USING COMPUTER VISION IN RETAIL ANALYTICS

MARCUS NORRGÅRD

STRATACACHE OY

Master of Science Thesis
Supervisors: Annamari Soini, Johan Lilius
Advisor: Niclas Jern
Software Engineering
Department of Information Technology
Åbo Akademi University
2020

ABSTRACT

This thesis comprises the creation of a computer vision-based analytics platform for making age and gender-based inference for retail analytics. Computer vision in retail spaces is an emerging field with interesting opportunities for research. The thesis utilizes modern technologies to create a machine learning pipeline for training two convolutional neural networks for classifying age and gender. Furthermore, this thesis examines deployment and inference computing of the trained neural network models on a Raspberry Pi single board computer. The results presented in this thesis demonstrate the feasibility of the created solution.

ACKNOWLEDGEMENTS

I would like to thank professor Johan Lilius and my supervisor Annamari Soini for their help and guidance during the writing of this thesis. I would also like to thank my advisor Niclas Jern for his support and my employer Stratacache OY for giving me the opportunity to make this project a reality.

Marcus Norrgård

Turku, April 30, 2020

Table of Contents

1 INTRODUCTION

The purpose of this thesis is to implement and design a proof of concept computer vision (CV) solution for gathering retail analytics insights for Stratacache OY. Further, the thesis tests and benchmarks how well the developed computer vision solution performs. The thesis will start by explaining core concepts and engineering technologies that are commonly used in computer vision, image recognition, and object detection.

1.1 Aim

The thesis focuses on the technical aspects of CV. First, it briefly introduces and compares so-called traditional computer vision techniques with modern state of the art technologies based on convolutional neural networks. It presents the technologies that are most popular and relevant today and accounts for the development of the system. Secondly, it examines the trade-off between hardware limitations, accuracy, and performance and presents the findings.

1.2 Stakeholders

The author of this thesis is a software engineer at Stratacache OY. This thesis documents work done for Stratacache OY, a company, which specializes in retail analytics. The thesis serves as a research project to evaluate the possible suitability of a computer vision-based analytics solution for Stratacache OY.

The company aims to give retailers a broader understanding of the behavior of their customers. Insights are produced by combining different data sources, such as Wi-Fi, Bluetooth, people counters, and point of sales data. The generated insight allows Stratacache's customers to optimize sales. New technologies that could help retailers streamline the experience for their customers are interesting to an analytics company like Stratacache OY. Recently the processing capabilities of small single-board computers have increased significantly. This development has opened the doors for more robust and affordable computer vision solutions.

1.3 Problem

Utilizing computer vision in a retail environment could function as an interesting data source for Stratacache OY. As an analytics company, Stratacache OY is interested in technologies that could enable insights from people flow, e.g., classifying faces of the people inside a store. A computer vision solution with face detection could allow for both demographical analytics in people flow and could even capture attention span, mood, or age in a retail environment. The goal of this thesis will be to create a computer vision-based solution that can infer useful results for age and gender gathered from a video feed.

Stratacache's customers are retailers, and thus the product is required to minimize any impact it has on the customer concerning space required or network bandwidth. The solution should preferably be both affordable and small in size. The less of a visible impact it makes, the better. Further, a small-sized device makes installation easier. Today, there are multiple small single-board computers available that can handle computer vision tasks. For this thesis, it was decided that the third generation of the Raspberry Pi single-board computer, together with the Raspberry Pi camera module, would function as a useful reference platform. Simultaneously, the Raspberry Pi's modest hardware specifications will work as a minimum hardware requirement for a final computer vision solution.

Two different problems need to be solved to obtain analytics data from the video stream. Firstly, a technique that finds the faces from each frame is required. Secondly, a way to make classifications on the faces that are found is needed. The design has to take into consideration the usually resource-heavy parts of computing classifications. Looking at the problem from a programming standpoint, the task boils down to being able to capture as many frames as possible from the camera video-feed and simultaneously find and classify each face in the frame. There must be a balance between the choice of techniques and the acquired frame rate. A more accurate detector and classifier will most likely require more of the hardware, which results in a lower frame rate. A low frame rate, in turn, results in missed faces, which means poor analytics. As resources will be scarce, we will need to find techniques that allow for lightweight face detection and analytical classification, without having to sacrifice too much accuracy or loosing too many frames.

A possible solution to this problem would be to use the computational power of a data center for the analytics part. The device could send each detected face for inference. There are some issues with this solution, apart from the ones already mentioned. A cloud-based system would require sufficient bandwidth to maintain an acceptable frame rate for the classification algorithms to work. This does not chime well with the requirement of trying to minimize the footprint on the customer's infrastructure, as it is essential to keep any bandwidth consumption as low as possible. Sending large amounts of image data over a customer's network would be both expensive and would put an unnecessary load on the system.

1.4 Relevance

Computer vision-based analytics is an emerging field and is interesting for an analytics company such as Stratacache. Many competing analytics companies provide computer vision-based products and services that can produce insight into the physical world. Staying up to date with the industry is naturally important.

Furthermore, the type of computations that are needed for making analytical classifications of image data have become increasingly more reachable, as multiple neural network specialized platforms and computing devices have become available. Therefore, it is relevant to Stratacache OY to research if the development of an in-house computer vision solution for retail analytics is viable.

1.5 Solution criteria

The criteria for the computer vision solution are simple. The device itself needs to be small in size. The small size facilitates the installation and overall management of the device. The computer vision solution is required to run on a device that has similar hardware specifications as the rest of the company's devices. The device should have a camera attached to it, and lastly, use the neural network model to make classifications locally. This means that the device has to be able to make classifications, called inference, of the age and gender of the faces found in the video feed.

1.6 Tools

The book series, Deep Learning For Python [1] [2] [3], by Adrian Rosebrock, has been useful for this project. The books contain tutorials and suggestions on how to solve many of the problems related to training and designing neural networks. Due to the experimental nature of training neural network models, many pitfalls could be avoided, thanks to the information in the books. Inspiration for both the computer vision device and the training pipeline was found in the guides and tutorials provided by the book series.

The created solution will be designed to run on the single-board computer Raspberry Pi 3B+ with an attached Raspberry Pi Camera Module V2. The software that is written for the implemented solution will be written in Python. The Python programming language is popular when it comes to working with scientific computing, such as neural networks, and it has a wide selection of libraries and frameworks related to CV.

The frameworks and libraries handle tasks related to the creation of a neural network, object detection, and parallelization. Object detection utilizes the open-source computer vision library [4], or OpenCV, one of the most popular computer vision libraries. OpenCV has existed since the late 1990s, and thanks to its popularity, it has an extensive set of bindings for the most popular programming languages, including Python. OpenCV is written in C/C++, making it highly optimized and efficient.

The creation of the neural network is done with the Keras [5] neural network library, which uses the TensorFlow [6] machine learning framework as a backend. Both are popular tools for training machine learning models.

Imutils [7] is an image processing library for OpenCV and Python. It is maintained by computer vision and machine learning researcher Adrian Rosebrock. Imutils contains methods for parallelizing I/O tasks related to reading video feeds from a camera.

In many parts of this project, the data is ordered in array form, which comes in handy when working with digital pictures. The NumPy [8] Python library is used for efficient numerical computing. It allows data to be stored in large array structures and contains convenient tools for manipulating them.

The training of the neural network produced will be performed on a graphical processing unit, GPU. The PC, which executes the training process, is fitted with a Nvidia GTX 980Ti GPU.

1.7 Ethical discussion

Later chapters and sections present techniques for performing age and gender prediction from a technical standpoint. However, it is equally important to analyze the impact of computer vision in a public space from a legal and ethical perspective. Face detection is a controversial subject, and this section will briefly discuss problem areas and limitations that a hypothetical real-world solution could encounter.

The goal of this thesis is to produce a solution for predicting and classifying both age and gender. Hence, it is crucial to clarify that age and gender classified metrics are considered to be solely indicative. Detecting and classifying people based on their looks is a difficult task, even with the latest technologies. Every person is created differently, and predicting the age or the gender of a person based purely on visual features will always be error-prone, not to mention that gender is not binary. Further, a person's appearance can vary due to random factors in the environment. Light conditions, weather, and the placement of the camera can all have a drastic impact on how a person would be classified from the camera's point of view. Not only does the environment introduce its own bias, but the limitations of the technologies in use also have an impact on the classifications. This is why it is important to remember that such a system cannot and should not be used with the intention to produce perfect age or gender classifications. Classifications will be purely indicative. Nevertheless, they still provide valuable insight into the retail world.

Legislation sets its own limitations to what is possible in regard to the problem this thesis is trying to solve. Stratacache OY has customers worldwide, and data protection laws vary from country to country. Much has been done regarding data practices within the European Union. The EU has arguably the strictest data protection laws in the world. These laws specify what is considered personal and biometric information. An image of a person's face is considered as biometrical data, and thus highly sensitive. The European Union gives all member states some room for interpretation, allowing them to introduce conditions to their data protection laws. However, the recently implemented general data

protection regulation [9], the GDPR, article 9, states that all biometric data requires the consent of the person whose information is collected. This makes it difficult, if not impossible, for any company to use computer vision analytics in a retail environment.

Camera-based technologies such as the one developed in this thesis are, understandably, met with caution by the general public. The situation does not improve when totalitarian regimes outside of the EU ignore any sense of morality, employing computer vision systems with Orwellian-like surveillance. Furthermore, there should be no doubt about the responsibilities that are set on the companies utilizing computer vision systems in public areas. That said, EU citizens can feel very safe when it comes to these kinds of situations.

Stratacache OY has been working with a wide range of analytics data, such as Wi-Fi-data, for a long time, and proper handling of sensitive data has always been a cornerstone for the company. In this lies the reason for developing this system in-house, instead of relying on a third-party service, to be able to verify and control correct and ethical handling of the data and the product itself. The developed face detection software would be expected to follow the same rigorous privacy pattern as the rest of Stratacache's products and would need to exercise equal levels of data anonymization. This means that under no circumstances would the system store or otherwise spread identifiable data, and the data that is collected would be transformed in such ways that it would only have a statistical value for the company's analytics platform. This way, we can ensure that the system cannot contain features that would allow it to be exploited for harmful intent, either by a third party or someone within the company. There are several ways that a system can ensure the security of the data that it collects, from encryption to physical locks on the device itself. However, the specifications set by the task of this thesis allow for a fairly basic approach that will ensure that the data is handled correctly.

The goal of the system is to be able to handle all of the detection onboard the device itself. This strategy will have an impact on the security of the system itself. Technically, the problem could be solved locally or remotely. A computer vision device could offload some, or all, of the classifying and detection work, the so-called inference part, by uploading the stream of images to a data center for further analysis. However, one could argue that a local image detection solution, ensuring that no image data ever leaves the device, is more secure compared to one that relies on the computational power of a

datacenter upstream. A cloud-based approach could, hypothetically, leave sensitive biometric data open to various attacks and increase the risk of data falling into the wrong hands. A local system that ensures that no image data is stored during inference is vital to guarantee that no sensitive data can be misused. Even under the unlikely scenario that the device itself would end up in the hands of a third party, it would be difficult to obtain any sensitive data.

The system developed in this thesis will never be able to understand any other metrics than the accuracies for the two classes specified in this thesis, age, and gender. All classifications are produced in the form of a two-dimensional array. The only produced metric is the numerical prediction probability in each element of the array. In conclusion, this means that none of the steps taken to produce a successful classification will leave identifiable image data on the device. Further, a classification can never be tied back to a person. Later chapters will discuss the structure of the outputted classes in-depth and define how classification is done.

1.8 Structure of the thesis

The structure of this thesis is the following. Chapter 2 consists of a presentation of terminology and gives a technical explanation to the background of core concepts. Chapter 3 presents technologies that are used in this thesis. Chapter 4 examines the steps required to produce the computer vision solution. Further, it presents the requirements for it. Chapter 5 describes the training process of the neural network models, age and gender, and presents the results acquired during training. Chapter 6 presents the results of running the neural network models on the computer vision device. Chapter 7 draws conclusions and discusses future work.

2 BACKGROUND AND TERMINOLOGY

2.1 Computer vision

Computer vision, CV, is a broad field and includes everything from acquiring, processing, and analyzing digital images to enable computers to gain a level of understanding of the visual world. Computer vision is widely used in automation processes such as manufacturing and logistics but is frequently found in everyday products such as cars or consumer electronics too.

A computer can build an understanding of the visual world with the help of digital cameras. The camera relays a constant feed of frames in the form of arrays of pixel intensities. From these frames or arrays, the pixel intensities can be combined, and together with object detection algorithms, the computer establishes a higher-level understanding of the image.

In this thesis, the algorithms for creating an understanding of the input frames are called detectors and neural networks. These will be explained in detail in later sections. Generally, the flow for these types of algorithms is to first find the object, or objects, in the frame and then react to the objects in one way or another. In manufacturing processes, computer vision can be used to sort out faulty products from a production line. In consumer electronics, computer vision can be used to set the autofocus target of a digital camera when it finds the face of a person in the viewfinder.

This thesis utilizes the OpenCV library thanks to many of its convenience functions for transforming and processing image data, both in the training phase and later when deployed on the edge device.

2.2 Digital image fundamentals

Computer vision techniques consume image data as input. Images are usually represented as multi-dimensional arrays to make them understandable for a computer. The smallest building block of a digital image is the pixel. A digital image consists of several pixels forming a grid-like structure. Programmatically this structure is used to translate the image into a multi-dimensional array, where each pixel is placed in its corresponding cell.

Colored images are expressed in the so-called RGB color space, RGB being an abbreviation for red, green, and blue. Each color is represented by a value ranging from 0 to 255, where the integer value dictates the intensity of each color, 255 for the most intense gradient and 0 for none. Together the three values form a triplet of integer numbers that can form any 24-bit color. Consequently, each pixel of an image can be represented in a three-dimensional coordinate system. Figure 1 visualizes how the colors of an RBG image can be split into each separate channel.

The size of an image is represented by its width and height, e.g., 1920 × 1080 pixels. An image in the RGB color space could, therefore, be described as a three-dimensional matrix of the size 1920 × 1080 × 3, where the third dimension consists of the RGB values of each pixel. The array coordinate system is used for transforming digital images. Grayscale images are represented as a single value ranging from 0 to 255.

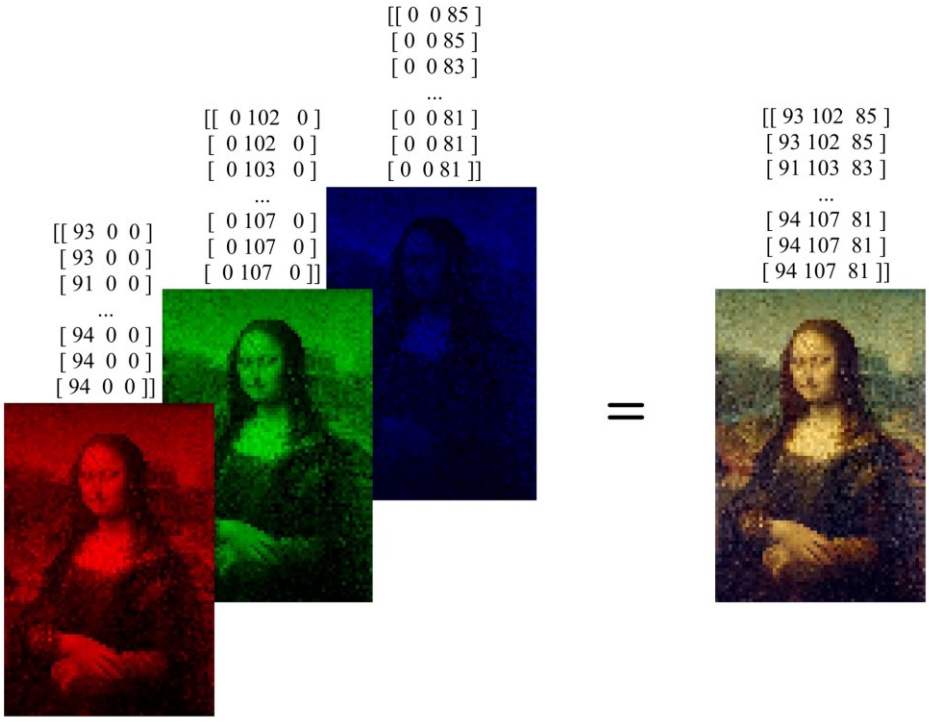

Figure 1. Each of the RGB color channels combined form the original image. Pixel intensities for the first row of pixels are represented in array form above each of the images. Original image [10]

2.3 Transforming digital images and convolutions

Image transformations utilize the multidimensional matrix structure of the digital image. The positional coordinates enable us to use functions that “slide” over the picture, applying different transformations on the values. Image transformation is, in its simplest form, mathematical calculations done on the values in the image matrix, element by element. Regular computer graphics functions such as sharpening, blurring, or rotating are all examples of image transformations. These image transformations are usually called convolutions or convolution filters.

A convolution is calculated within the area of a smaller matrix, a so-called kernel. Kernels are small in size, smaller than the whole image itself, and they slide over the image surface. The kernel applies a convolution to the area it covers by taking the pixel values

of the original image and applying the kernel convolution. Kernels can be used to blur, sharpen, or move the values of a digital image. Figure 2 visualizes the effect of applying the so-called Gauss blur, with a 3x3 kernel on an image. Gaussian blur produces a smoothing effect, which is often utilized in computer graphics to reduce the noise in images. If smoothing is applied multiple times, it results in an image with only the edges of the object visible. Edge detection kernels commonly use this property of the Gauss blur function. Fundamentally, a convolution is an element-wise multiplication of two matrices followed by a sum. Kernels performing convolutions are one of the building blocks of convolutional neural networks, which will be discussed in later sections.

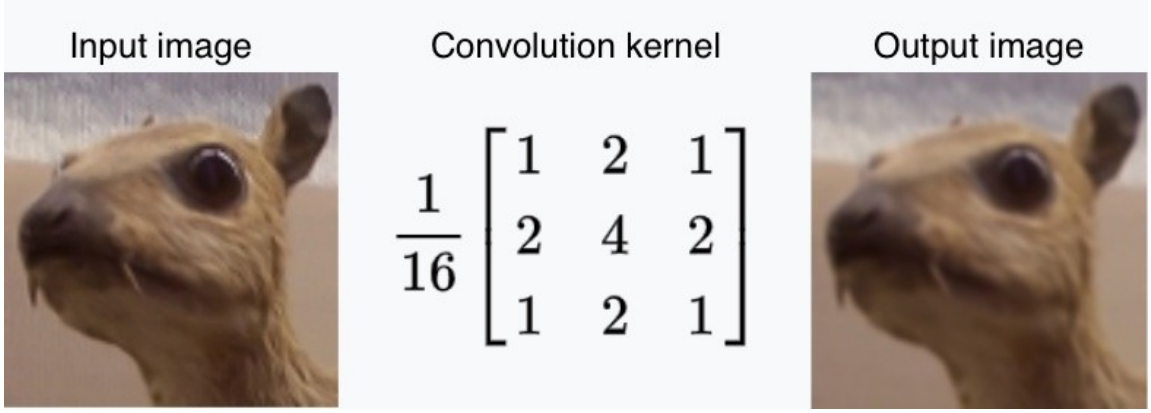

Figure 2. An example of the result of a convolutional kernel applying blur to an input image [11] and its resulting output [12].

2.4 Image classification and object detection

Image classification is the task of finding the correct label to an input image from a set of labels. There are different approaches to accomplish image classification. The classifiers in this thesis are called convolutional neural networks, known as CNNs.

The classification itself is a result of the input image being passed through the layers of the neural network. The network accepts the image as an array, and the network returns a class label, usually together with a metric such as a probability for the classified label. As an example, the output of a neural network that distinguishes between cats and dogs could return 90% Cat and 10% Dog for an input image of a Cat.

To be able to make valuable classifications, the region of interest needs to be found and extracted. In other words, the system needs to find where the subject that is to be classified is placed on the image. This is called object detection. Figure 3 contains an example of how an object detector has located each object in the frame and then classified them. The detector detects the x- and y-coordinates of the bounding box that contains the object. This area is then supplied to the classifier. Both classification and detection are topics that will be explored further later in the thesis.

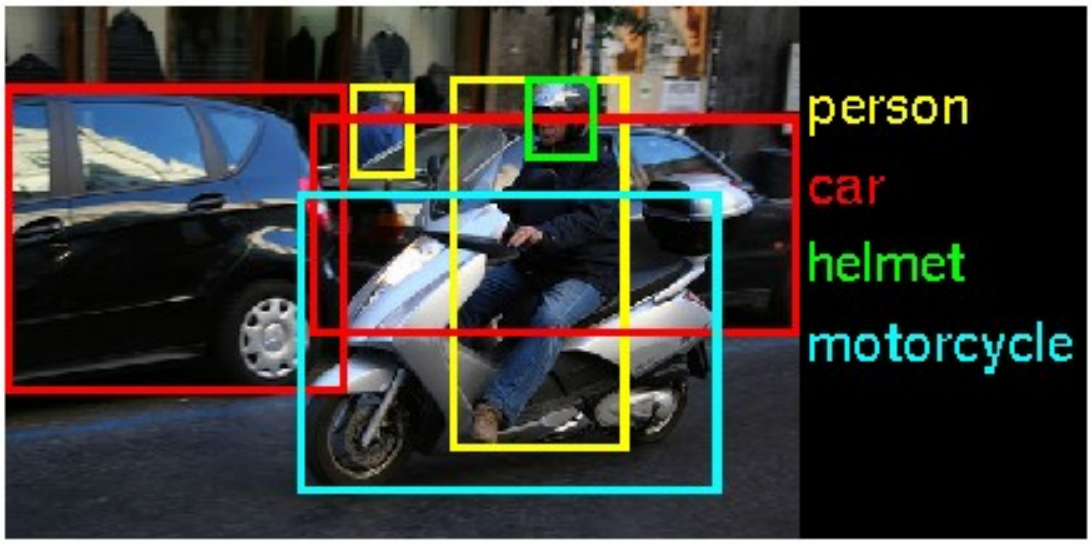

Figure 3. An image with objects localized and classified. Bounding boxes are drawn around each of the objects and their corresponding classes have been labeled. [13]

2.5 Machine learning

The previous section presented how objects are found in an image and how classification is done. To achieve the actual classification metrics the classifier algorithm needs knowledge of what it is looking for. The process of teaching the computer is called machine learning. This is the process of making the algorithm learn from a set of training data to enable it to make classifications automatically. This thesis focuses on machine learning that is specific for making classifications of image data.

Manually trying to teach an algorithm to classify an input image would be an enormous, if not an impossible, task. Machine learning is not a program with hundreds of if and else

statements. Previously it was mentioned that convolutional filters could be used as building blocks for making object classifications. Instead of defining which convolutional filters are needed in order to make a successful classification, a machine learning system should be able to learn these automatically. The learning process of machine learning is called training. During training, the system is presented with a set of training data, and it recognizes differences between the inputs by learning which convolutional filters produce the correct classifications. Training and how to achieve good results during machine learning are subjects that will be discussed in later chapters.

This is where a so-called machine learning model is useful. The problem can be reduced into a set of parameters that will be learned through training. Together these parameters form a function that can be used to classify any input data. A machine learning model makes the system more versatile, and since it only requires a few parameters, the size of the model will be greatly reduced.

2.6 Parameterized learning

The goal of parameterized machine learning is to abstract the method of finding the correct output for an input into a machine learning model. Parameterization is usually split into four parts: the training data, the scoring function, weights and biases, and the loss function. These four parts, put together, will make it possible to create the model and to utilize it for classification.

There are many benefits of parameterized models. One could naively approach a classification problem by comparing an input image to every image in a dataset containing the possible classes. This way, one could be able to find which class resembles the input image the most. However, the class dataset, with possibly thousands of example images, would have to be stored and quickly accessible at all times. A parameterized model will try to find a function that can solve the classification problem during training. This reduces the amount of information that is required and makes packaging and reusing the model quick and convenient. The following sections present each of the four parts.

2.6.1 Training data

Training data could be considered the most fundamental building block when training a machine learning model. The quality of the training data impacts the outcome of the trained model. There is no clear definition of how big a dataset should be for an optimal result. A dataset containing training data can contain thousands or even tens of thousands of images for every class label. As a rule of thumb, the more data, the better. Finding or creating a good dataset that solves the problem that one is trying to solve can be half the battle.

Data consists of a collection of images together with their class labels. The model requires training data of each class to be able to learn. During training, each image from the dataset will be fed to the network together with its corresponding label. The trained model learns the characteristic features of the images in the dataset. This means that a model that is trained on images of cars from the front will not be able to make sensible classifications on images of cars from any other angle. It is important to realize that the data itself drives how the classifier can be used in the end. Further, this is why it is essential to thoroughly verify what type of images a training dataset contains before starting the training of a model.

Labeling a dataset can be a tedious task, but fortunately, many organizations and educational institutes make labeled datasets available online. Datasets can be found for most problems, but one should keep in mind that these datasets may have licenses that limit how a model trained on them can be used. Building a custom dataset and labeling it takes time. Many search engines have image search functionality that could help with the building of a custom dataset, but the subject of building datasets is outside of the scope of this thesis.

One data point, or one image, from the testing dataset, is called a feature vector and is usually denoted as x_i. In this thesis, the data consists of a dataset that is popular for age and gender recognition. The dataset is called Adience [14], and consists of over 20000 images split into two classes for gender and eight classes for age. The structure of the Adience dataset is presented in chapter 3.

2.6.2 Scoring function

The scoring function outputs class labels from input data, creating predictions from the inputs. For the sake of simplicity, this section presents the concepts of the scoring function from the viewpoint of a linear classification problem. More advanced machine learning models have more complex scoring functions and nonlinear classification. The linear classifier is, however, still a fundamental building block for most machine learning algorithms. The goal of the linear classifier is to find a hyperplane that can be fitted to separate the two classes on an n-dimensional surface.

The scoring function of a linear classifier consists of three parts, W representing the weights matrix, b the bias vector, and x_i representing a feature vector, or an input image. The feature vector x_i has the dimensionality of its dimensions multiplied together into one column vector, e.g. $32 * 32 * 3 = 3072$.

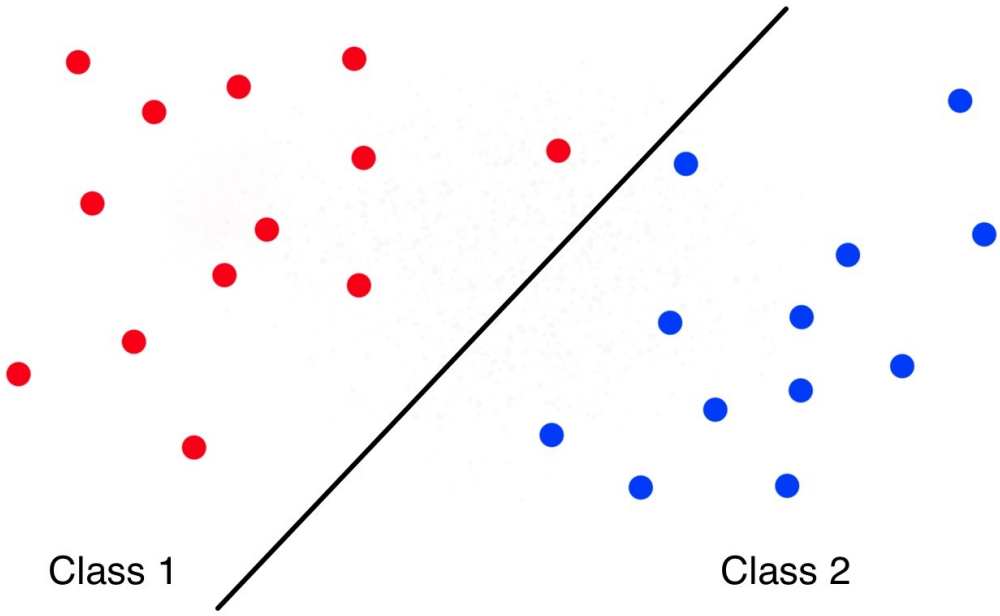

Figure 4. Linear classification of a problem with two classes

2.6.3 Weights and bias

The weights matrix W contains the parameters of the model. The weights matrix, together with the bias vector, can be altered and the task is to find the optimal values, such that the model achieves the best accuracy. The weights matrix has the shape of the dimensionality times the number of classes. The bias vector, b, is the last element of the scoring function. The function of the bias vector is to serve as a way to steer classification without changing the weights matrix. The size of the bias vector is the number of classes times one. Predictions are made by computing the dot product of the Weights W and the feature vector x_i together and adding the bias b.

$$f(x_i, W, b) = Wx_i + b.$$

Figure 5. The scoring function for a linear classifier

2.6.4 Loss functions

To achieve learning, the scoring function has to find the most suitable hyperplane separating the classes. This requires optimization, which steers training towards higher accuracies. A loss function is used to calculate the loss of the scoring function. The loss function measures how well the model is performing. The lower the calculated loss, the better the model is at predicting class labels from an input. The goal of the loss function is to tweak the weights and biases according to the scoring functions so that the optimal loss can be reached. This is achieved by trying to minimize the loss function, thus achieving the best score possible.

2.7 Optimization

Optimization methods are used to find the optimal parameters, weights, and biases for the scoring function. An optimizer works over a loss landscape. The loss landscape is an n-dimensional surface with a maximum and minimum. The optimal accuracy is found at the global minimum.

One of the most popular optimization algorithms for convolutional neural networks is based on gradient descent. The idea behind gradient descent is to evaluate the parameters in the scoring function and compute the loss. When the loss is obtained, the algorithm takes a step in a direction that minimizes the loss, thus taking a step along the slope of the loss landscape towards a local minimum. The step size is specified by the user and is called the learning rate. When a better value is found, the algorithm updates the weights and bias parameters. This algorithm is called stochastic gradient decent.

The values for the weight parameters are usually randomly initialized before the start of the training. Random initialization of starting parameters can help to overcome the issue of getting stuck in local minimums. The learning rate controls the size of the step taken by the optimizer.

Consider the figure below, where a robot is randomly placed somewhere on the loss landscape. It computes the gradient of the plane and takes a step towards the bottom of the minimum. The gradient of the optimization surface is the only thing that the robot is aware of in regard to its surroundings. If the learning rate is too large, the robot overshoots the minimum and will have problems ever finding the bottom of the minimum. If the learning rate is too low, the robot takes a long time to get down the minimum and risks getting stuck in a local minimum.

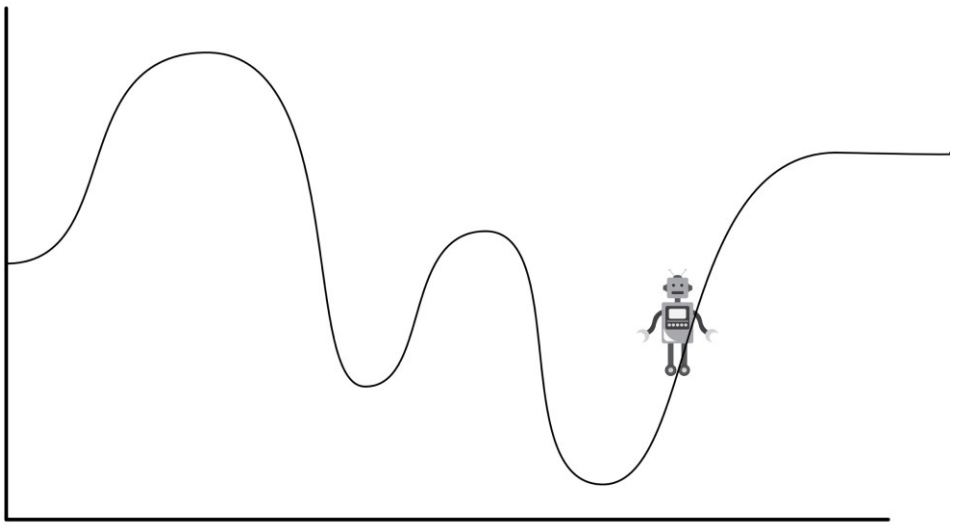

Figure 6. The loss landscape visualized. A robot has to take a step in the correct direction to reach the minimum.

Stochastic gradient descent may be one of the most popular optimizers, but it is not the only optimizer used for training neural networks. Other, more advanced methods utilize different techniques for handling the learning rate and may have different weight parameters. One of these optimizers is called Adam [15]. The Adam optimizer can help training models on challenging datasets. The optimizer is designed to work well on large, noisy datasets. A noisy dataset means that the loss landscape contains few and sparse minima, making them especially challenging to find. The Adam optimizer is outside of the scope of this thesis but presented here as it is suggested in the last chapter as a possible future improvement.

The machine learning model that is created in this thesis uses a stochastic gradient descent optimizer. The stochastic gradient decent optimizer is efficient since it only requires a subset of the data on the loss surface in order to update the weights and calculate the gradient. This approach makes training substantially faster.

2.8 Neural networks and deep learning

Neural networks have been mentioned earlier in this thesis. This section presents neural networks and, in particular, the type relevant for this thesis, convolutional neural networks.

Neural networks are inspired by and loosely based on the human nervous system. There are many similarities when it comes to naming and patterns. A neural network is built up of nodes that interact with each other via links, or so-called edges. Depending on its function, the neural network can consist of an arbitrary number of layers. From the inputs to the outputs, the network forms a treelike structure. Similar to how the human nervous system works, each node in a neural network performs a computation. This function is called an activation function, and it determines whether or not the node should activate and to which connected node the signal should be sent. When the node activates, the edge will carry the signal onward to the chosen node.

When designing neural networks, layers are specified to have different properties. For example, the number of edges carrying the signal to the next layer can be altered randomly. Learning is a complicated process, and the structure and the properties of the layers and nodes directly influence how well the network learns.

The signal in a neural network is carried in one direction from the input to the output. Each layer in the network architecture has a predefined role where every node in the layer performs a task depending on what the type of layer it is. Some of the most common layer types will be presented in the next section.

The figure below presents the neural network structure of the so-called Perceptron [16] developed in the 1960s. The network is very rudimentary by modern standards, but it is still an excellent example of the structure of a neural network. The network follows the same principles as networks today. It has a set of inputs followed by a layer of weights. These weights are connected to a net input function that gathers the signals and relays them to the activation function. The activation function contains a threshold that determines whether it activates or not. And lastly, there is an output node.

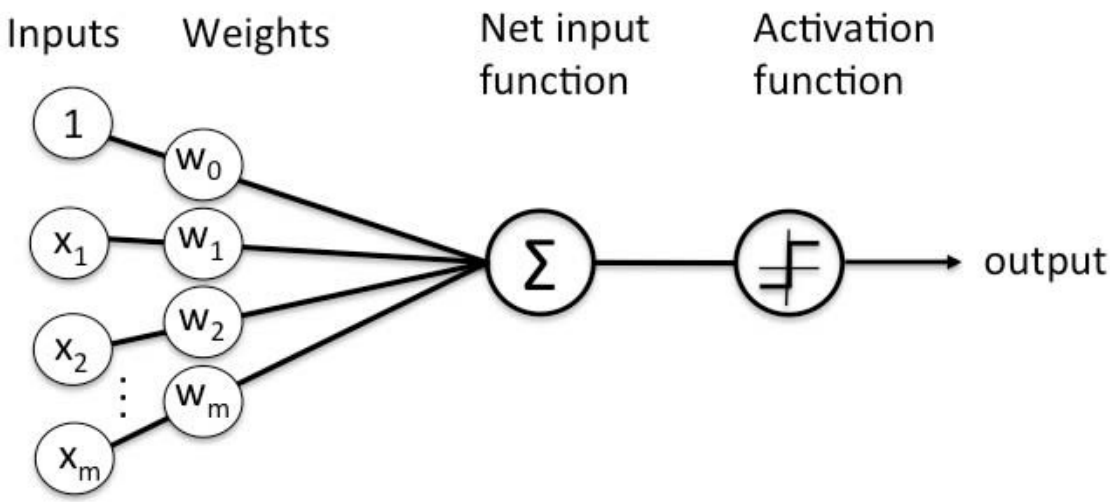

Figure 7. The Perceptron was one of the earliest implementations of a neural network. [17]

The term deep learning is commonly used when referring to the field of machine learning methods used to train neural networks. Even though the name suggests that, the depth of a neural network is not an essential feature. The structure of a neural network can vary based on its function, and usually, deeper networks can learn more features, but there are no guidelines that say to how deep a network has to be to learn more. More important is the combination of different layer types and in which order they are presented in the

network topology. Many neural network architectures can share common structural traits, especially the combinations of layers that have been proven beneficial when trying to achieve better performance of some sort. Different layer combinations can improve the efficiency of the network. How layers are structured can, for example, directly improve its feature detection capabilities and reduce the need for computational power. The next section gives an example of different layer types and how they enhance a network.

Deeper neural network architectures tend to be more time-consuming to train and require more computational power when calculating predictions. This means that the structure of a network should be designed depending on the intended use case. Complex and deep networks are suitable for platforms with sufficient processing power. This thesis will focus on finding and implementing a neural network optimized for use on mobile embedded platforms.

2.9 Convolutional neural networks

Convolutional Neural networks are structurally similar to “normal” neural networks. However, CNNs only take images as inputs, and they contain layers specifically for computing convolutions.

Generally, CNNs have reoccurring architectural structures. A CNN commonly consists of combinations of reoccurring patterns of layers. Common layers are convolutional, activation, pooling, fully connected, batch normalization, and dropout layers. A typical architectural trait for CNNs is that they usually have only one fully connected layer at the very end. This section briefly introduces the different layer types.

The most characteristic layer type for CNNs is the convolutional layer. A CNN has at least one convolutional layer. The term convolution stands for image processing, where a kernel performing image transformations slides over the input image. Common image editing actions such as blurring, smoothing, or sharpening in any image editing software are all examples of convolutions. These convolutional actions are split up into any arbitrary number of filters. A filter handles one image manipulating action and is essentially a small matrix of a chosen size in pixels. These filters are then applied, or convolved, onto the input volume. Similar to basic image editing techniques, the image manipulation is applied via a sliding window algorithm, where the filter is moved from

the left to the right of the input area. At each stop, it computes the value of each pixel intensity, and the result is transferred to a new image that serves as the output of the layer.

The goal off applying these image transforming convolutions over the input image is to produce so-called feature maps. A feature map is the output of one of the convolution filters and contains the features, edges, of the image. The feature maps produced by the convolutional functions get smaller and smaller the deeper into the network you get. The idea is that by using this method, every class in a classification problem can be reduced into a set of smaller features. If this is possible, the set of convolutions can be used to determine what type of class the input image belongs to. For the model to figure out if an input image contains a cat or a dog, it will reduce the input into smaller and smaller features. The same logic applies to the training, where the correct layers that produce features are learned, and an understanding of what the input image is built up of is achieved.

Every convolutional layer is followed by an activation layer. The activation layer contains a function that decides whether or not the edge should activate, meaning that it determines if the signal should be carried forward in the network. Different types of activation layers can be used to improve the performance of the model. These attributes will be described in later chapters.

Depending on the input images, the produced feature map will contain different features depending on their position in the input. This means that certain class objects have reoccurring positional features. This is a problem as, preferably, the model should be able to generalize and find features from images containing shifts, rotations, and flips as well. This is where pooling layers come in. Pooling layers reduce the size of the feature map and, at the same time, remove the positional variance that the model has. A pooling function is often small, only 2x2 pixels in size. Reducing the layer input size means that it simultaneously reduces the number of parameters in the network, thus improving performance.

Fully connected layers are, as their name suggests, fully connected. That means that the activation layers of the previous layers have connections to each of the nodes in the fully connected layer. In a CNN, the fully connected layers are usually placed at the end of the network structure, before the output layer which produces the classification.

2.10 Training process

The training process of a neural network is the process where input data is fed forward through the network, followed by the calculation of the loss function, and finally, the updating of the network's parameters. This process is repeated over and over again. After each training pass, called an epoch, the algorithm stops to tweak the weights of the network. By utilizing these techniques multiple times, the weights and biases of the neural network are brought closer and closer to values that minimize the loss function, thus producing the smallest error.

As mentioned in previous sections, a neural network has several parameters that need to be set. Some of these parameters are picked randomly, and some are found by trial and error. In general, a few tests are needed to verify what values work and which do not.

Backpropagation is the algorithm that computes the gradient based on the weights with respect to the loss function. The two steps taken are called forward pass and backward pass. The forward pass feeds the inputs through the network until the predictions are produced at the last output layer. The backward pass consists of computing the loss function and updating the weights of the network.

Training is a computationally heavy task. The network has to convey the input through all its layers, and compute the convolutions and other calculations at every step. The training time depends on the network's depth and the hardware the training process is being executed on. Training time can be reduced with more processing power, especially graphical processing units, GPUs, which are well suited for training neural networks thanks to their high bandwidth and thread parallelism. The neural networks which are trained in this thesis will be trained on an Nvidia GTX 980Ti GPU.

Most of the training steps mentioned above are abstracted away when using a machine learning API. In this thesis, the Keras machine learning API will be used together with TensorFlow's GPU module, which is optimized for GPU training. Chapter 5 describes the steps required to train a CNN with these tools successfully.

2.11 Underfitting and overfitting

Even with state-of-the-art hardware, training a deep convolutional neural network can be a time-consuming task. Large networks can take hours, days, or even weeks to finish training.

Due to the time it takes to finish the task of training a neural network it is crucial to have insight into how the network is performing during the training process. This is beneficial since it allows for a quick response by the user if the training seems to take steps in the wrong direction. Most machine learning APIs contain functions that output metrics that help with establishing the fitness of the model. The training performance is outputted during training and functions as a metric to verify how well the model is performing at each epoch. The performance can, for example, be plotted out as a graph where components such as loss and accuracy would be plotted out as separate graphs. Under- and overfitting are metrics that can be used to describe how well the network is performing during training, and they are directly tied to the components mentioned above.

Underfitting occurs when the model shows signs of low variance and high bias. That is, the model is too good at generalizing and, as such, it will never be able to learn. Underfitted models tend not to have enough complexity because of a lack of a sufficient number of layers.

A more common issue faced when training neural networks is called overfitting. Overfitting is the opposite of underfitting. It means that the model has become too good at classifying the training data and, as such, it has problems generalizing. A model that is overfitting will have a difficult time making any accurate predictions on images outside of the training dataset. Overfitting usually occurs when the dataset that the model is training on is too small. There are ways to overcome overfitting, and these techniques will be discussed in chapter 4.

2.12 Haar-cascade classifiers

Before the introduction of CNNs most image detection methods were based on, as mentioned before, hand-determined classifiers, much like the Haar-based classifiers. A Haar classifier has many similarities to the filter kernels in a CNN. However, the filters in a Haar classifier are not learned by training but have been chosen by hand. The Viola and Jones [18] paper on rapid object detection with Haar classifiers is considered to present the first machine learning classifiers that could provide meaningful results in a real-world setting. The technique uses the sliding window method, similar to how filters are used in CNNs, to extract the facial landmarks, also known as the region of interest, from a picture. The Haar classifier contains filters that detect the eye, nose, and forehead regions in an image of a person's face. Figure 8 demonstrates how these filters are used on a face to find the facial landmarks.

OpenCV provides a Haar classifier for object detection, which is based on the original Haar classifier by Viola and Jones. In this thesis, the face classifier in OpenCV will be used to find the region of interest from the video feed. These simple detection solutions are usually computationally more effective than a CNN.

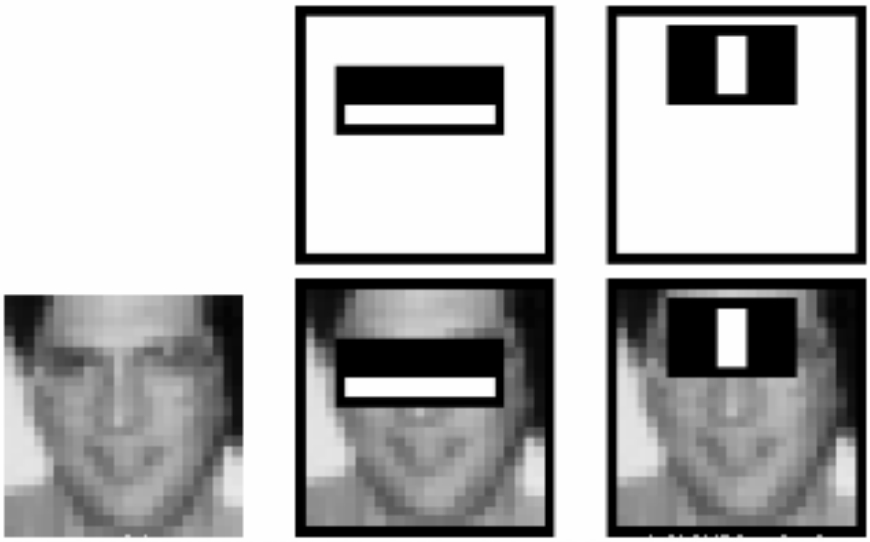

Figure 8. Two filters [19] used by the OpenCV Haar-cascade classifier to find the eye, nose and forehead region on a face. These anchor points are used to determine the borders for the whole face.

2.13 Transfer learning and fine-tuning

Transfer learning is a method where a pre-trained model is used as the foundation for the training of a new model. Transfer learning is commonly used when a pre-trained model is employed as a feature extractor for a new model. The outputs from the pre-trained model are then inputted to the new model and passed through the network. In this thesis, a type of transfer learning is used, called fine-tuning. When fine-tuning a neural network, the pre-trained model has to be modified. Figure N describes the steps of fine-tuning. A pre-trained network has its output layers removed and a new model is formed by adding new output layers to it. The new model consisting of the old pre-trained layers and the new set of output layers will serve as the basis for the training. Models that have been pre-trained on datasets that are challenging and contain a wide array of classes excel at feature extraction. The rich feature extraction capabilities can be used to an advantage when training the new model. The pre-trained layers make training the new set of outputs faster and more accurate. It has been proven that fine-tuning is a highly efficient method for training machine learning models [2]. Before the training starts, the old layers are first frozen. This means that they are ignored during the backpropagation so that their weights do not change. Apart from these steps, training is done the same way as one would during "normal" training.

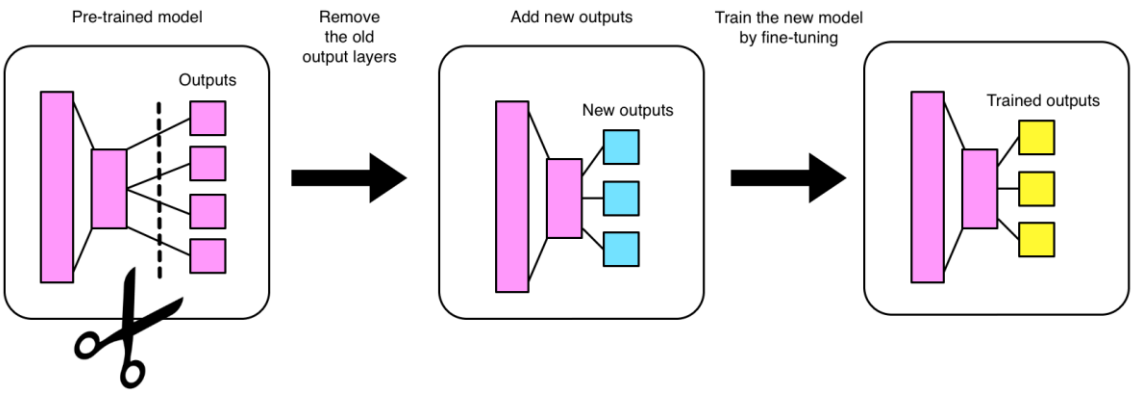

Figure 9. The image presents the flow of the fine-tuning on a pre-trained model

3 CURRENT TECHNOLOGIES AND OTHER PLAYERS

This chapter briefly presents technologies that are used in this thesis, including two datasets and two machine learning frameworks.

3.1 Adience

The Adience dataset by the Open University of Israel is an age and gender benchmark dataset. The dataset consists of 26,580 images of unfiltered faces. The goal of the dataset is to provide images for age and gender recognition that are as true as possible to real-world imaging conditions. The dataset contains face pictures from varying angles, taken in situations with different light and noise conditions. Among its 2284 subjects, the data is organized into both age and gender labels. The gender labels either indicate male or female, whereas the age labels are split into eight different groups. The age labels are divided into groups from, 0-2, 4-6, 8-13, 15-20, 25-32, 28-43, 48-53, 60-years. The figure below shows example images from the gender and age datasets.

Figure 10. Example images from the Adience dataset representing every class [20]

3.2 TensorFlow and Keras

TensorFlow and Keras are both open-source projects and maintained by researchers at Google. Both can be used with Python but are very different in both function and purpose.

TensorFlow is a machine learning platform and backend, and contains pre-trained models that can be downloaded and customized by the user. Both training and inference can be done without the use of any other machine learning tools. However, Keras serves as an alternative high-level machine learning API and library on top of Tensorflow. Keras simplifies much of TensorFlow's actions into consistent and simple APIs which make development both faster and easier. Keras was initially developed as a tool for Google researchers, but gradually, the project caught the interest of the machine learning community, and the project has grown substantially ever since.

3.3 ImageNet

The ImageNet [21] project is one of the more extensive image databases, with over 14 million hand-labeled images, containing over 20 000 classes of objects that can be found in everyday situations. ImageNet is maintained by the Vision Lab at Stanford University and uses crowdsourcing for its labeling process. ImageNet is a popular dataset used as a benchmarking dataset for various deep learning purposes. As the whole dataset is so large, it is common that models are trained on a subset of ImageNet. For instance, the models Google provides via its frameworks are often trained on an ImageNet dataset containing 1000 classes.

3.4 MxNet

The MxNet machine learning framework is developed by Amazon and is provided as a part of their cloud services. The MxNet machine learning library was initially considered as the foundation of this thesis. A short chapter on MxNet's so-called SqueezeNet model will be presented in later chapters.

3.5 Neural compute devices

Inference with a neural network model can usually be done on CPU on most devices. Most frameworks provide functionality for exporting the model into a suitable format and inference capability. These methods work well for desktop or server-based approaches that have more computational power. However, deploying the model on a mobile or embedded platform commonly requires both structural changes to the model and specialized hardware. Similar to how training time is improved with GPUs, inference time can be improved by using dedicated hardware that contains routines for neural computing. The two following sections present two options for specialized neural computing on embedded devices.

3.6 Google Coral

Google Coral is the name of Google's line of AI platforms for accelerated computing. It contains both development boards and separate USB compute units. All the products contain Google's edge tensor processing unit (TPU), coprocessor, which is a separate processor for neural network inferencing. The USB compute unit is especially interesting as it allows for parallel neural computing to be added to any device that has a USB port.

Google Coral devices are compatible with modified versions of TensorFlow models called TensorFlow Lite. TensorFlow Lite models are optimized for running on embedded processors such as the Edge TPU. Creating a TensorFlow Lite model is done with the TensorFlow Lite converter tool. One cannot train models directly to TensorFlow Lite format, and a regular TensorFlow model has to serve as the basis for the Lite model. TensorFlow Lite is an optimized version of the regular TensorFlow model. This means that not all of TensorFlow's layer operations are supported in TensorFlow Lite. A model might require modification to its layers before the conversion can be done.

3.7 Intel Movidius NCS2

Intel offers a USB accessory device called Intel Movidius NCS2. Similar to Google's Coral TPU, it is a small computing device for neural network computing on edge devices, containing a Myriad X Vision Processing Unit. The Movidius compute stick supports a wide range of machine learning frameworks, including TensorFlow. Trained models need to be converted with Intel's so-called OpenVINO toolkit to make them compatible for inference. The Intel Movidius NCS2 is suitable for use with Raspberry Pi devices.

4 DESIGNING THE COMPUTER VISION SOLUTION

This chapter presents what is needed to create a fully functioning computer vision solution. It gives an overview of different design options and describes the design decisions made during the design process. The chapter begins with a presentation of the requirements for the design.

4.1 Requirements

The requirements for the computer vision device are as follows: find a face in an image from a video feed and classify the found face based on age and gender as accurately and efficiently as possible. The classification is done with a neural network model. To create such a model a machine learning pipeline has to be developed.

The solution must take into consideration the hardware limitations of the device it will be deployed on. These limitations impact each part of the project. Therefore, the hardware will be presented first.

Stratacache utilizes a range of Linux-based computers that serve as platforms for its products. The solution produced in this thesis is designed with these devices in mind. The device should attempt to match the company's current hardware stack. Simultaneously, it should serve as a minimum hardware requirement for the computer vision solution. If the computer vision solution works on the Raspberry Pi, it should work on all of the company's devices. This is why it was decided that the Raspberry Pi 3B+ would be a suitable computer vision platform for this thesis project.

The first step to achieve analytics results is to find the face, or the facial landmarks, from an image in a video feed. Finding the facial landmark is done with a face detector.

First, the detector has to locate where the face is in the frame. After that it makes more precise detections to find the area that makes up the so-called region of interest. The region of interest consists only of the sample of pixels containing the face area required to make a classification. The detected region of interest is then used for inference.

There are multiple face detector methods that could be used for finding the region of interest. The biggest difference between them is how much computational power they

would need in order to make a detection. A more sophisticated detector, such as a machine learning-based detector, would possibly allow for wider detection ranges, meaning that the detector could detect a face that is partly turned away from the camera or even partly hidden. However, sophistication means drawbacks in the form of more CPU load. Taking the hardware limitations into account it was decided that a very simple and lightweight detector would suffice for this project. The Haar cascade detector was chosen as it is fast and forgiving on the CPU and, as such, fitting for the computer vision platform. The Haar cascade detector is convenient as it comes built-in to OpenCV, which means that the detector itself does not require any further development. However, the Haar cascade has a few drawbacks. Haar-based detectors tend to be prone to false positives. This means that the detector detects face regions from images that do not contain a face. Other drawbacks are the Haar detector's limited detection angle, meaning that a person who is facing the camera at an angle might not be detected. These drawbacks are still considered negligible as the overall goal is to find a baseline for classification, and future implementations could use more advanced detectors. The solution in this thesis does not account for the detection faults, however, a future product should quite easily be able to filter them out. Classified false positives should stand out among the rest of the classifications as their class probabilities are assumed to be zero.

The next step to produce an analytical result is to compute the class predictions. The region of interest is supplied to the classifier for inference. A CNN that is optimized for embedded or mobile devices is needed as the Raspberry Pi would not be able to run a complex CNN due to the large memory footprint of such neural networks. Two shallow and computationally light CNNs were tested. Firstly, MxNet's SqeezeNet model was considered as an alternative for the machine learning classifier. However, focus quickly shifted over to the MobileNet [22] machine learning model. There were two reasons for this. The original MobileNet paper [22] points out that MobileNet outperforms SqeezeNet in accuracy on the ImageNet dataset, and it aims at keeping the network structure shallow but still able to produce competitive results compared to deeper models. Further, compatibility is important considering future work. A TensorFlow-based machine learning model, which MobileNet is, would facilitate development on both the Google Coral development board and Intel's Movidius Compute stick. The Google Coral computing device is only compatible with TensorFlow models. Future work is discussed

in the last chapter of this thesis. Having chosen the type of the classifier, the next step is training.

The machine learning classifier should be either manually trained from start to finish, or one could try to find a pre-trained model online that would meet the project requirements. It was decided that instead of trying to find a suitable pre-trained model that most likely would contain licenses limiting the use of it for educational use only, a model would be created. Creating a custom model is an interesting approach as it gives insight into how to implement a training backend and how to train and deploy the model. However, re-implementing the network structure of the MobileNetV2 [23] network would be unnecessary work. TensorFlow lets the user instantiate pre-trained MobileNet networks. These models have been trained on a subset of the ImageNet dataset, can classify 1000 classes, and are suitable for transfer learning. The reason for this is that the ImageNet dataset contains a wide variety of objects and, thus, a network that is fine-tuned on it will likely excel at generalizing. A fresh set of output layers, two for the gender classifier and eight for the age classifier, are added to the instantiated base model. Through fine-tuning the new output layers learn the same rich feature extraction capabilities that the original model had acquired.

Figure 11 visualizes the entire system, including the machine learning pipeline and the computer vision platform, starting from the pipeline and the training and ending with the model deployed on the computer vision device performing inference.

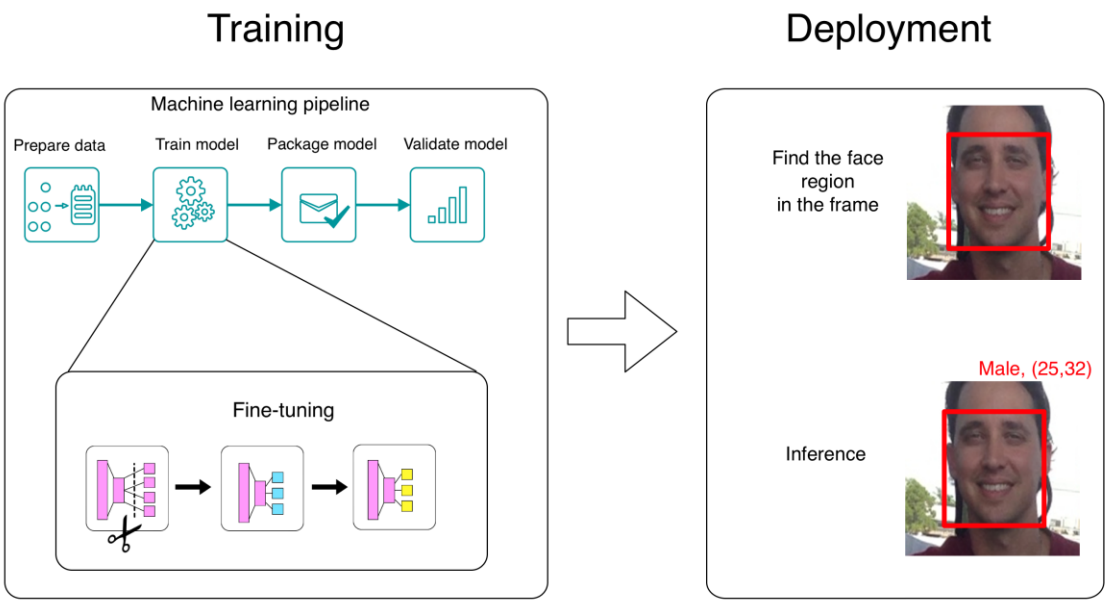

Figure 11. Visualization of both the required steps for the training pipeline and the computer vision device.

To summarize, a substantial part of the design, research, and implementation work will be put on the training pipeline. The machine learning pipeline consist of modules that handle preparing, pre-processing, and loading of data. Additionally, it performs and monitors the training and validation of the models. A robust and reusable machine learning pipeline can be extended in the future. The pipeline could possibly handle tasks such as automatic deployment of models for edge devices. The pipeline could be used as a platform to serve other machine learning projects and can be reused for any improvements related to the created model. The following sections account for each part of the pipeline and the computer vision device more thoroughly.

4.2 Convolutional neural network

Neural networks suitable for computer vision processes are called convolutional neural networks. The required model has to be specifically designed with embedded devices in mind. CNNs are often presented together with their parameter count. The number of

parameters refers to the number of possible filters the network can learn. CNNs usually have tens of millions of parameters. The MobileNetV2 paper [23] presents a shallow network structure, containing around 3 million parameters, depending on the network configuration.

Google has implemented a series of neural networks that they have released under open source licenses and that are included in their neural network framework, TensorFlow. As the name suggests the MobileNet family of neural networks is designed and optimized for mobile devices. TensorFlow contains both version one and an improved version two of MobileNet. Additionally, TensorFlow contains different flavors of the models with varying input sizes and pre-trained weights. The latest version of MobileNet is denoted as MobileNetV2. The pre-trained weights have been acquired through training the networks on a subset of the ImageNet benchmark dataset containing 1000 classes. With the help of the pre-trained model weights a new model will be trained on the Adience dataset through fine-tuning. The next section describes how the training of the models is done in the machine learning pipeline.

4.3 Machine learning pipeline

The training pipeline consists of Python modules that produce and train the MobileNetV2 model. The modules use the Keras machine learning API for most of the parts required for training. The model below visualizes the required steps of a machine learning pipeline. A machine learning pipeline consists of preparing the data, training the model, packaging the model, validation of the model and, lastly, deploying the model. This section is split into subsections containing the motivation behind each part.

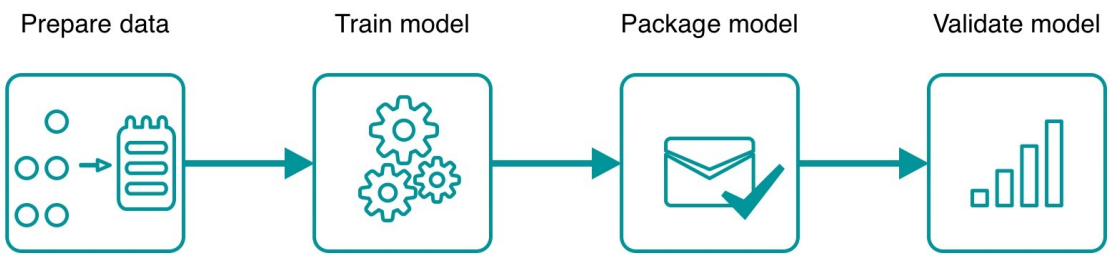

Figure 12. The image shows each of the steps of the machine learning pipeline. Inspiration taken from Microsoft Azures web page [24]

4.3.1 Splitting and preparing the dataset

Preparing data is a crucial step for any machine learning task. The data used for training a machine learning model is seldom structured in a suitable format. Depending on what framework one is working with and what kind of task one is trying to solve, the data usually require some re-structuring. The Adience dataset is no exception. Many machine learning libraries use special file formats to be able to quickly read and handle large amounts of training data. This means that the data has to be repackaged into the correct file structure or format before training. TensorFlow supports a wide range of file formats and luckily the Adience dataset fits into memory so it can be loaded directly from disk. Each data point can be accessed from text files containing the paths to each image on disk.

The Adience dataset is labeled by its maintainers, however, it consists of some faulty labels, and some data points are completely missing labels. This means that the dataset requires all faulty data points to be removed before it can be used for training. It is important to verify the integrity of the whole dataset since the introduction of faulty or missing labels could have an unforeseen impact during the training phase and result in odd behavior. The next task when preparing a dataset is to organize the data into classes. The preparation step requires organization of the data in a way that classification of both age and gender is possible. This could be done in two different ways. One could create a single model which could classify both age and gender at the same time. The second alternative is to create two separate models, one for the age classification and one for gender classification. From a product as well as a business perspective it might be

interesting to provide a separation between age and gender classification. A separation of the two makes it possible to provide only classification of either one of the cases. Furthermore, splitting the model makes it possible to run inference in parallel. This is why it was decided that the data should be structured in such a way that two different models are created.

A supervised machine learning algorithm requires validation data during training. This means that the dataset that is going to be used for training has to be split up into at least two, preferably three, parts. These parts are called training, validation, and testing datasets. As a rule of thumb the parts are split up so that 70% of the data will be in the training dataset and the remaining 30% will be equally split between the validation and testing datasets. To make supervised learning possible the datasets cannot share any data among each other and they need to be separate. The supervised machine learning algorithm uses the training dataset during training as its source of data, and after each training pass the validation dataset is used to compute the validation metrics which in turn are used by the algorithm to tune the model's parameters. The testing dataset is used to perform a final evaluation of the trained model when the training is finished.

With these requirements in mind a module that creates lists of the datasets is needed. First it filters out every faulty data point from the dataset and then it builds the prepared list files for both age and gender training. Further, the data is split into training, validation, and testing datasets according to the rules mentioned above. These lists can now be used during the training of the two classifiers.

4.3.2 Pre-processing

This section discusses the pre-processing that is needed to turn the data point into the form required by the neural network. Further, it presents normalization and regularization, which are methods that will help the models perform better during training. Even though these topics are not directly linked to pre-processing, they are discussed here, as they are conveniently performed at the same time as the rest of the pre-processing.

Pre-processing of individual images is required to make them suitable for the model during training. To make pre-processing quick and convenient a dataset loader module

should be created. Keras contains three different methods for starting the training. These methods are in charge of supplying the model with data during training. The methods are called *fit, fit_generator,* and *train_on_batch*. They all accomplish the same task; however, their function is very different. The *fit* method is suitable for training on small and simple datasets and does not allow for much control of the data. The *train_on_batch* method, on the other hand, will let the user control every aspect of how data is loaded and pre-processed and is typically used when one is training on state-of-the-art datasets. Considering the structure of the Adience dataset and the need for pre-processing, together with the possibility to perform extra regularization, the *fit_generator* method is chosen. The *fit_generator* method accepts the training and validation data in parallel batch jobs in the form of Python generator objects. Loading data with the help of a batch generator is convenient, the data is provided in smaller batches, and thus only parts of the dataset are loaded into memory at one time. The figure below describes the required pre-processing that is done by the batch generator on an image. An image has to be loaded from the list, re-sized, and turned into a NumPy array suitable for training.

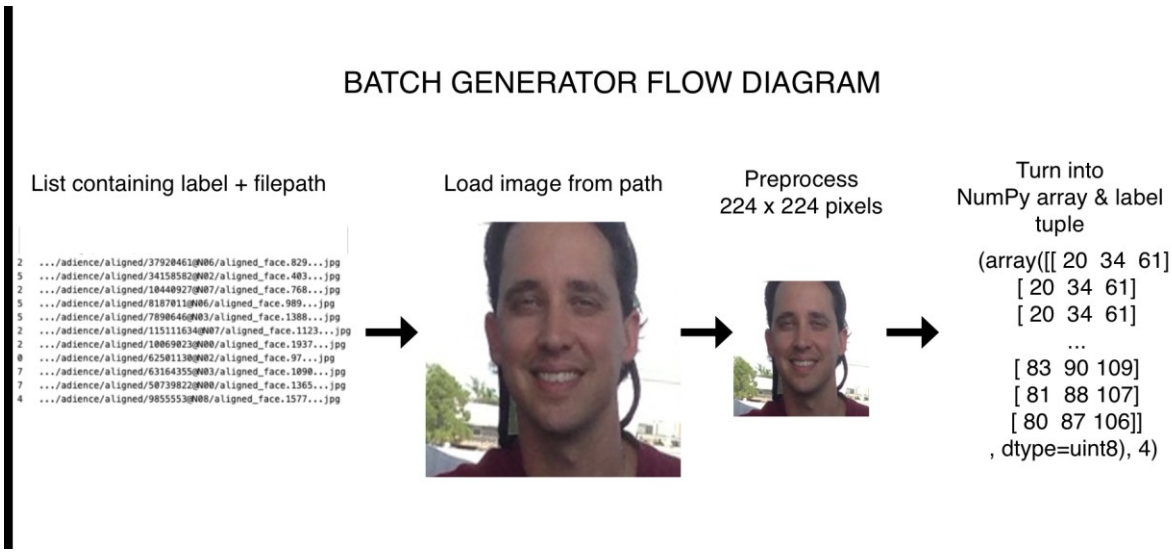

Figure 13. The image describes how data should be loaded and preprocessed in the batch generator

The Python generator should take the list files containing the paths for every data point in the dataset, load the images, and turn them into NumPy arrays. Another benefit of using this approach is that it makes the pre-processing scalable. Pre-processing is handled on the CPU while training is done in parallel on the GPU. Further, the dataset could easily

be extended in the future. Data can be stored in unstructured folder structures as long as the pipeline gets the lists containing the paths to each data point. Extending the dataset with varying file formats, such as TensorFlow hdf5 datasets, which can contain collections of data elements, raw images, or even pure arrays of pixel data, could also be possible. Finally, the module should resize all images into the size required by the input of the neural network. These are the steps necessary to begin training; however, there are still steps that can be done to make training more efficient.

Here two different methods are presented, mean value subtraction and data augmentation. They are not pre-processing steps per definition, as they are not required to achieve training. But since they can significantly improve training results and work directly on the images in the dataset, they should be introduced at the same time as the rest of the pre-processing. Mean value subtraction is a form of normalization that helps to even out the data before training. By performing normalization, the data points' pixel intensities are smoothed. The gradient calculated by the optimizer will behave in a more stable manner with mean subtraction. Without it, the dataset contains data points with varying pixel intensities, which lead to unstable gradients and unstable training results. Mean value subtraction is done by subtracting the mean RGB values from every data point in the dataset.

The second pre-processing step is data augmentation. Keras contains a class for performing data augmentation at the same time as the data is supplied to the model during training from the generator. During training, a model can, due to either the structure of the dataset or the chosen parameters, behave poorly. A common problem is overfitting, which means that the model becomes too good at classifying only the data in its training dataset, but loses the ability to classify data outside of the training dataset. More regularization should be applied to remedy possible overfitting. Data augmentation can significantly improve training performance. The augmentation adds random rotations, shifts, and horizontal flips to every image in the dataset before training. This adds randomness to the dataset, which is desirable to combat overfitting and helps the model to generalize.

With these pre-processing steps combined into one generator module, the training can be started with the required pre-processing and augmentation. The generator class supplies

batches of image and label pairs, looping over the data indefinitely until training is completed.

4.3.3 Training, packaging and, validating the model

The training itself is a task that is repeated over and over again until satisfactory results are reached. Each training pass is called an epoch. Two separate models should be created, one for classifying age and one for classifying gender. As the training is performed through fine-tuning, the pre-trained base models are required to be downloaded with Keras. Keras supports the option to leave out the last output layers from the pre-trained MobileNetV2 models. After the models are downloaded, their layers should be frozen so that the strong generalization capabilities that the network has acquired from pre-training are preserved when training continues. After this, the custom class outputs for the age and gender classification are added. The models have a different amount of outputs depending on what class type we are training on, two outputs for gender, and eight for age.

The training task is started in Keras with the *fit_generator* method. The method expects two instances, one for training data and one for testing data, of the custom Python generator classes to be supplied. The last two requirements for training are the training monitor and checkpointing callbacks. These callbacks are built into Keras and need to be added to the *fit_generator* to provide extra functionality.

The training monitor creates plotted graphs of the training metrics and can output metrics to the console during the training. Training metrics are essential and enable visual validation of the training progress and serve as a method to determine which of the created models is the best one. The charts are updated after each epoch. The model performance is calculated by computing accuracy metrics based on the supplied validation and training datasets. By keeping an eye on these metrics, poor model performance can be spotted early in the training, and the process can be canceled. This way, unnecessary time spent on training unsuccessful models is avoided.

The second Keras callback required is the checkpointing callback. The checkpointing callback will continuously save a new copy of the model at every epoch of training. This ensures that training results are not lost in an unforeseen situation or if an error occurs.

Training is finally stopped by the user when acceptable results have been reached. When the training process finishes, the model should be saved one last time, and validation of the finished model is done. Now the trained models can be deployed onto the computer vision device for real-life testing.

4.4 Computer vision device

This section accounts for the design choices and requirements for the computer vision device. The process contains three steps, polling for frames, finding the facial region, and inference.

Polling for frames from the camera is done with OpenCV's *read* method. This method is a blocking I/O method and, therefore, it should be parallelized to reduce any impact on the rest of the system. Parallelization is done by utilizing the threads of the CPU to separate the frame polling from the rest of the object detection and inference tasks. Threading could be implemented from scratch, but during the development of the project, a convenient library containing various computer vision tools was found, called Imutils. The Imutils package contains a built-in class for parallel polling of frames from the Raspberry Pi camera module. Therefore, this package should be included to handle the parallelization tasks of the computer vision device.

The computer vision software on the Raspberry Pi will use OpenCV's Haar cascade detector to find faces from the video stream coming from the camera module. The Haar-cascade classifier was chosen as it is conveniently included in OpenCV and is one of the most computationally light detectors available. There are other possibilities for object detection, such as neural network-based detectors, but they were not considered due to the processing power they would require.

After that, the device should use the trained models for computing classifications on age and gender. The saved and packaged models outputted by the pipeline are saved in Keras format. Deploying them to the computer vision devices required Keras to be used for inference. Keras performs the predictions on the found facial regions and produces the required metrics. The inference could be done using other tools that Keras. As an

example, OpenCV contains a method for loading a neural network model directly and utilizing it for inference. To enable compatibility with OpenCV, an exported Keras model has to be converted to TensorFlow format. Exporting and converting models is a topic that requires further research and is, therefore, left outside the scope of this thesis. Possibilities of other inference methods are discussed further in chapter 7.

For testing purposes, the device is required to visually output the acquired frames and draw a bounding box around the detected face and its classified labels. Finally, it should time the classification times and count the frames processed per second. These metrics are presented in the next chapter.

5 DESCRIPTION OF THE MACHINE LEARNING PIPELINE

This chapter explains how the training was performed and what steps were required to produce the age and gender models. The chapter ends by presenting the training results for both the age and gender models separately.

5.1 Training process

The training script is started by executing a training script. The pipeline fetches the pre-trained network and performs the required modifications to it depending on what type of model is being trained. The pre-processing Python generator resizes images from the disk to the chosen model input size of $224\ x\ 224\ x\ 3$ pixels. After that, the images are converted into NumPy arrays and yielded to the training process together with their labels. Random horizontal flips, sheers, and rotations are applied. After this, the actual training begins.

The most crucial hyperparameter to begin training with is the learning rate. The learning rate variable controls the steps taken by the optimizer. Even though Keras can automatically adjust the optimizer slightly during training, it is essential to find a suitable starting value. The learning rate is a small positive number between 0.0 and 1.0. Typical learning rates are 0.1, 0.01, 0.001. Finding the optimal learning rate requires a few tests so that a good baseline for the training can be found.

During the training, the training monitor callback outputs the accuracy metrics for the model so that the performance of the training can easily be followed by analyzing the console output or by looking at the training graph. The pipeline outputs a training graph containing loss and accuracy for both the training and validation datasets during every epoch, as is commonly done when training neural networks. From these graphs, one can quickly determine if the training is achieving desirable results, or stop it if it seems that it is not. During a well-performing training process, the curves of the accuracy and loss graphs follow a recurrent pattern. The loss and accuracy tend to have an exponential curve. The loss starts high and decreases quickly, whereas the accuracy starts low and will increase rapidly during the epochs. After a certain number of epochs, both curves are going to flatten out. A flat curve is usually a sign of the model reaching its maximum

accuracy and minimum loss. At that point, the training should be stopped, and one can suppose that the best possible result has been found, or one could try changing the learning rate once more and restart training from the last checkpoint.

If the curves deviate from the uniform increase or decrease, it is likely a sign of a problem. Overfitting and underfitting are both quickly noticeable from the plotted graphs. An underfitting model will produce oscillating curves, where the loss and accuracy increase and decrease every epoch. Overfitting models tend to have a high training accuracy, which would signify that the model is performing well. However, the validation accuracy is likely significantly lower, and the validation loss is high as a result of the validation step at the end of the epoch where the output is cross-referenced with an actual label.

The training process itself is experimental in nature as it requires multiple training runs to find suitable parameters and to discover possible flaws in the design. The training was carried out during a month when different parameters, pre-processing methods, and modifications to the model were tested. The training process was performed step by step. It started by training models without data augmentation to build an understanding of the training process. This way, each part of the pipeline was tested, and an understanding of what combinations produced desirable results was reached. This process helped to identify any weaknesses that the pipeline might have. Gradually, hyperparameters were tweaked, data augmentation was added, and possible flaws in the design were identified and fixed. The benefits of utilizing TensorFlow's GPU accelerated training capabilities became apparent during the experimentation phase. Training one model on either of the datasets was surprisingly fast. These steps are discussed in more detail in the following section. The time for one epoch to finish did vary slightly, but generally, one epoch took around 2 minutes. As the next section will present, most models did quickly reach acceptable training results. This meant that training one model took roughly half an hour.

5.2 Training results

This section presents the results of the training of the age and gender MobileNetV2 models. It accounts for the adjustments that were made to the model hyperparameters to enable the models to start training. Further, it presents the discovered weaknesses of the pipeline and mentions what measures were taken to assist training performance.

The training was started on the gender dataset. It took several attempts to tune hyperparameters before acceptable results were achieved. Most of the initial attempts ended with the model not being able to learn at all. Training validation was noisy, which resulted in an oscillating validation curve. The learning rate was too high, and the optimizer was taking too big steps over the loss landscape, sometimes finding a minimum only to step out of it during the next epoch. The initial learning rate was lowered, and the model started to produce more stable results. It was found that an initial learning rate of 0.001 worked well for training the gender model. Even with the acquired learning rate, the model started rapidly overfitting after the few first epochs. It was clear that something was wrong, as the model should fairly quickly stabilize instead of overfitting. The pipeline required improvements. Data augmentation was included during the pre-processing step. Data augmentation rotates and flips the training data before training and helps the model to generalize. Another improvement that was added was the mean value subtraction. Mean value subtraction was performed on the dataset before it was loaded. Mean value subtraction normalizes the pixel intensities of each image in the dataset. This will make the optimizer more stable as the dataset is more centered, and outliers are moved thanks to the normalization step. The data augmentation helped to alleviate the overfitting, and the normalization made the training more stable. One could see that the model started to find a baseline. Figure 14 shows that during the first few epochs, the loss quickly dropped, and accuracy was improving.

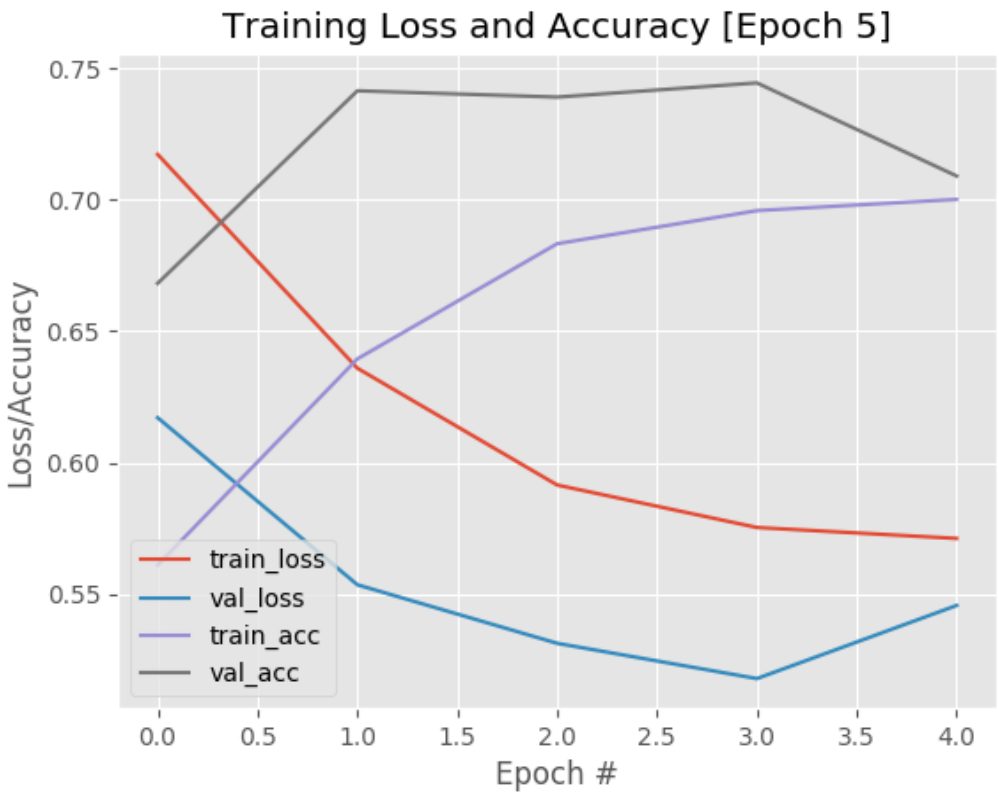

Figure 14. The loss/accuracy chart shows the beginning of the training as the curves are getting their characteristic form

In Figure 15, one can see that after epoch eight, both the loss and accuracies started to show signs of overfitting. The validation loss curve was steadily increasing from epoch eight onward. The reason for this could be the learning rate still being too big. The training was continued for a few more epochs, and at epoch fourteen, still no signs of improving results could be noticed. The training was manually paused, and the initial learning rate was lowered once more down to 0.0001 to see if the model would improve even more.

Figure 15. The loss/accuracy chart shows how the model is slightly overfitting around epoch 8.

Figure 16 shows the training plotted from epoch eight onward with the new learning rate. The model was stabilized and was left training until epoch twenty-four. By then, it seemed that the best possible result had already been found and no better results were possible. The final results of the gender model achieved an accuracy of 74,5% on the gender dataset, which was reached at epoch 3.

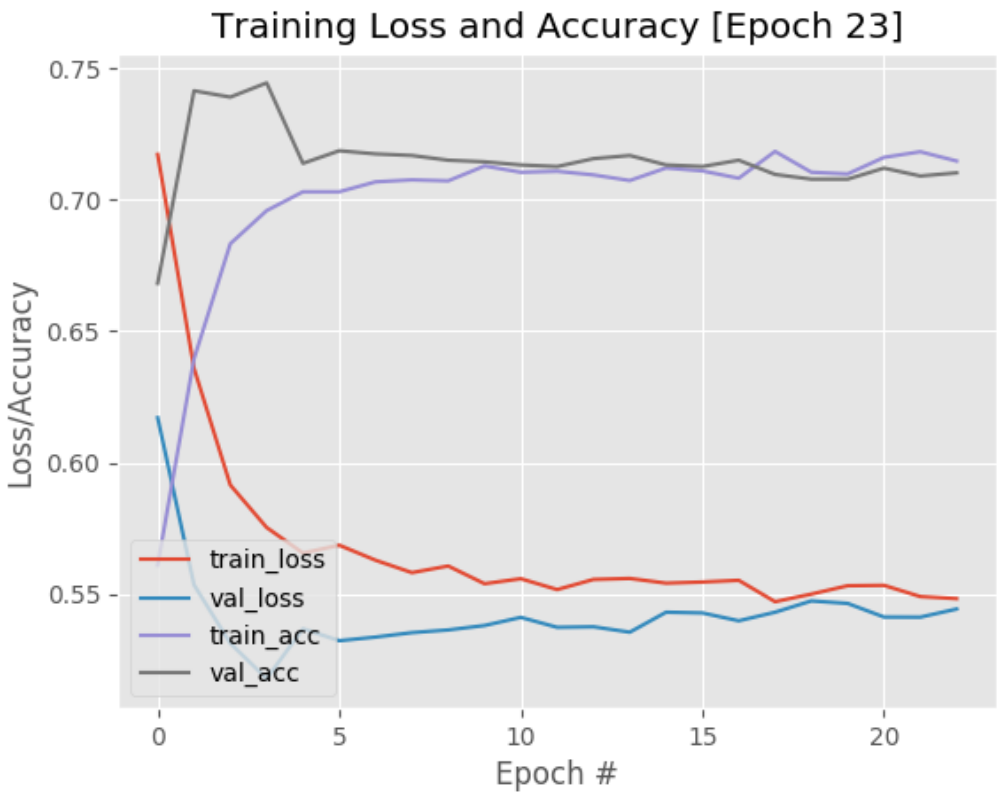

Figure 16. The loss/accuracy chart shows how training is continued from the checkpoint at epoch 6 with a new learning rate.

Figure 17 shows the summary of the model parameters after training and the ration of frozen and non-frozen weights. One can see that the final model has a total of 2.5 million parameters, and roughly 320 000 of them were part of the added custom gender outputs. The saved model had a size of 12.1 megabytes.

```
=================================
Total params: 2,586,177
Trainable params: 328,193
Non-trainable params: 2,257,984
```

Figure 17. The parameter summary of the trained gender model

The age dataset turned out to be more challenging. The training was started with the same learning rate of 0.001 as in the previous experiment. However, it turned out to be challenging to get the model to perform well. Even with the added pre-processing and normalization steps, the model had trouble finding a good baseline. The reason for this is yet uncertain, but a detailed discussion will be presented at the end of this chapter. Figure 18 shows the loss and accuracy plot for the age model. The curves do not show their characteristic form, and it is as if they were stuck at a local minimum from the beginning.

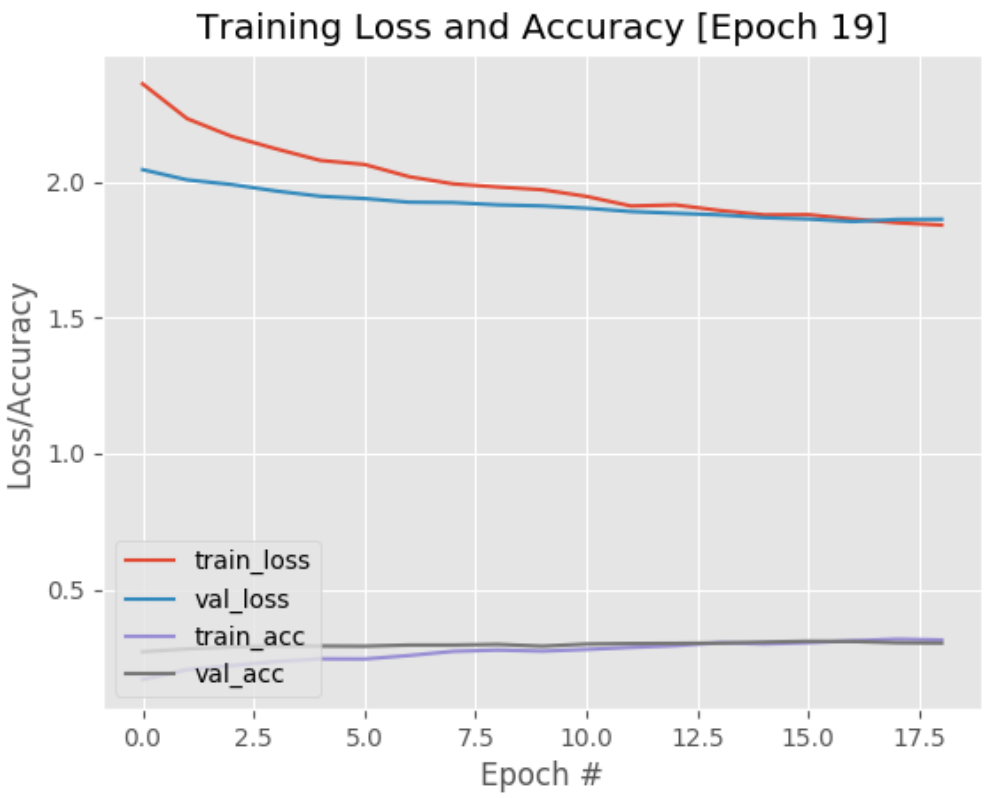

Figure 18. The loss/accuracy plot for the age model

The model stopped improving after epoch 16 and achieved an accuracy of 31% on the age dataset. Figure 19 presents the parameter summary. Similarly, to the gender model, the age model contains 2,5 million parameters, and 329 000 of them were trained for the added age outputs. The age model has an identical size to the gender model of 12.1 megabytes.

```
=================================
Total params: 2,587,976
Trainable params: 329,992
Non-trainable params: 2,257,984
```

Figure 19. The parameter summary of the trained age model

5.3 Discussion

The experiments discussed in this chapter accounted for the training of the age and gender MobileNetV2 models. Training a neural network model is a complicated task, and as expected, it took a large number of experiments until an acceptable model was produced. During the time of roughly a month, different parameters and methods for supporting training were tested. This development phase pinpointed weaknesses and flaws in the design. Thanks to this, the pipeline was improved, and stable results for the gender model were produced.

The gender model turned out to be easier to train than the age model. This is most likely due to gender classification only being a two-class problem. Acceptable accuracies as high as 74% were reached. However, more work is required to find and to solve the issues related to training the age classifier. Results indicate that there is an underlying issue with the training, either related to the data or to the structure of the network. Future research will have to find which modifications will enable the model to learn. Thankfully, during the research process, multiple options to solve the issues where identified, and these are presented in detail later in this section.

The training process itself was faster than anticipated. Thanks to TensorFlow's GPU training module, the time to finish one epoch was roughly two minutes. The time to train one model was approximately half an hour. This is beneficial, as the time for prototyping and overall development is significantly reduced. Additionally, if future work involves training a model from scratch, instead of fine-tuning, the GPU-assisted training pipeline will have a significant impact on the overall training time

The designed pipeline did facilitate and support training significantly. However, some limitations were found and addressed during training, such as the lack of data augmentation and normalization. The amount of data in the dataset is considered to be sufficient, as the Adience dataset is widely used as a benchmark dataset to measure how well models perform. However, spending time gathering and labeling additional data to the dataset would not be a bad idea, and could help in achieving even better training results.

Training the age classifier turned out to be significantly harder than expected. It is possible that there are multiple factors behind the poor training results. As a note, the training itself could have been left running for longer. The longest training experiment was just under 40 epochs. But even then, the model did not show any signs of improving.

Firstly, there might be some fundamental problems with the training data itself, which make it unsuitable for performing fine-tuning on the pre-trained model, as fine-tuning relies on the similarities between the new dataset and the one used for pre-training in order to achieve results. There is a possibility that the structure of the data or the images themselves in the Adience dataset are so different to the pre-trained network that it cannot make clear classifications on the training data during fine-tuning. This, however, is unlikely, due to the broad range of classes in the ImageNet dataset. As the ImageNet dataset contains classes such as *person,* one would assume there should be enough similarities to the Adience dataset to make training feasible. If this should turn out to be the reason for the poor performance, one should try to train a MobileNetV2 model completely from the beginning without any transfer weights.

Another possible reason for the model's poor performance is linked to the structure of the network's layers. It is possible that to classify the age groups the network would require a more complex output structure. Adding more untrained layers before the outputs could be required so that the network would have enough complexity to learn the features found in the dataset. Adding complexity would increase the number of parameters of the model, but would probably not impact inference times.

The last identified issue with the age classifier ties together the dataset and the optimizer. The model was trained with a stochastic gradient descent optimizer. The Adience dataset for age classification is clearly more challenging than the gender dataset. The model did have trouble finding a minimum, and this might demonstrate the limitations of standard

stochastic gradient descent. To help the model find a minimum, a more advanced optimizer, such as Adam, should be considered. The Adam optimizer is designed to achieve results on large datasets with challenging loss landscapes. Fine-tuning the network with the Adam should not be too different from how it was done with the stochastic gradient descent optimizer.

To conclude, both the gender and the age models have parts that could be improved upon. Future research is needed to discover the reason for the poor performance of the age classifier. Extending the monitoring callback to output images with their corresponding labels for visual inspection of the training output could help when debugging. As both models are based on the same dataset, it is likely that any solutions found for the age model would improve the performance of the gender model as well.

6 EVALUATION AND TESTING OF THE COMPUTER VISION DEVICE

This chapter contains tests and the evaluation of the computer vision device. The tests account for the computer vision device's performance during real-life situations. The first test is a frame rate test where the device's potential performance is measured with and without a face in the frame. The second test measures how long it takes for the computer vision solution to make a prediction from the moment a face is found until a prediction is made. The third test measures how long it takes to load the model, and the last test contains an example image of what a successful classification looks like visually.

6.1 Performing inference on the computer vision device

The platform for the computer vision device is a Raspberry Pi 3B+ single-board computer. The Raspberry Pi comes with a 1.4 GHz 64-bit quad-core ARM A53 processor. The memory chip consists of a 1 GB LPDDR2 SDRAM. The device has the trained models saved on disk and a Raspberry Pi camera module attached to it. The detection and classification are done in parallel to the reading of camera frames, to improve the frame rate. Parallelization is done with the Imutils *VideoStream* class that handles the blocking I/O methods.

6.1.1 Performance test 1: Frames per second

To measure a baseline for the maximum possible frame rate, the camera view is left without a face in the frame. During this test, the device achieves 2 fps on an average. When a face enters the frame, the fps drops to just below one fps on average.

6.1.2 Performance test 2: Inference time

The time it takes for the model to calculate the inference is measured from when a face is found to when a classification is outputted. The device outputs the class labels both visually, drawn on the camera feed, and to the command line. On average, the inference takes 1.8 seconds.

6.1.3 Performance test 3: Model loading time

By executing a shell script, the process for age and gender classification is started. The startup includes loading both the neural networks, and their loading time is measured on startup. On average, it takes 140 seconds for the device to load both neural networks.

6.1.4 Example output

Two random images, one from the female and one male class, were picked from the testing dataset. As they belong to the testing dataset, one can be assured that the models have not seen them during training. The images have been classified by the computer vision device the same way as a frame from the camera would be. A bounding box highlights the region of interest, and the inferred labels are displayed above the face. Interestingly, the poor accuracy of the age model expresses itself as a reoccurring faulty classification. The classifier consequently labels an input to belong to the third class, 8-12 years. Poor performance in the form of incorrect labels was expected, but the reason why the third class dominates among the classifications is unknown.

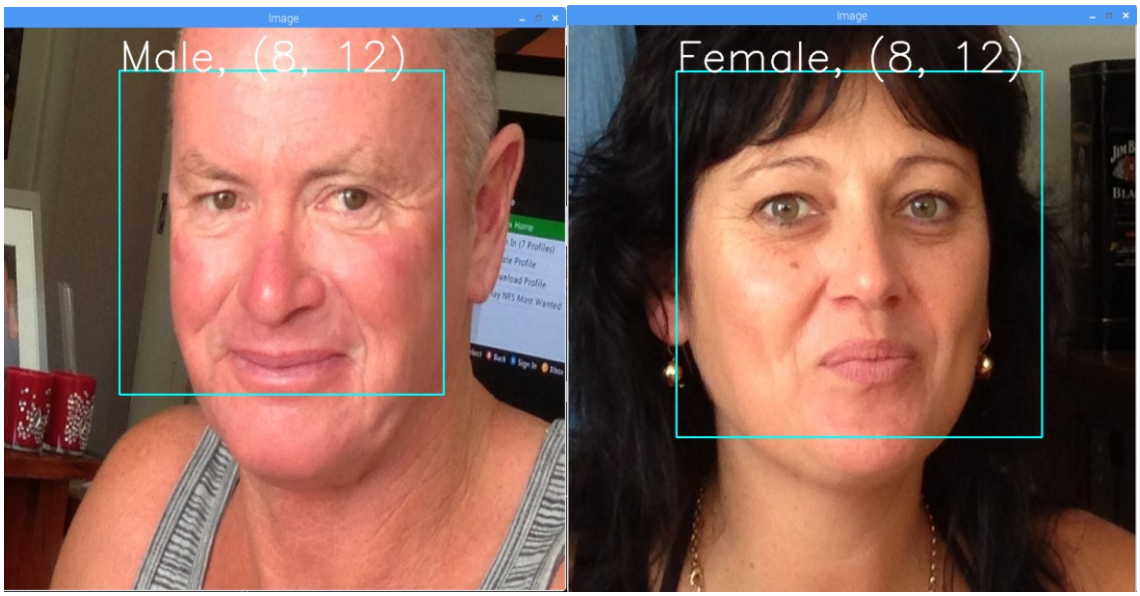

Figure 20. Screenshots of with visualized classifications produced by the computer vision device.

6.2 Discussion

A computer vision device based on the Raspberry Pi 3B+ was created to perform age and gender classification from the input of a video stream. The tests demonstrated that inference on age and gender is possible on the chosen platform, even though the hardware limitations are obvious.

Both the platform's hardware, and the models themselves do introduce limitations on the performance. The models are now loaded as pure Keras models, which are not optimal for use on embedded devices. The Keras models contain extra metadata regarding the neural network itself, which is not needed when computing inference. TensorFlow allows exporting Keras models into a more optimized format, which removes the additional data from the model. This would require freezing the models and converting them to the so-called protobuf format. For the scope of this thesis, the optimizations are left out as they require more research and possibly modifications to the computer vision stack.

Computing inference is a costly process, and changing platform or adding a USB-based neural compute unit, such as the Google Coral or the Movidius NCS2, to the Raspberry Pi would speed up inference times significantly. The table in figure 21 shows benchmarking results of inference with MobileNetV1 and MobileNetV2 on various platforms. These models have been downloaded directly from TensorFlow and classify 1000 ImageNet classes. It is interesting to see that the Coral USB accelerator offers substantial speedups compared to the Raspberry Pi's CPU. There is an 11x speedup between the Raspberry Pi 3, at 654 ms, compared to the Google Coral compute unit, at 58 ms, on inference times. Comparably, the Intel Movidius NCS2 provides a 5x speedup from the base TensorFlow model running on the Raspberry Pi CPU. It is assumable that the age and gender models trained in this thesis should follow similar speedups as they fundamentally are MobileNetV2 models.

Board	MobileNet v1 (ms)	MobileNet v2 (ms)
Coral Dev Board	15.7	20.9
Coral USB Accelerator	49.3	58.1
NVIDIA Jetson Nano (TF)	276.0	309.3
NVIDIA Jetson Nano (TF-TRT)	61.6	72.3
Movidius NCS	115.7	204.5
Intel NCS2	87.2	118.6
MacBook Pro	33.0	71.0
Raspberry Pi (TF Lite)	271.5	379.6
Raspberry Pi (TF)	480.3	654.0

Figure 21. The table [25] contains benchmark results of inference times of TensorFlow's MobileNetV1 and MobileNetV2 models across different platforms.

Optimizing the models is interesting both from the point of view of the hardware and models themselves. OpenCV supports loading protobuf models, which would make the model loading faster. The optimizing step leaves out unwanted metadata from the model and simplifies some of the network's layers. It is uncertain if the current model structure could be optimized directly or whether the network would require topological changes before optimization can be done.

Additionally, optimizing the models is required to make them compatible with other platforms such as the Google Coral or Movidius NCS2. Google Coral devices only support TensorFlow lite models. TensorFlow lite was an area that was not explored in this thesis and would require familiarization.

7 CONCLUSION AND FUTURE WORK

7.1 Conclusion

This thesis set out to research the possibilities of computer vision-based age and gender classification for the retail environment. The task was to, as accurately as possible, and still respecting the individual's integrity, produce age and gender analytics from a video stream.

A machine learning pipeline was created by utilizing the latest technologies and frameworks. This pipeline performs the tasks related to training machine learning models. Two convolutional neural network-based age and gender classifiers were trained and validated. The models were deployed on the computer vision platform for inference and testing.

Firstly, the proof-of-concept product meets the specified requirements. The computer vision solution proves that execution on the chosen single board computer platform is possible, and it can perform inference on age and gender. Despite this, the age classifier's results leave much to be desired. The age classifier did not reach satisfactory results during training and, as such, the classifier does not produce accurate results.

Secondly, the developed machine learning platform is a suitable base to create other future machine learning-based projects. Choosing a pipeline structure for training was vital as it supports easy modifications and extensions. The modularity of the pipeline enables easy change of pre-processors or training of other types of neural networks. Theoretically, in the future, the pipeline could even be extended to handle automatic deployment to edge devices.

Finally, the chosen neural network library is considered to be successful. TensorFlow models support future work well and fit development platforms such as Google Coral TPUs.

7.2 Future implementation

The solution developed within this thesis serves as a good ground for further research, if considered as a basis for future products for the company.

The models produced in this thesis are loaded as pure Keras models. Within the scope of this thesis, the subject of optimization was not studied. The raw models contain metadata that is not needed for inference. TensorFlow includes tools for optimizing a model by removing the extra metadata. This thesis recommends that special investigation into the suitability of optimized TensorFlow models and OpenCV should be considered.

Both classifiers were trained by fine-tuning. Fine-tuning on the age dataset turned out to be especially demanding. One explanation for the poor performance could be the unsuitability of the combination of data and chosen model when fine-tuning. Training the classifiers with normal methods could result in better accuracies. Alternatively, fine-tuning could be tried with adding complexity to the model, adding more training data, or changing optimizers.

At the time of writing, version two of TensorFlow has been released. Version two contains significant changes to the previous version and becomes a complete ecosystem with Keras natively integrated. This means that going forward, Keras should be used within TensorFlow itself via the so-called *tf.keras* package. This integration will make development more manageable as the two are incorporated, and issues related to version mismatches are less likely to occur. The new TensorFlow version includes better multi-GPU training, provides finer control of building custom models, and includes the TensorFlow Lite package for optimizing models for embedded platforms. With these improvements in mind, the pipeline should be updated with the new TensorFlow version.

As this thesis demonstrates, hardware has a significant impact on the performance of computing inference. Future work could research utilizing specialized hardware for computing inference, either by changing the single-board platform or by adding a USB-accessory for neural computing. Additionally, it is pointed out that at the time of writing, the latest model of the Raspberry Pi single board computer is the model 4, with improvements to performance in the form of faster hardware.

SUMMARY IN SWEDISH – SAMMANFATTNING PÅ SVENSKA

DATORSEENDE SOM ETT ANALYSVERKTYG I AFFÄRSOMGIVNING

Detta diplomarbete har utförts av en arbetstagare på företaget Stratacache oy. Projektet innefattar utvecklandet av ett koncept för en produkt för datorseende.

Företaget Stratacache oy verkar inom branschen för försäljningsanalys. Dess kunder finns inom försäljning såsom detaljhandeln eller restaurangbranschen. Stratacache oy levererar skräddarsydda analysverktyg för att ge sina kunder övergripande information om kundernas beteende. Dessa data sätts ihop genom att kombinera flera olika datakällor. Datakällorna inkluderar bland annat trådlös internettrafik, transaktioner från kassaregister samt data från dörräknare. Genom att kombinera dessa kan Stratacache oy ge sina kunder en detaljrik analys över människors beteende som i sin tur kan möjliggöra effektivering av försäljning. Att hålla sig välinformerad om branschutvecklingen är viktigt för företaget. Därför intresserar sig företaget för att undersöka användandet av datorseende som datakälla för försäljningsanalys.

Utvecklingen av inbyggda datorsystem har framskridit inom den senaste tiden. Denna utveckling möjliggör användandet av mer sofistikerade metoder för datorseende. Datorseende innefattar tekniker där en dator används för att bilda en uppfattning om den synliga världen. Datorn analyserar bilder i en videosekvens för att försöka bilda sig en uppfattning av vad bilden innehåller.

Artificiella neuronnät samt Haar-detektorer innefattar de tekniker som står för datorseendet i denna avhandling. För att genomföra en klassificering behöver systemet för datorseende först hitta den del av bilden som innehåller området som ska analyseras, varefter en faktisk analys kan genomföras. Alltså är processen tvådelad.

Haar-detektorn används för att hitta den del i en bild där objektet som ska analyseras finns. Därefter skickas denna del till ett neuronnät för klassificering.

Artificiella neuronnät är nätverk som försöker imitera det nätverk av synapser och neuroner som finns i människohjärnan. Strukturen för ett artificiellt neuronnät är trädlik och byggs upp utav ett godtyckligt antal lager av noder från neuronnätets ingångar till

dess utgångar. Det artificiella neuronnätet kräver träning för att kunna utnyttjas för tillämpningar inom datorseende. Utgående från exempeldata försöker man genom träning lära nätverket att hitta mönster i de data som presenteras. Det tränade neuronnätet kan senare användas för att programmatiskt känna igen beståndsdelarna av en bild.

Detta arbete innefattar utvecklandet av en maskininlärningsplattform för att producera ett artificiellt neuronnät. Det neuronnät som utvecklas i avhandlingen är av typen faltningsnätverk. Faltningsnätverk är en typ av neuronnät som är specialiserade på bildigenkänning. Denna maskininlärningsplattform fungerar som ett verktyg för att möjliggöra fullständig träning av det artificiella neuronnätverket. Ytterligare sköter plattformen de förberedande stegen och stödjande processerna som krävs för att möjliggöra slutförandet av träningen av det artificiella neuronnätet. Slutligen driftsätts de tränade neuronnäten på en enkortsdator av typen Raspberry Pi 3B+.

Programmeringsspråket Python samt biblioteken Keras och TensorFlow används för utvecklingen av de system som används i avhandlingen. TensorFlow och Keras är populära bibliotek som används för utvecklande av maskininlärningsalgoritmer.

Med hjälp av den utvecklade maskininlärningsplattformen tränas två olika faltningsnätverk. Det första står för klassificerandet, eller inferensen, av kön och det andra för inferensen av ålder.

Neuronnätet för könsklassificering når en noggrannhet på 74,5 % under träning medan neuronnätet för ålderklassificering når en noggrannhet på 31 %.

De båda neuronnäten driftsätts på en enkortsdator för att genomföra tester i realtid. Det första testet mäter hur många bilduppdateringar som är möjligt per sekund, både med och utan ett ansikte i bilden. Det andra testet mäter den tid det tar att ladda neuronnäten medan det tredje testet mäter den tid det tar att utföra en fullständig klassificering.

Under bilduppdateringsfrekvenstestet nås en genomsnittlig frekvens på 2 bilder per sekund utan ett ansikte i rutan. Då ett ansikte finns i bilden mäts en frekvens på 1 bild per sekund.

Inferenstiden för en genomförd klassificering är i genomsnitt 1,8 sekunder och tiden det tar att ladda nätverken är 140 sekunder.

Överlag uppnår konceptprodukten sina utsatta krav. Lösningen för datorseende är exekverbar på den valda enkortsdatorn och den klarar av att göra inferens på kön och ålder i en videosekvens. Trots detta lämnar neuronnätens noggrannhet rum att önska. Som framtida arbete lämnar denna avhandling rum för förbättrande av de tränade neuronnäten. Förbättringar kan göras i form av optimeringar på nätverkets filformat men även faktiska omstruktureringar av dess nätverksstruktur kan behövas. Exempelvis innehåller det filformat som de artificiella neuronnäten är exporterade i mycket metadata. Denna metadata behövs under träningsprocessen men är onödig för utförandet av inferens. I framtiden bör man undersöka vilken typ av förbättringar som bäst lämpar sig för att optimera modellen för inferens. En klar orsak för ålderklassificerarens låga noggrannhet går inte att bestämma. Däremot konstateras att neuronnätets struktur och träningsdatas natur kan vara möjliga orsaker till varför ålderklassificeraren når en nöjaktig noggrannhet.

Dessutom konstateras att den utvecklade maskininlärningsplattformen kan stödja eventuella framtida arbeten på de neuronnät som här presenteras samt fungera som stöd för företagets andra projekt inom datorseende. Ytterligare framhålls att valet av neuronnätverk tillåter fortsatt utveckling på ett brett urval av plattformar för enkortsdatorer.

BIBLIOGRAPHY

1. Rosebrock A. Deep Learning for Computer Vision with Python: Starter Bundle. PyImageSearch; 2017. (Deep learning for computer vision with Python).

2. Rosebrock A. Deep Learning for Computer Vision with Python: Practitioner Bundle. PyImageSearch; 2017. (Deep learning for computer vision with Python).

3. Rosebrock A. Deep Learning for Computer Vision with Python: ImageNet Bundle. PyImageSearch; 2017. (Deep learning for computer vision with Python).

4. Bradski G. The OpenCV Library. Dr Dobbs J Softw Tools. 2000;

5. Chollet F, others. Keras [Internet]. 2015. Available from: https://keras.io

6. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Internet]. 2015. Available from: https://www.tensorflow.org/

7. Rosebrock A. jrosebr1/imutils [Internet]. 2020 [cited 2020 Mar 25]. Available from: https://github.com/jrosebr1/imutils

8. van der Walt S, Colbert SC, Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. Comput Sci Eng. 2011 Mar;13(2):22–30.

9. EUR-Lex - 32016R0679 - EN - EUR-Lex [Internet]. [cited 2020 Mar 25]. Available from: https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng

10. Vinci L da, page CG de tableaux en très haute définition: image. Italian: Ritratto di Monna Lisa del GiocondoPortrait of Mona Lisa del Giocondotitle QS:P1476,it:"Ritratto di Monna Lisa del Giocondo" [Internet]. 1503 [cited 2020 Mar 10]. Available from: https://commons.wikimedia.org/wiki/File:Mona_Lisa,_by_Leonardo_da_Vinci,_from_C2RMF_retouched.jpg

11. Plotke M. English: Before any kernel convolution. [Internet]. 2013 [cited 2020 Mar 17]. Available from: https://commons.wikimedia.org/wiki/File:Vd-Orig.png

12. MortenZdk. English: Gaussian blur 5x5 of related image [Internet]. 2016 [cited 2020 Mar 17]. Available from: https://commons.wikimedia.org/wiki/File:Vd-Blur_Gaussian_5x5.png

13. ILSVRC2012_val_00004465.png (473×253) [Internet]. [cited 2020 Mar 10]. Available from: http://www.image-net.org/challenges/LSVRC/2014/ILSVRC2012_val_00004465.png

14. Eidinger E, Enbar R, Hassner T. Age and Gender Estimation of Unfiltered Faces. IEEE Trans Inf Forensics Secur. 2014 Dec;9(12):2170–9.

15. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. ArXiv14126980 Cs [Internet]. 2017 Jan 29 [cited 2020 Apr 6]; Available from: http://arxiv.org/abs/1412.6980

16. Block HD. The Perceptron: A Model for Brain Functioning. I. Rev Mod Phys. 1962 Jan 1;34(1):123–35.

17. general-diagram-of-perceptron-for-supervised-learning.jpg (667×306) [Internet]. [cited 2020 Mar 10]. Available from: https://www.simplilearn.com/ice9/free_resources_article_thumb/general-diagram-of-perceptron-for-supervised-learning.jpg

18. Viola P, Jones M. Rapid object detection using a boosted cascade of simple features. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2001. 2001. p. I–I.

19. haar.png (334×203) [Internet]. [cited 2020 Apr 6]. Available from: https://docs.opencv.org/3.4/haar.png

20. Rodriguez P, Cucurull G, Gonfaus J, Roca X, Gonzàlez J. Age and Gender Recognition in the Wild with Deep Attention. Pattern Recognit. 2017 Jul 1;

21. Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09. 2009.

22. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. ArXiv170404861 Cs [Internet]. 2017 Apr 16 [cited 2020 Mar 23]; Available from: http://arxiv.org/abs/1704.04861

23. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. ArXiv180104381 Cs [Internet]. 2019 Mar 21 [cited 2020 Mar 23]; Available from: http://arxiv.org/abs/1801.04381

24. pipeline-flow.png (1733×360) [Internet]. [cited 2020 Mar 20]. Available from: https://docs.microsoft.com/en-us/azure/machine-learning/media/concept-ml-pipelines/pipeline-flow.png

25. Benchmarking Machine Learning on the New Raspberry Pi 4, Model B [Internet]. Hackster.io. [cited 2020 Apr 5]. Available from: https://www.hackster.io/news/benchmarking-machine-learning-on-the-new-raspberry-pi-4-model-b-88db9304ce4