. Francesc Serratosa

Computer Security Engineering and Artificial Intelligence

UNIVERSITAT
ROVIRA i VIRGILI

Tarragona

2024

Contents

1 Introduction

The use of artificial intelligence (AI) for person recognition has become well-established across various domains, ranging from unlocking mobile devices to criminal apprehension. Traditional methods predominantly rely on biometric factors. However, there is a growing interest in exploring non-invasive recognition techniques. This master's thesis addresses this emerging area by developing a person recognition system that leverages attributes such as clothing, height, and other non-invasive features. This system is intended to enhance the ski access control mechanisms of SKIDATA, the project's sponsor, with the ultimate goal of preventing unauthorized access.

2 Problem Statement

Ski pass purchasers are contractually obligated not to transfer their tickets to others. Enforcing this rule in practice poses significant challenges. Current methods involve photographing customers at the ski resort ticket office upon ticket purchase, with these images being periodically reviewed by resort staff. This manual and sporadic inspection is inherently unreliable, as it depends on human oversight, thereby allowing instances of fraud to go undetected.

In collaboration with SKIDATA, we propose a more efficient solution that integrates artificial intelligence for image verification. The developed AI system automatically compares the initial ticket purchase images with those captured at the access point. Crucially, the AI does not autonomously make final decisions. Instead, it flags discrepancies, prompting a resort employee to review the case and determine whether the ticket should be confiscated. The AI system was developed using TensorFlow, a Python-based framework for building neural networks.

3 Theory of Neural Networks

3.1 Background

Neural networks are computational models inspired by the biological structure of the brain. They utilize artificial neurons designed to mimic the behavior of natural neurons and their

connectivity. Neural networks are central to deep learning algorithms and represent a crucial area within the broader field of machine learning (ML) [Goodfellow et al., 2016].

A neural network comprises nodes, also known as neurons, which are organized into various layers. The initial layer is the input layer, followed by multiple hidden layers, and concluding with the output layer. Each neuron (or perceptron) is interconnected with other neurons, each possessing an assigned weight and a threshold value. When the output of a neuron exceeds its threshold, it activates and transmits data to the subsequent layer in the network [Bishop, 1995]. Neural networks learn and enhance their capabilities over time through training data.

Each neuron operates similarly to a linear regression model, consisting of input data, weights, a bias, and an output. The function can be expressed as:

$$\alpha = \Sigma_{i=1}^{n} w_i * x_i + bias \tag{1}$$

$$\begin{aligned} output = & \\ & = 1 \quad \text{if} \quad \alpha >= 0; \\ & = 0 \quad \text{if} \quad \alpha < 0; \end{aligned} \tag{2}$$

The weights are assigned after the input layer and define the significance of each input value, influencing the overall output. All inputs are multiplied by their corresponding weights and summed. Subsequently, an activation function processes the result. If the result surpasses a specified threshold, the neuron activates, passing the data to the next layer. This process is known as feedforward logic [Haykin, 2009].

Neural networks often employ sigmoid neurons, which produce outputs between 0 and 1. These networks function similarly to decision structures, transferring data from one neuron to another, thus minimizing the impact of individual neurons on the network's overall performance. An extended approach involves incorporating an activation function after the neuron, which modifies the output to optimize the entire network.

To illustrate this concept, consider an example (see Figure 1)

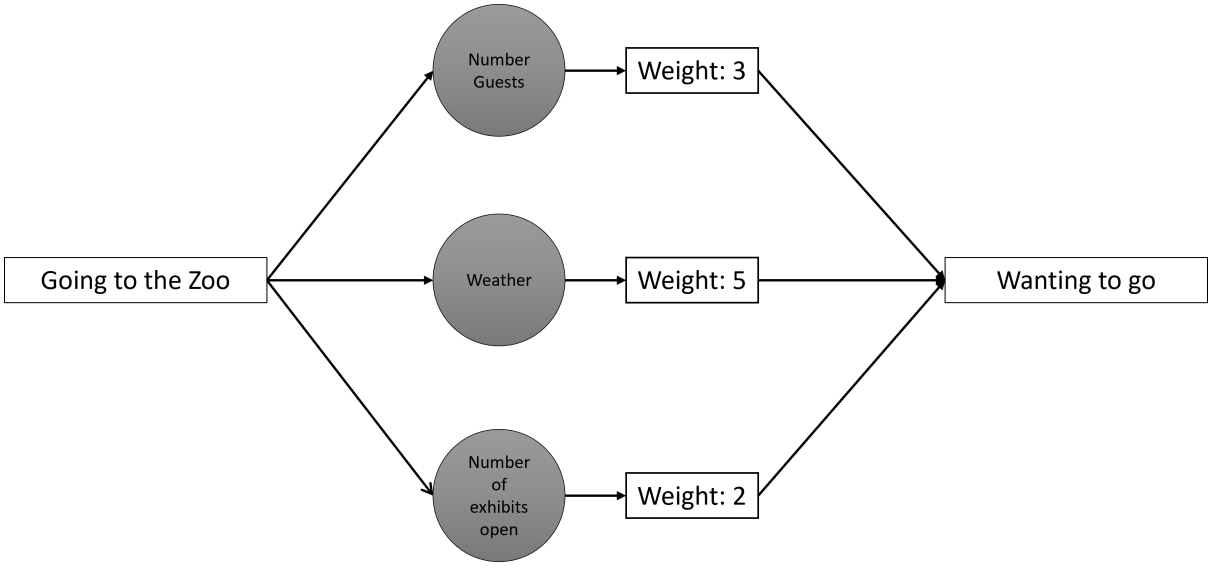

Figure 1: Example of a neural network

In this example, the scenario involves deciding whether to visit the zoo. The weighting is determined based on subjective preferences, with the weather being the most significant factor, hence assigned the highest weight. The neural network receives the following inputs:

- Number guests = 0, as there are many guests in the zoo.
- Weather = 1, as the sun is shining.
- Number of exhibits open = 1, as all enclosures are open.

A bias of -3 is applied, resulting in the following calculation:

$$y = (0 * 3) + (1 * 5) + (1 * 2) - 3 = 4 \tag{3}$$

In this simplified example, the neural network would recommend visiting the zoo, as the output value of 4 is greater than 0. In more complex networks with numerous factors, the decision-making process becomes significantly more intricate.

When training a model, ensuring accuracy is paramount, typically achieved through a cost or loss function. A commonly used loss function is the Mean Squared Error (MSE), which aims to minimize the cost function to suit the specific problem [Bishop, 1995]. As the model updates its weights and biases, it utilizes the cost function and reinforcement learning to self-correct. This adjustment process employs gradient descent, allowing the model to identify the optimal direction to reduce errors and minimize the cost function. Consequently, the model's parameters are iteratively adjusted, leading to a gradual convergence towards zero [Rumelhart et al., 1986, Brause, 2013].

Neural networks generally follow two types of logic. The predominant type is feedforward logic, where information flows unidirectionally (forward). Conversely, the backpropagation approach involves training the network in reverse, calculating the associated errors and their attribution for each neuron [Rumelhart et al., 1986].

Neural networks employ various activation functions for training. The most widely recognized model is the Rectified Linear Unit (ReLU), which has become the standard in recent years. The activation function, located within the neurons of the network, is applied to the weighted sums. This nonlinear activation function enables the neuron to understand nonlinear relationships, making it essential for comprehending more complex problems. It is through this function that neural networks gain the capability to model intricate processes. Without it, a neural network would merely represent a linear regression [Nair and Hinton, 2010, Lang, 2023].

However, there are other activation functions as well. Some of the most notable include:

- Sigmoid Function: This function maps input values to a range between 0 and 1.
- Tanh Function: This function maps input values to a range between -1 and 1.
- Softmax Function: Similar to the Sigmoid function, Softmax maps input values to a range between 0 and 1. However, it is particularly well-suited for representing probability distributions, making it ideal for use in the output layer.

The aforementioned activation functions are nonlinear, meaning their output curves do not maintain a proportional relationship with the input variable x. Simply put, an increase in x by 1 does not necessarily result in a corresponding increase in y. However, this nonlinearity introduces certain challenges:

- Vanishing Gradient/Saturation: For extremely large or small values, the function approaches being parallel to the x-axis, resulting in an almost zero gradient. This significantly impacts backpropagation, as the error feedback into the neural network is minimal. Consequently, the network experiences negligible learning due to minimal weight updates [Hochreiter, 1998].

- Non-Centered Output: Functions such as Sigmoid and Softmax have output values centered around 0.5, making them not zero-centered. This lack of zero centering can complicate the learning process, as the weights tend to maintain the same sign (+), potentially leading to instability in learning. A possible solution involves normalizing the data and transforming the input values to be centered around zero [LeCun et al., 2002].

3.2 TensorFlow

3.2.1 Technical Background

TensorFlow is an open-source Python framework designed for data flow-oriented programming. It is a library for developers to build and train neural networks. Originally developed by the Google Brain team for internal use at Google, it has been freely available under the Apache 2.0 open-source license since 2015 [Abadi et al., 2016].

TensorFlow is designed to be low-code, meaning that minimal coding is required to achieve results. It enables the use of GPUs for training, significantly reducing the learning time. While the primary focus of the library is on Python, it also supports development in JavaScript, albeit at a slower performance compared to Python [Smilkov et al., 2019]. TensorFlow supports both Deep Neural Networks and Feedforward Networks. Additionally, TensorFlow offers a playground website that simplifies the representation of Deep Neural Networks and clearly demonstrates the execution steps [Schuermann, 2020].

For implementation, the high-level neural network Application Programming Interface (API) Keras was used. Keras is a library that interfaces with TensorFlow, simplifying the design, training, and evaluation of deep learning models. It provides a user-friendly interface for creating and training neural networks with just a few lines of code [Chollet and others, 2018]. Keras also comes equipped with a variety of optimization algorithms, loss functions, and metrics that can be easily customized and used for model training. Keras can run on both CPUs and GPUs, further simplifying the training process [Chollet and others, 2018].

3.2.2 Theoretical Realization

Training a neural network involves several key considerations. Initially, the network generates weights autonomously, starting with random weight assignments. To facilitate this process, training data must be manually created and input into the neural network. The system processes these training datasets, comparing the expected output with the actual output. The algorithm then adjusts the weights to minimize the difference between the expected and actual outputs in subsequent training iterations, thereby progressively refining the network's performance.

This process can be illustrated by the example shown in Figure 2.

The algorithm analyzes the actual output and compares it with the expected output. The difference between these values is computed, multiplied by the input value, and added to the current weight. The resulting value represents the updated weight. This can be represented by the following equation:

$$y_i^{new} = w_i^{old} + \eta * (y - \hat{y}) * x_i \tag{4}$$

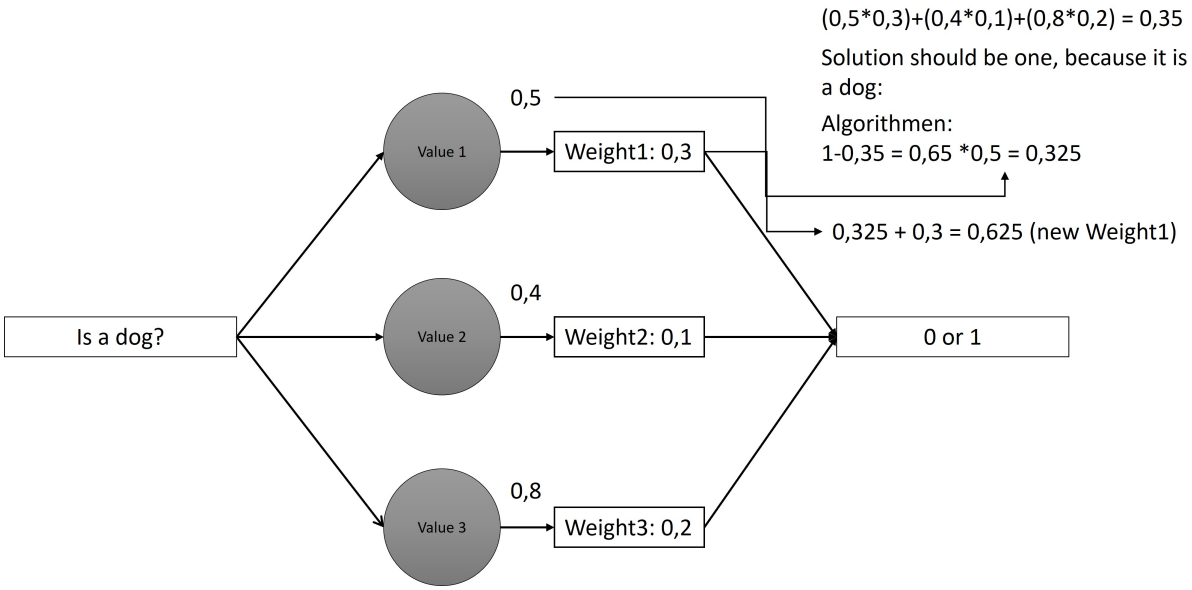

Figure 2: Example of the neural network algorithm

where Eta denotes the learning rate, which can be adjusted to control the speed of the learning process.

Care must be taken to avoid overfitting when incorporating training data. Overfitting occurs when the network becomes too specialized to the training data, impairing its ability to generalize. For instance, in our previous example (see Figure 2), insufficient diversity in the training data, such as a limited number of dog images, could lead the network to recognize only a specific dog. To mitigate this, the training data should include "malfeasance" data, enabling the network to identify errors.

Weights are not updated for each individual image but rather processed in batches, with each cycle termed an epoch. New training data are subsequently introduced to the network to verify correct recognition.

Given that the problem involves comparing two human beings, a "Siamese network" was deemed an appropriate solution.

3.2.3 Siamese network

Siamese networks represent one of the most advanced methodologies for image comparison, requiring significantly less data than conventional neural networks. A Siamese network comprises two or more identical sub-networks, with each sub-network having identical architecture, parameters, and weights. It is crucial that all values are mirrored across the networks [Rosebrock, 2020, Chopra et al., 2005, Bromley et al., 1993].

To address the problem of person recognition, a network capable of performing verification tasks is necessary. This approach requires significantly less training data compared to classification or regression tasks. A further simplification in Siamese networks is the omission of a 'None of the above' class, which handles images that do not match any of the predefined classes. In practice, this class often introduces errors. By focusing solely on whether the images are equal or unequal, this issue is effectively circumvented.

As illustrated above, many components of a neural network share similar structures, with significant differences primarily in the final layer. This final layer, known as the embedding layer, calculates the Euclidean distance between the outputs of the two sub-networks. The

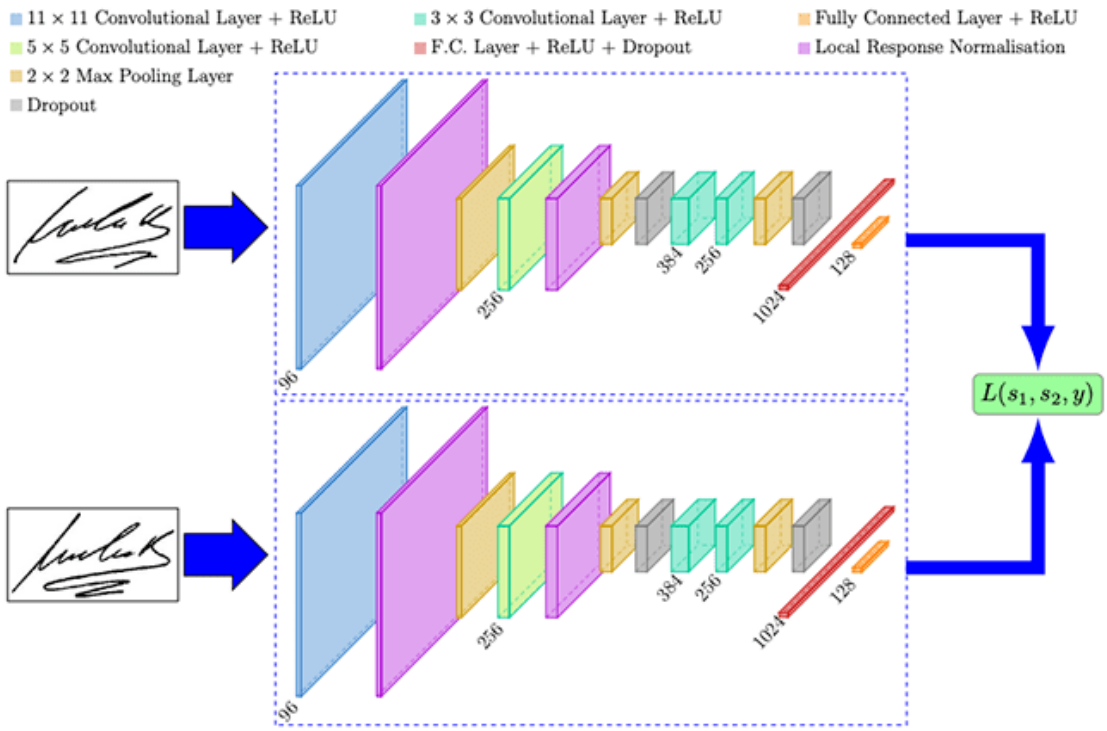

Figure 3: Visualization of the layers of a siamese network [Dey et al., 2017]

weights of the sub-networks are adjusted to ensure accurate decision-making.

The right section of Figure 3 calculates the 'loss' function, which combines the outputs of the sub-networks and verifies the accuracy of the Siamese network's decision. Various loss functions are available, with the most popular being:

- Binary cross-entropy [Hadsell et al., 2006]
- Triplet loss [Schroff et al., 2015]
- Contrastive loss [Hadsell et al., 2006]

Each loss function has its own advantages and disadvantages. However, the 'Binary cross-entropy' function is particularly suitable for our purposes, as it involves only two outcomes, corresponding to the two values our network needs to determine: whether the images are the same (1) or different (0).

The training methodology involves providing both Siamese networks with two inputs. To effectively train the network, both positive and negative pairs are required.

- Positive pair: Two images that belong to the same class, i.e. in the case of this master's thesis, two images of the same person [Chopra et al., 2005].
- Negative pair: Two images that belong to different classes, i.e. two different people [Koch et al., 2015].

We randomly select positive and negative pairs for training, providing the Siamese network with training data to learn similarities [Koch et al., 2015].

4 Theoretical Realization of a Dataset

Before developing the neural network, it is essential to analyze the existing data and devise an approach for utilizing this data to train the network. Our primary requirement for the neural network is its ability to recognize individuals based on their clothing in previously captured images.

4.1 Data

SKIDATA systems generate two types of images: address images and control images. These images are not stored directly in the database; instead, the database references the image indices. The images themselves are saved as user-defined media files and require conversion.

Address images are provided by customers and not generated by SKIDATA devices. Control images, on the other hand, are captured by SKIDATA systems at access barriers and are used for verification purposes. A ski resort employee can compare an address photo with a control photo to determine if they depict the same person.

Both sets of images are suitable for training the neural network. Given the requirements and available data, we developed the following approaches for training the AI:

- The AI compares the address photo with the control photo, mirroring the current manual solution.
- The AI uses the first captured image of the day, referred to as the base image (similar to the address image), and compares it with other captured images throughout the day.

The advantages and disadvantages of these two approaches are detailed below:

4.2 Version 1

Version 1, which involves the use of address and control images, was initially considered the superior approach due to insufficient information on how to acquire the images. The concept is as follows: address images are taken when the ticket is purchased. Consequently, the clothing remains unchanged, allowing it to be used for comparing individuals. This results in the base image having a much higher resolution than the control images, thereby increasing the likelihood of obtaining a high-quality image.

In practice, however, many of these images are outdated, taken months or even years prior. This diminishes the effectiveness of clothing as a recognition factor. Additionally, by anonymizing the images, many other potential recognition factors are lost. Moreover, the clothing in most images is rarely identical.

4.3 Version 2

The second version, which involves using control images exclusively, was developed only after analyzing the images. A significant disadvantage of this method is that many control images are of inferior quality, often blurred or unclear. To address this, an algorithm should be developed to identify the most suitable image and use it as a basis. However, this is not a primary concern for the current prototype. For now, the base images are manually selected, ensuring that the chosen image has the highest quality and clearly shows the individual.

It is crucial to have as many images of each person as possible. The greater the number of images with the same clothing, the more effectively the AI can learn from the dataset. This principle applies to both versions.

4.4 Structuring the Data Set

To effectively utilize the images, it was necessary to develop a structured dataset. This process involved addressing several key questions:

How can the images be organized to distinguish between different individuals? And how can an image be defined as a base image?

The following approach was adopted for structuring the dataset. All images were stored in a main folder named 'Dataset'. This folder contained subfolders named Person1, Person2, Person3, etc., with each subfolder holding the respective images of an individual. Within each subfolder, one image was designated as the base image and renamed to "Base". All other images were retained with their original names.

The resulting folder structure is illustrated below:

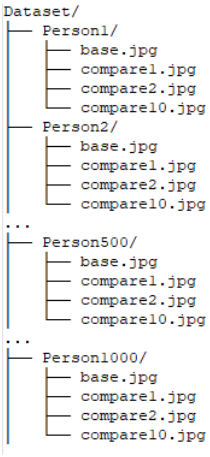

Figure 4: Structure of the data set

This structure facilitates easy expansion of the system to include more images or individuals. Initial tests were conducted with ten individuals and then progressively expanded.

4.5 Legal concerns

Once the rough data structure was defined, implementation could proceed. However, legal considerations are critical in such projects, necessitating consultation with a legal expert. The expert identified several key points:

Firstly, the data belongs not to the developer but to the customers. Consequently, most usage rights reside with these customers. To transfer these rights, a contract had to be drafted.

This contract established a legal framework for obtaining data that complies with both the new EU AI Act and the General Data Protection Regulation (GDPR). The GDPR is a regulation that protects the privacy and personal data of individuals within the European Union.

The initial challenges emerged here. Neither the customers nor the legal expert were willing to release unedited images. To address this, a compromise was proposed: to develop a face blurring algorithm that would automatically analyze and blur faces in all images. Additionally, the images had to be transmitted in encrypted form, ensuring that only the developer had access to them.

Another crucial contract point was the limited duration for which the images could be used (from January 1, 2024, to June 30, 2024). The intended use of the images was also clearly defined. With these preparatory steps completed, the technical development could begin.

5 Technical Realization of the Dataset

Various technical steps were involved in preparing the dataset to facilitate the training of the neural network. These steps included data extraction, face blurring, and sorting, each contributing to the overall goal of creating a robust and reliable dataset.

5.1 Extraction of Data

Data extraction was a crucial initial step where the relevant images were pulled from the database. The dataset comprised two types of images: address images provided by customers and control images captured by SKIDATA systems at access barriers. The extraction process involved selecting images that met specific criteria, such as being captured within a particular timeframe and having corresponding control images. This ensured that the dataset was relevant and sufficient for training the neural network.

5.2 Faceblurring

To comply with legal and ethical guidelines, particularly concerning data privacy, all images underwent a face-blurring process. A face-blurring algorithm was developed to anonymize the images by automatically detecting and blurring facial features. This step was vital for protecting individuals' identities and focusing the neural network on non-invasive recognition factors such as clothing and body structure. The blurring process involved multiple stages to ensure maximum accuracy, applying different threshold settings to capture all facial features effectively.

5.3 Sorting of the data

Once the images were extracted and anonymized, the next step was sorting them into a structured format. The images were organized into a main folder named 'Dataset,' with subfolders for each individual (Person1, Person2, etc.). Within each subfolder, one image was designated as the base image, and all other images retained their original names. This structure facilitated the easy expansion of the dataset and streamlined the process of training the neural network. The sorting process also included manually verifying and correcting any errors to ensure the dataset's accuracy and reliability.

5.4 Dataset Visualizer

To further enhance the dataset's organization and usability, a dataset visualizer was developed. This tool allowed for the visualization of the dataset's structure, making it easier to verify that images were correctly sorted and properly anonymized. The visualizer provided a graphical interface to review the images, ensuring that each subfolder contained the correct individual's images and that the base image was appropriately selected. This step was essential for maintaining the integrity of the dataset and ensuring that the neural network training would proceed smoothly without data-related issues.

6 Programming of Neural Network

The neural network training involved a comprehensive process of creating both a pre-made dataset and dynamically generating the dataset during the training phase. This dual approach ensured that the network was exposed to a diverse range of images and scenarios, enhancing its

ability to generalize and accurately recognize individuals based on non-invasive features such as clothing and body structure.

6.1 Creating of a Pre-made Dataset

The initial step in training the neural network was to create a pre-made dataset. This dataset served as the foundation for the training process, providing a structured set of images that the network could learn from. The images in the dataset were meticulously selected and processed to ensure they met the necessary quality standards. Each image was categorized and labeled appropriately, allowing the neural network to understand and learn the differences and similarities between various individuals.

6.2 Dynamically Generating the Dataset During Training

In addition to the pre-made dataset, the neural network also utilized dynamically generated data during the training phase. This approach involved creating new data on-the-fly as the network trained, ensuring a continuous supply of new images for the network to learn from. Dynamically generating the dataset helped prevent overfitting, as the network was constantly exposed to new and varied data, reinforcing its learning.

6.3 Configurations and Settings

The training process required various configurations to optimize the performance of the neural network. Key configurations included:

- **Batch Size:** The batch size, or the number of images processed in each iteration, was configured to optimize memory usage and training speed. A larger batch size can speed up training but requires more memory, while a smaller batch size is slower but uses less memory.
- **Number of Epochs:** The number of epochs, or complete passes through the entire dataset, was set to ensure the network had ample opportunity to learn from the data without overfitting. Each epoch provided the network with a chance to refine its understanding and improve its accuracy.
- **Paths for Output Folders:** The paths for output folders were defined to organize the results of the training process, including logs, and performance metrics. This organization was important for tracking progress and making adjustments as needed.
- **RAM Usage:** Efficient RAM usage was very important to ensure that the training process could handle large datasets and complex computations without running into memory issues. Configurations were adjusted to balance memory usage with training efficiency.

6.4 Detailed Technical Steps

Several detailed technical steps were carried out to create the dataset and train the model. These steps included:

- **Data Preprocessing:** The images were preprocessed to standardize their size, format, and quality. This preprocessing ensured that the images were consistent and suitable for training.

- **Model Architecture Design:** The architecture of the neural network was designed to optimize its performance for image pair classification. Layers were configured to extract relevant features and compare images effectively.
- **Training and Validation:** The network was trained using dynamically generated datasets. Validation data was used to monitor the network's performance and prevent overfitting by providing feedback on its accuracy and generalization capabilities. If overfitting applied, the training process was stopped.
- **Performance Evaluation:** The network's performance was evaluated using metrics such as accuracy, precision and recall. These metrics provided insights into the network's effectiveness and areas for improvement.
- **Model Saving and Deployment:** Once the network was trained and validated, the final model was saved and prepared for deployment. This step involved exporting the model to a format suitable for real-world use.

7 Hardware

Determining the optimal hardware configuration was a important component of the project, ensuring that the neural network could be trained efficiently and effectively. This process involved several stages, from initial testing on local hardware to final implementation on a virtual machine (VM) provided by SKIDATA. Each stage required careful consideration to maximize performance.

7.1 Initial Tests on Local Hardware

The first stage of hardware testing involved using a local laptop. This initial testing phase was essential for several reasons:

- **Accessibility:** The local laptop provided immediate access to hardware resources without the need for additional setup or configuration.
- **Baseline Performance:** Running initial tests on the local hardware helped establish a baseline performance level, providing insights into the computational requirements and potential bottlenecks of the training process.

7.1.1 Limitations of Local Hardware

However, the local laptop quickly revealed several limitations:

- **Insufficient RAM:** Training neural networks, especially with large datasets, requires significant memory resources. The local laptop's RAM capacity was insufficient to handle the dataset, leading to slow performance and frequent memory errors.
- **Limited Processing Power:** While the laptop's CPU could handle basic tasks, it lacked the computational power needed for efficient neural network training, which often relies on parallel processing capabilities provided by GPUs.

7.2 Transition to Virtual Machine (VM) on SKIDATA's Server

To overcome these limitations, the project transitioned to using a virtual machine (VM) on SKIDATA's server. This move provided several advantages:

- **Scalability:** VMs offer scalable resources, allowing for adjustments in CPU, RAM, and storage based on the project's needs. This flexibility ensured that the hardware could be scaled up as required.
- **Dedicated Resources:** Using a VM on SKIDATA's server provided dedicated resources for the project, eliminating competition with other processes and ensuring consistent performance.
- **Remote Access:** VMs can be accessed remotely, allowing the project team to work from different locations without compromising on performance.

7.2.1 Benefits of Using a VM

The transition to a VM provided significant benefits, addressing the key limitations encountered with the local hardware:

- **Increased RAM:** The VM was configured with additional RAM, sufficient to handle the large datasets involved in training the neural network. This increase in memory capacity eliminated the memory errors and slowdowns experienced on the local laptop.
- **Enhanced Processing Power:** The VM's configuration included powerful CPUs. This enhancement dramatically improved the training speed and efficiency.

7.3 Resource Allocation and Optimization

Optimal resource allocation was achieved through a process of trial and error, involving several iterations of testing and configuration adjustments:

- **Initial Configuration:** The VM was initially configured with a baseline level of resources, including a specific amount of RAM, CPU cores, and storage space.
- **Performance Monitoring:** During the training process, system performance was continuously monitored to identify bottlenecks and areas for improvement. Metrics such as CPU utilization, memory usage, and disk I/O were analyzed.
- **Incremental Adjustments:** Based on the performance data, incremental adjustments were made to the VM's configuration. This iterative process involved increasing RAM, adding more CPU cores, and optimizing storage allocation to balance performance and cost.
- **Final Configuration:** The final configuration of the VM was determined after several iterations, providing an balance of resources that ensured efficient and effective training of the neural network.

7.4 Specific Hardware Configuration

The final VM configuration included:

- **High RAM Capacity:** Sufficient RAM to handle large batches of image data during training, minimizing memory swapping and ensuring smooth performance.

- **Powerful CPUs and GPUs:** A combination of high-performance CPUs and dedicated GPUs to leverage parallel processing capabilities, significantly speeding up the training process.
- **Ample Storage:** Adequate storage space to accommodate the entire dataset.

8 Challenges

8.1 Quality of Pictures

One of the most significant challenges was the quality of the images. The hardware originally used was not designed for such purposes, resulting in images that did not meet the required standards. Of the more than 40,000 images extracted from the database, only a few were sharp and clearly showed the person. Additionally, unfavorable weather conditions, such as snowfall during image capture, further compromised the image quality. Many images were blurred, presenting a substantial obstacle.

8.2 Blurring Algorithm

Another challenge was posed by the blurring algorithm. Although it had been tested with preliminary data, the volume of poor-quality images was significantly underestimated. The algorithm incorrectly recognized faces in many images, either because there was no face present or due to poor lighting conditions that hindered face detection by the AI. In these cases, manual blurring of the faces was necessary. Moreover, the partial unreliability of the algorithm led to some images being completely or largely blurred, rendering them unusable.

8.3 Size of Training Set

The size of the training set also presented a major challenge. With the target set to include 1,000 people and approximately 8,000 images, sorting this data was an enormous task. Additionally, the large dataset posed significant demands on the hardware (see chapter 7).

8.4 Number of Epochs

Determining the appropriate number of epochs for training was problematic. The goal was to avoid excessively long training times while still achieving a prototype capable of recognizing most individuals. Initially, 300 epochs were selected for the first run with 1,000 people, but this proved insufficient. Consequently, a second training run with 1,000 epochs was necessary, leading to a significant amount of time wasted on retraining.

8.5 Hardware Performance

None of the hardware components used were specifically designed for AI training.

As a result, determining the optimal setup required trial and error. A system designed for AI training with multiple Nvidia graphics cards would have significantly reduced training times. However, such a system is costly and not currently available at SKIDATA. Therefore, 24 CPUs were utilized instead. Despite the relatively high number of CPUs, the training time for the dataset was nearly 21 days.

9 Results

The model was trained in several steps, with the number of images increasing at each step. This approach allowed for the determination of the minimum number of images needed to achieve usable results and facilitated hardware optimization with each step. To avoid unnecessary use of epochs, the algorithm described in chapters 6.1 and 6.2 was employed to stop the program if it was determined that no further improvements could be achieved after several epochs.

9.1 First Attempt: 10 Persons, 114 Pictures

The initial tests were conducted with a dataset comprising 10 people and 114 images, representing a relatively small training set. Nonetheless, this test provided some useful insights.

The *'image_pairs.npy'* file containing all the data was already 365 MB in size and required almost 8 GB of RAM. This indicated that the method used to generate the training data was impractical for very large training sets. Additionally, it was the first time a training set was taught. The program was terminated after 22 epochs as no further improvements were observed beyond this point. The result was unsatisfactory, with the training set only able to recognize

Figure 5: Plot after training for 114 pictures

the correct person in 60 percent of cases, even when evaluating values with a size of 0.5 as the same person. This outcome was expected given the small number of people. It also highlighted the necessity of using different hardware for larger training sets. The training set was processed on an HP laptop with an Intel Core i5 and 16 GB RAM, lacking a dedicated graphics card. The neural network required 4 hours and 22 minutes for these 22 epochs. Consequently, it was decided to use different hardware for subsequent steps (see chapter 7).

9.2 Second Attempt: 600 Persons, 4872 Pictures, 20 Epochs

The next test involved a dataset comprising 600 people with 4872 images and 20 epochs. Data sets created before the training were no longer functional with this configuration. Despite 256

GB RAM, the program stopped after a few seconds with the initial version, as more than 90 percent of the RAM was fully utilized.

Thus, the dataset had to be created dynamically. Although this process is more time-consuming, it was the only viable solution in this case. Twenty epochs were chosen to verify whether the code for dynamically creating a dataset was functional. The system required one day to complete the twenty epochs. As expected, given the small number of epochs, the result was not highly accurate in recognizing people. Nonetheless, it demonstrated that the system could handle larger datasets.

Figure 6: Plot after training for 4872 pictures

9.3 Third Attempt: 1000 Persons, 8349 Pictures, 300 Epochs

The next test involved the entire dataset intended for this master's thesis, consisting of 1000 people and 8349 images. This was the first attempt at a larger, longer training run, with 300 epochs chosen. However, this number proved insufficient. Although the program ran for more than 7 days, the plot indicated that the AI's full potential had not been reached. The training accuracy continued to rise and could likely increase significantly with more epochs. The training loss also decreased with each step. A longer training period would likely yield even better results.

Validation accuracy also improved in this instance, albeit more slowly than training accuracy. This is attributed to the random selection of validation data, resulting in more frequent outliers or poorer datasets than desired.

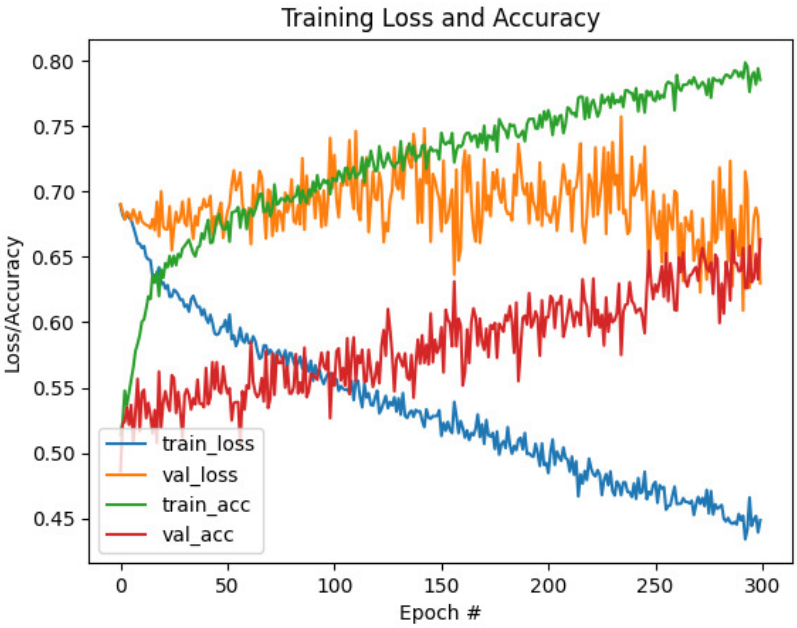

Figure 7: Plot after training for 8349 pictures

The following result was achieved with the training time and the specified data (see Figure 7).

```
Train loss: 0.4487, Train accuracy: 0.7853
Validation loss: 0.6297, Validation accuracy: 0.6633
```

Figure 8: Values after training for 300 epochs

With 300 epochs, a training accuracy of 78.53 percent was achieved, while the training loss was 44.87 percent. After analyzing this result, a critical decision had to be made: whether a training accuracy of 78.53 percent was sufficient or if it was better to restart the program with a significantly higher number of epochs. The decision was made to retrain the program with 1000 epochs to ensure maximum accuracy within the available time.

9.4 Fourth Attempt: 1000 Persons, 8349 Pictures, 1000 Epochs

In the final test, the AI was trained with a dataset comprising 1000 people and 1000 epochs. The program ran for 14 hours and 20 minutes over 18 days. It was observed that the learning effect slowed significantly during the last 200 epochs, with the recognition rate increasing by only 1.2 percent. This deceleration is clearly reflected in the plot. While further improvements in the recognition rate are possible, they would require a substantially longer training period. As in the previous test, the validation accuracy increased more slowly than the training accuracy. The following results were achieved with the specified training time and data (see Figure 9).

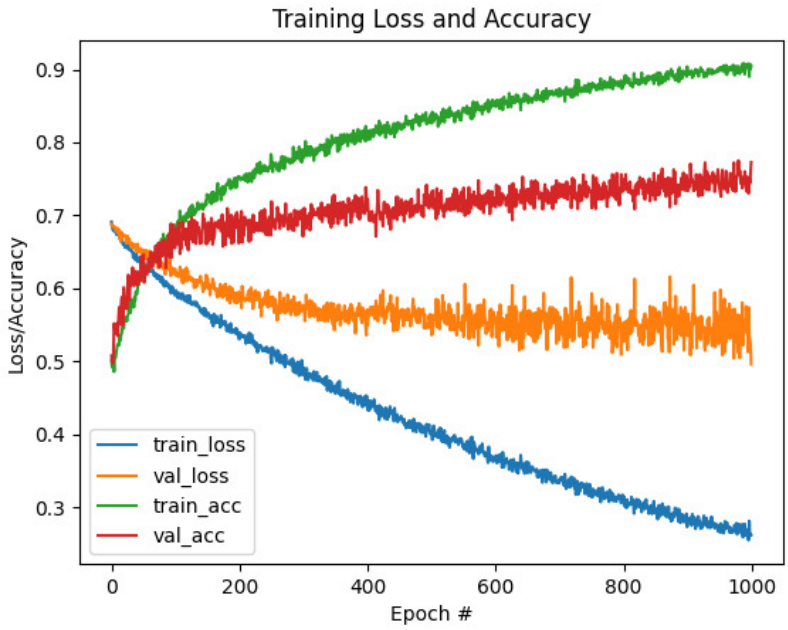

Figure 9: Plot after training for 8349 pictures

```
Train loss: 0.2618, Train accuracy: 0.9045
Validation loss: 0.4958, Validation accuracy: 0.7728
```

Figure 10: Values after training for 300 epochs

With 1000 epochs, a training accuracy of 90.45 percent was achieved, while the training losses were 26.18 percent. This indicates that the company's goal of achieving a recognition rate of over 90 percent was met.

Having completed the model training, the final step was the analysis. A Python program was developed to use unused images and compare them using the trained AI.

9.5 Program for Testing the Neural Network

To evaluate the neural network (a .keras file), a program was developed to apply the model and compare images outside the training and validation sets. The goal is to determine the model's accuracy in recognizing individuals and its speed in comparing images, critical for real-world

applications. This section outlines the program's development and functionality. Two scripts were created: *checkpicture.py* and *custom_layers.py*. These scripts utilize a neural network for image pair classification by processing images through a trained model and calculating their similarity. The code includes image preprocessing, model prediction, and result analysis. Below is an explanation of the functionality and rationale behind each part.

9.5.1 custom_layers.py

This script contains a function to calculate Euclidean distance between two vectors, which is important for determining the similarity between image embeddings in a Siamese network.

9.5.2 checkpicture.py

This script handles image preprocessing, model prediction, and result analysis.

It imports standard libraries for tasks like file operations, random generation, numerical operations, and image processing, along with TensorFlow and Keras for neural network operations. Additional libraries manage image processing and result visualization. The script includes methods to enhance image quality, such as improving contrast and correcting overexposure. It also loads a pre-trained model for feature extraction and processes images for input. Functions check and adjust image exposure, scale images to the required size, and prepare them for the model. The script also calculates similarity between images using a trained model and displays prediction results, including accuracy and prediction times. The main function coordinates these tasks, creating image pairs, making predictions, and summarizing overall performance through recorded results and visual plots.

10 Conclusion

The model was ultimately tested with 185 images that were not included in the training set.

10.1 Results

The model was evaluated using the code detailed in section 9.5. The results were as follows:

Out of 185 predictions, 132 were correct, yielding a success rate of 71.35 percent. This was significantly below the target success rate of 90 percent. However, it was noted that with optimal lighting conditions, the model achieved a 90 percent recognition rate. Conversely, poorer image quality reduced the likelihood of correct recognition.

The average performance time was 0.1549 seconds per image, indicating that the speed of recognition would not cause delays in the ticket presentation process.

The RAM consumption was also minimal, with a peak value of 5.4 GB (see Figure 12). This is a crucial consideration since the ski system runs numerous programs simultaneously, leading to limited available RAM.

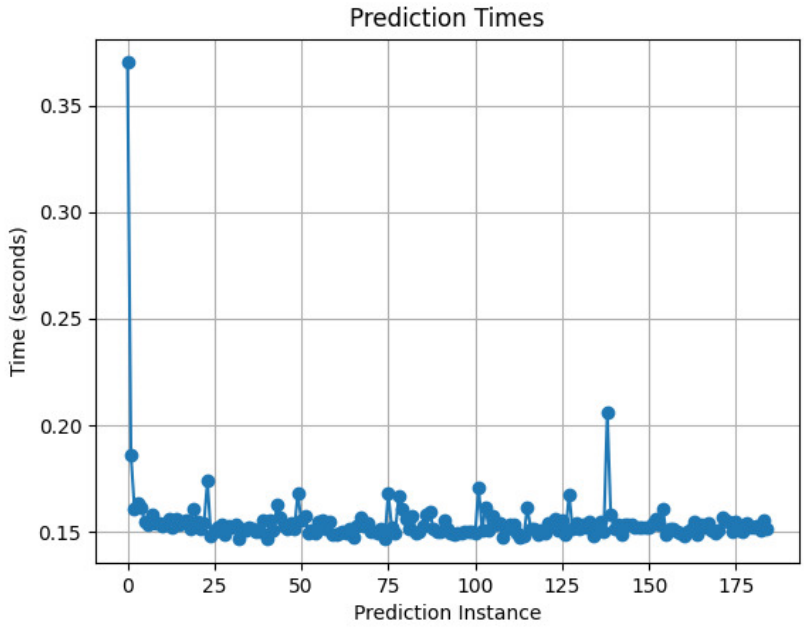

Figure 11: Performance of recognising people

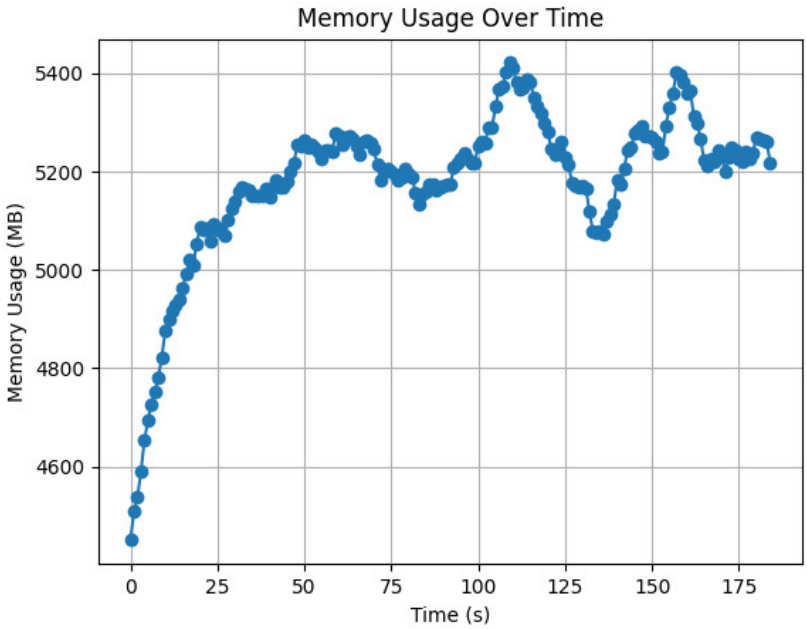

Figure 12: Memory usage while analysing the pictures

In conclusion, while the results are sufficient for a prototype, they are inadequate for active deployment. Nonetheless, several potential improvements could enhance the system and achieve better results in future versions. Some of these improvements are discussed in the next chapter (see chapter 11).

11 Future work/Improvements

Although satisfactory results were achieved with the available data and time frame, there remains considerable room for improvement.

11.1 Quality of images

The quality of the images presented a significant challenge during the AI development process. Only a few of the 100,000 images were of sufficient quality to meet the requirements. Additionally, the blurring algorithm further degraded the image quality. An effective solution to this problem would be to install higher-quality cameras in the SKIDATA devices. Numerous inexpensive cameras are available that produce superior images.

Improving the image capture algorithm in conjunction with the barriers would also be beneficial. Many images were blurred or captured the wrong person. Better synchronization of the capture times could help ensure that moving individuals are still in focus.

Furthermore, software solutions could enhance image quality. While some basic steps were integrated into the image enhancement algorithm, there was insufficient time to incorporate a more sophisticated AI model that not only brightens images but also adds pixels or information. Promising AI models could be employed to further improve the images.

Three examples of AI models that could be used:

1. **Learning Multi-Scale Photo Exposure Correction**

   Description: This model uses a coarse-to-fine deep learning approach to correct exposure errors in images. It processes the image across multiple layers of a Laplacian pyramid, correcting global color and detail information on each layer in turn. This method ensures that both underexposed and overexposed areas are effectively corrected, improving overall image quality.

   Repository Link: https://github.com/mahmoudnafifi/Exposure_Correction

2. **Exposure Correction Model by Karlsruhe Institute of Technology**

   Description: Developed by researchers at the Karlsruhe Institute of Technology, this model is designed for exposure correction of both underexposed and overexposed images. It utilizes a dataset created specifically for exposure correction and incorporates various deep learning techniques to improve image quality.

   Repository Link: https://github.com/yamand16/ExposureCorrection

3. **End-to-End Exposure Correction Model**

   Description: This model has an end-to-end architecture that addresses both underexposure and overexposure in images. It consists of an image encoder, residual blocks, and an image decoder, forming a comprehensive framework for exposure correction. The model has been shown to significantly improve image quality in terms of PSNR and SSIM metrics.

   Repository Link: https://deepai.org/publication/exposure-correction-model-to-enhance-image-quality

11.2 Blurring algorithm

There is also potential for improvement in the blurring algorithm. Although the current algorithm sufficed for the prototype, it frequently failed with low-quality images. An enhanced algorithm would reduce the time required to prepare the dataset and ensure that all faces are adequately blurred.

Examples of alternative blurring algorithms:

1. **Face Detection and Blurring with OpenCV**

   Description: This project uses Python and OpenCV to recognize and blur faces in images, videos, or live webcam streams. It employs a pre-trained Haar Cascade classifier for face detection and applies a Gaussian blur to the detected facial regions.

   Repository Link: https://github.com/vaiibs/Face-detection-and-blurring

2. **Deface: Video Anonymization by Face Detection**

   Description: Deface is a command line tool for automatically anonymizing faces in videos or photos. It recognizes faces using a neural network and offers various anonymization methods, including blurring, pixelation, and fixed boxes. It supports multiple backends such as OpenCV and ONNX.

   Repository Link: https://github.com/ORB-HD/deface

3. **Face Detection and Blurring Using YOLOv5**

   Description: This repository uses the YOLOv5 algorithm for face recognition and applies pixelation to blur the recognized faces. It is suitable for both image and video processing and provides a robust solution for anonymizing faces using state-of-the-art object recognition techniques.

   Repository Link: https://github.com/tprhat/face_recognition

11.3 Larger Dataset

Expanding the dataset, including the number of people and the number of images per person, could significantly enhance the AI's quality. Although the images are already available, this improvement primarily depends on time and resources. Additionally, the program could be developed to collect information and make improvements during customer checks, reducing the initial dataset development time. However, this approach poses the risk of the neural network training on unprocessed facial data if the blurring algorithm fails, which must be avoided at all costs.

11.4 Better hardware

As discussed in chapter 8.5, the training hardware can also be improved. The current setup involved a server with 24 CPUs, mainly due to the lack of a better system(see chapter 7). For a commercial application, using Nvidia graphics cards or similar hardware for training would be crucial. This could significantly reduce training time, allowing for a much larger dataset to be used. Consequently, the AI would be better prepared for borderline cases or difficult images (e.g., poor lighting, children, sledges), thereby significantly increasing the recognition rate's quality.

References

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., et al. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283.

[Bishop, 1995] Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

[Brause, 2013] Brause, R. (2013). *Neuronale Netze: Eine Einführung in die Neuroinformatik*. Springer-Verlag.

[Bromley et al., 1993] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6.

[Chollet and others, 2018] Chollet, F. and others (2018). Keras: The Python Deep Learning library. Astrophysics Source Code Library, record ascl:1806.022.

[Chopra et al., 2005] Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 539–546. IEEE.

[Dey et al., 2017] Dey, S., Dutta, A., Toledo, J. I., Ghosh, S. K., Lladós, J., and Pal, U. (2017). Signet: Convolutional siamese network for writer independent offline signature verification. *arXiv preprint arXiv:1707.02131*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[Hadsell et al., 2006] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE.

[Haykin, 2009] Haykin, S. (2009). *Neural networks and learning machines, 3/E*. Pearson Education India.

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

[Koch et al., 2015] Koch, G., Zemel, R., Salakhutdinov, R., et al. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille.

[Lang, 2023] Lang, N. (2023). Was ist die relu-funktion (rectified linear unit)? https://databasecamp.de/ki/relu.

[LeCun et al., 2002] LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (2002). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Rosebrock, 2020] Rosebrock, A. (2020). Building image pairs for siamese networks with python. *ttps://www. pyimagesearch. com/2020/11/23/building-image-pairs-for-siamese-networks-with-python/dated Nov*, 23.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[Schuermann, 2020] Schuermann, T. (2020). Tensorflow grundkurs 1: Neuronale netzwerke, komponenten, tensoren. https://www.linkedin.com/learning/tensorflow-grundkurs-1-neuronale-netzwerke-komponenten-tensoren.

[Smilkov et al., 2019] Smilkov, D., Thorat, N., Assogba, Y., Nicholson, C., Kreeger, N., Yu, P., Cai, S., Nielsen, et al. (2019). Tensorflow. js: Machine learning for the web and beyond. *Proceedings of Machine Learning and Systems*, 1:309–321.