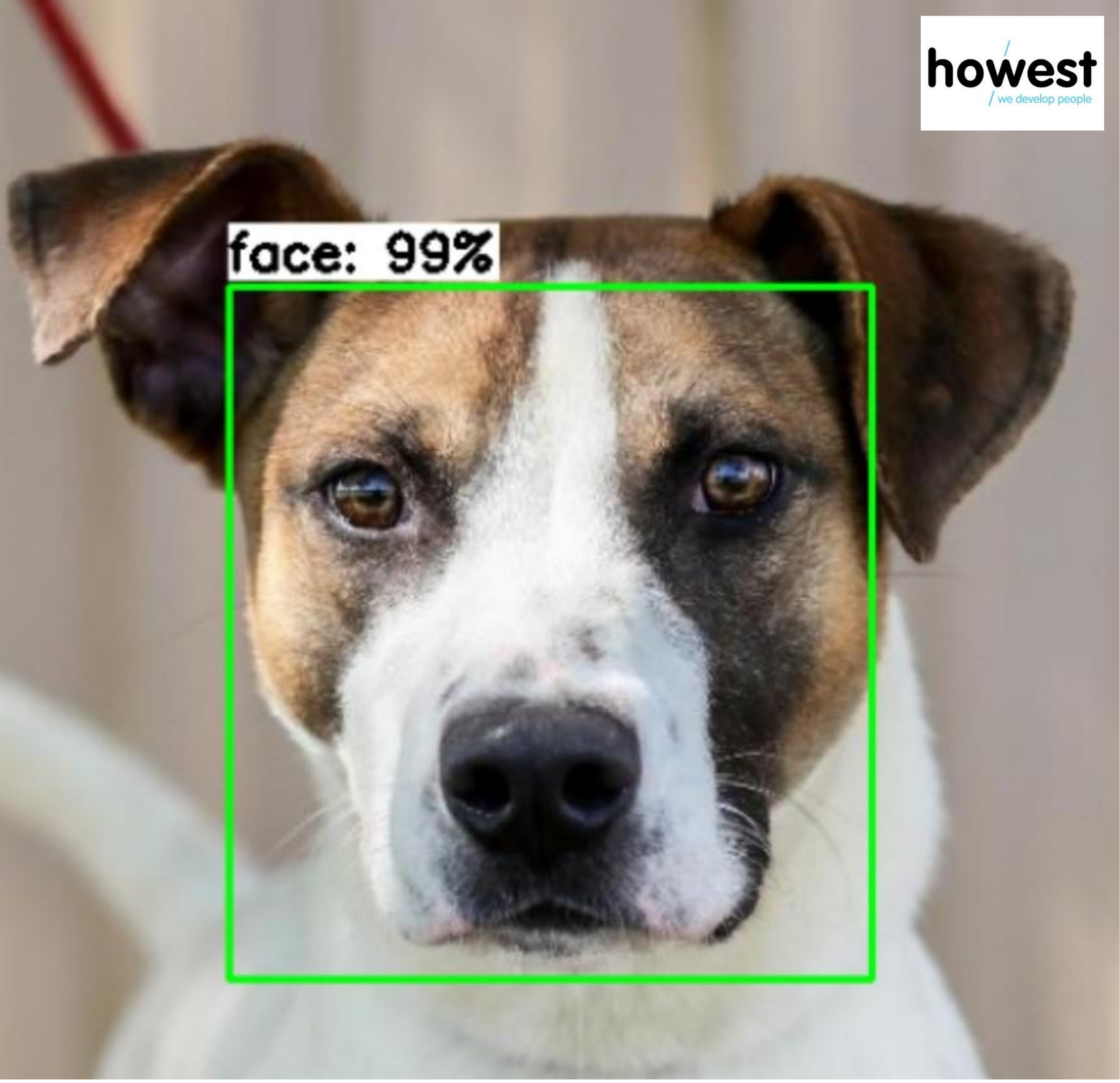


How can you automate pet doors using an embedded facial recognition model?

Internal promotor: Marie Dewitte
External promotor: Dr. Belhaoua Abdelkrim

RESEARCH QUESTION BY

BENJAMIN VIERSTAETE

FOR OBTAINING A BACHELOR'S DEGREE IN

MULTIMEDIA & CREATIVE TECHNOLOGIES

Howest | 2022-2023

Preface

This Bachelor's thesis is the culmination of my studies in New Media & Communication Technology Bachelor degree at Howest University West Flanders in Kortrijk. The focus of this thesis is on answering a specific research question: "How can pet doors be automated using an embedded facial recognition model?"

The thesis initially discusses my research and technical demonstration carried out in the Researchproject module. Going over my findings and how the proof of concept seen in this thesis was build. Followed with a reflection on the results of this project, ending with advice for anyone trying a similar project and a conclusion answering the question mentioned above.

For this research I developed a proof of concept prototype consisting of a pet door with two small micro servo motors serving as a lock. Using a Raspberry Pi Camera Model V2 connected to a Raspberry Pi 3+ to detect and recognize animal faces.

I would like to thank Marie Dewitte for the support in the creation of my research project and this thesis. I would also like to thank all the teachers at MCT Howest for the knowledge shared over the years. Furthermore I would like to thank all the experts that helped me with suggestions and feedback. A special thanks to my external promoter Dr. Abdelkrim Belhaoua a researcher at Barco labs for agreeing to read this paper and grade it.

Finally I would like to thank my girlfriend for not only proofreading this thesis but for coming up with the research question.

Benjamin Vierstraete 29-05-2023

Abstract

This paper addresses the question of automating pet doors using an embedded facial recognition model. This technology could help to improve pet safety by preventing unwanted animals or intruders from entering the home using the pet door.

A proof of concept pet door was built using a Raspberry Pi to detect and recognize pet faces based on the animals it was trained on. Several models were researched and options like embedded training were tested and evaluated.

Speed and performance were important considerations given the use of a Raspberry Pi, furthermore using deep learning for computer vision posed significant challenges on the embedded device. While the project did not quite meet the necessary speed requirements for real-time detection, it was able to accurately control a pet door based on pet recognition, demonstrating the feasibility of such a solution with additional computational power and further optimization.

Using the suggestions and learnings provided in this paper, an improved version can be made. These suggestions include quantisation, pruning, model fusion and training on lower resolution. These should prove useful in lowering inference time, providing a more real-time recognition. In addition using a Jetson Nano instead of a Pi could also lead to improved inference time.

Finally, there are several alternative approaches mentioned. One option is to simplify the task by using image classification instead of the more complex techniques used like facial recognition and object detection. Another alternative is to focus on movement detection instead of continuously detecting each frame. Additionally, other techniques such as FOMO and SIFT are explored as alternatives.

Considering these alternatives can help determine the most suitable approach for this specific constrained task.

Table of contents

1 Introduction ... 9
2 Research ... 10
2.1 What facial recognition models are available? ... 10
2.1.1 FaceNet ... 10
2.2 What detection models can be used for detecting the faces? ... 11
2.2.1 YOLOv7 ... 11
2.2.2 SSD-MobileNetV2 ... 12
2.2.3 Conclusion ... 15
2.3 What facial recognition technology is best scalable to pets? ... 16
2.3.1 Transfer learning ... 16
2.3.2 Custom model ... 16
2.4 What is the minimal hardware needed for embedded real-time predictions? ... 19
2.3.1 Think embedded ... 19
2.3.3 Hardware ... 21
2.5 Is it possible to train a model embedded? ... 23
2.6 Is it practical to train a model embedded? ... 23
3 Technical Research ... 24
3.1 Introduction ... 24
3.2 Hardware ... 24
3.3 Face detection ... 25
3.3.1 Data ... 25
3.3.2 Model ... 25
3.3.3 Training ... 26
3.3.4 Scoring ... 26
3.4 Face recognition ... 27
3.4.1 Data ... 27
3.4.2 Model ... 28
3.4.3 Triplet generation ... 28
3.4.4 Training and scoring ... 29
3.4.5 Databases ... 29
3.5 Using camera Raspberry Pi ... 29
3.6 Deploying models ... 30
3.7 Auto Preprocessing ... 31
3.8 Result ... 31
3.9 What is the minimum camera resolution needed? ... 33
4 Reflection ... 35
4.1 Strengths and weaknesses. ... 35
4.1.1 Strengths ... 35
4.1.2 Weaknesses ... 35
4.2 Is it useable? ... 35
4.2.1 project ... 35

4.2.2 Methods 35
4.2.2 Implementation hurdles 36
4.3 Alternatives and suggestions 36
4.3.1 Non-technical 36
4.3.1 Technical 37
4.4 Is there any social, economic or socio-economic value? 41
4.5 Further research 42
5 Advice 43
5.1 Should the same technologies be used? 43
5.2 Recommendations 43
5.2.1 Optimizing interference time 43
5.2.2 Optimizing performance 43
5.3 Practical advice implementation 45
5.4 Use the auto pre-processor tool 45
5.5 Collect diverse data 45
5.6 Further advice 45
6 Conclusion 47
7 References 48
8 Attachments 52
8.1 Report IBM 52
8.2 Report SupportSquare 53
8.3 Installation Manual 54
8.4 User manual 59

Table of figures

Figure 1: Triplet loss [5] 10
Figure 2: Showcase three components. [8] 11
Figure 3: MobileNetV2 Architecture [12] 13
Figure 4: normal convolution on 12x12 image with 3 channels [13] 13
Figure 5: Depthwise convolution on 12x12 image with 3 channels [13] 14
Figure 6: Pointwise convolution on output Depth wise convolution [13] 14
Figure 7: MobileNet base with SSD head [15] 15
Figure 8: Example of facial differences in images from training data 16
Figure 9: ResNet concept [21] 17
Figure 10: DenseNet concept [21] 17
Figure 11: Mutual information scores [24] 18
Figure 12: Map score [24] 18
Figure 13: comparing quantized and floating point model [33] 20
Figure 14: Example of pruning reducing complexity model [34] 20
Figure 15: comparison raspberry pi versions with and without USB accelerators [35] 21
Figure 16: Performance and temperature depending on clock rate [37] 22
Figure 17: Interference time on Raspberry pi model 3: without Coral(left), with Coral(right) [42] 23
Figure 18: test subjects 24
Figure 19: visualisation wiring 25
Figure 20: mAP/AP score of MobileNetV2 model on 135 validation images 26
Figure 21: predicted bounding boxes vs true bounding boxes 27
Figure 22: Data for each class 28
Figure 23: max convergence 29
Figure 24: example of predictions using Euclidean distance 29
Figure 25: Detection and recognition flow 30
Figure 26: YOLOv7 output of cropped faces 31
Figure 27: Snapshot of Raspberry Pi prediction feed 32
Figure 28: example of proof of concept in use 32
Figure 29: Results Face Recognition for a resolution of 32x32. 33
Figure 30: Classification report 32x32 resolution 33
Figure 31: Results Face Recognition for a resolution of 16x16. 34
Figure 32: Classification report 16x16 resolution 34
Figure 33: Ivan Pavlov's research setup [52] 37
Figure 34: result background subtraction [53]. 38
Figure 35: Heatmap produced by the output layer [54] 38
Figure 36: Centroids predicted to count bees. [54] 39
Figure 37: potential key point compared with neighbouring pixels [57] 40
Figure 38: Profiler results visualised using snakeviz 41
Figure 39: Profiler results after reducing detection from every frame to every 25 41
Figure 40: ROC curve with and without alignment. using FaceNet [61] 44
Figure 41: Face Alignment pipeline [62] 44

List of Abbreviations

AP	Average precision
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
DoG	Difference of Gaussian
E-ELAN	Extended Efficient Layer Aggregation Network
FLOPS	Floating-point Operations Per Second
FOMO	Faster Objects, More Objects
FPN	Feature Pyramid Network
FPS	Frames Per Second
GB	Giga Byte
GDPR	General Data Protection Regulation
GHz	Gigahertz
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
JSON	JavaScript Object Notation
Mhz	Megahertz
POC	Proof Of Concept
RAM	Random Access Memory
RGB	Red Green Blue
ReLU	Rectified Linear Unit
SIFT	Scale-Invariant Feature Transform
SSD	Single Shot Detector
SURF	Speeded Up Robust Features
YOLO	You Only Look Once
mAP	Mean Average Precision

Glossary

Keras	Keras is a high-level neural networks library, written in Python, that simplifies deep learning model development.
Loss	Loss is a metric used to quantify the inconsistency between predicted and actual values in machine learning models.
Opencv	OpenCV is a computer vision library that provides tools and functions for image and video processing tasks.
Pytorch	PyTorch is a popular open-source deep learning framework that provides tensor computation and dynamic neural network building capabilities.
Tensorflow	TensorFlow is a widely-used open-source machine learning framework that enables the development and training of deep learning models.
Transfer-learning	Transfer learning is a machine learning technique where knowledge gained from pre-trained models is applied to a different but related task to improve performance and reduce training time.
Bounding-box	A bounding box is a rectangular region defined by its coordinates, used in object detection tasks to locate specific objects in an image or video.
Convergence	Convergence refers to the state in machine learning where a model's training process reaches a stable and optimal point, where further training does not significantly improve.
Convolutional-neural-network	Convolutional-neural-network is a deep learning architecture specifically designed for image processing and computer vision tasks.
Detection	Detection is identifying and locating objects or patterns within images or videos using computer vision techniques.
Edge-devices	Edge devices refer to computing devices that perform data processing and analytics locally.
Embedded	An embedded device is an independent device responsible for executing a particular task or set of tasks.
Embedding	Embedding refers to representing data, such as words or entities, as dense vectors in a lower-dimensional space.
Epoch	An epoch refers to a single iteration of training, where the entire dataset is passed through the

	model forward and backward to update its parameters.
Euclidean-distance	Euclidean distance is the straight-line distance between two points in Euclidean space, calculated using the Pythagorean theorem.
Feature-extraction	Feature extraction is the process of automatically capturing relevant information or patterns from raw data to create compact and representative feature representations.
Feature-map	A feature map refers to the output of a particular layer in a convolutional neural network (CNN) that contains the learned features extracted from the input data.
Gaussian	Gaussian refers to the Gaussian distribution or Gaussian function, which is a continuous probability distribution characterized by its bell-shaped curve.
Inference	Inference, in the context of machine learning, refers to the process of using a trained model to make predictions or draw conclusions from new, unseen data.
Kernel	Kernel refers to a small matrix or filter used to perform operations, such as convolutions, on input data
Layer	A layer refers to a component that processes input data or features in a structured manner. Each layer in a neural network performs specific computations.
Neuron	A neuron, also known as a node, is a fundamental unit in a neural network that processes and transmits information.
Parameters	Parameters are learned variables in machine learning models that determine their behavior and are optimized during training.
Recognition	Recognition refers to the process of identifying and classifying objects.
Weights	Weights are numerical values assigned to connections between neurons in a neural network, representing the strength of the connections and determining their contribution to the overall output.
Keras	Keras is a user-friendly, high-level neural networks library in Python that simplifies the development and training of deep learning models.

1 Introduction

Some people consider pet doors to be a security risk not worth taking. As a child, I heard myths of burglars sending children through pet doors so they could open the main door. While it is unclear if these myths are true, in some countries like the USA unwanted animals are the real threat. Having a pet door that only allows your own dog or cat to enter can prevent such risks. Collar-controlled pet doors are available on the market, but there are concerns about pets getting stuck with collars. To ensure safety for both pets and homes, a potential solution could be a door that uses facial recognition instead.

This thesis will answer the question if it is possible to automate a pet door using facial recognition while also providing a proof of concept. In this POC two models were used each for a different purpose. First a MobileNetv2 [1] model trained on new data using transfer learning. This was used to detect the faces. Secondly a ResNet model was trained using triplet loss to recognize specific dogs from these faces.

It will answer some underlying questions:

- What facial recognition technologies are available?
- What facial recognition technology is best scalable to pets?
- What is the minimal hardware needed for embedded real-time predictions?
- Is it possible to train a model embedded?
- Is it practical to train a model embedded?
- What is the minimum camera resolution needed?

The data used for training and validating the MobileNetv2 comes from two publicly available datasets: CatsVsDogs [2] and Animalfaces [3] supplemented with private pictures of my three dogs. For the ResNet model only images of my own dogs were used. The POC was developed through the following steps:

- Selecting the required hardware
- Collecting and processing data for face detection
- Training face detection models
- Collecting and processing data for face recognition
- Training face recognition models
- Writing Python code to access the Raspberry Pi camera
- Programming servomotors to open and close
- Integrating the trained models into the code

All code was written in Python, the models were trained using the TensorFlow and Keras libraries locally and on Google Colab. Both models were converted to TensorFlow lite format to run optimised on the Raspberry Pi.

This thesis consists of a few different sections, a section for research answering all the theoretical questions previously mentioned followed by a section going over my proof of concept. Then a detailed personal reflection where I ask experts ranging from data scientists to software developers on the topic. Ending with advice for real world applications and a conclusion to the thesis.

2 Research

2.1 What facial recognition models are available?

The process of face recognition usually consists of three main steps; detecting a face in an image, extracting features from the face and face recognition. In the case of this project the face recognition consist of identifying the pets face in a dataset of known faces. There are many models available with new ones consistently being published. Some examples are; Arcface, Deepface, VGGface, etc. In this research the focus is on FaceNet for its state of the art performance, community and the fact that it is open-source.

2.1.1 FaceNet

FaceNet was brought forward by Google in a paper called "FaceNet: A Unified Embedding for Face Recognition and Clustering" [4]. The model produces a vector embedding of 128 values. By predicting and comparing these to the embeddings of know faces in a dataset, faces can be recognised. To compare these, the embeddings are projected in a high-dimensional Euclidean space. Here the distance between points corresponds to a measure of face similarity. A lower distance means more chance of the two embeddings to belong to the same face.

FaceNet uses a 22 layer deep CNN trained using triplets of matching and non-matching faces. These triplets consist of three images; an anchor image, a positive image and a negative image. The anchor and positive are selected from the same class, while the negative is selected from a different class. The goal is to minimise the Euclidean distance between the images from the same class while maximizing the distance from the other class. To encourage this a loss function was introduced called Triplet loss.

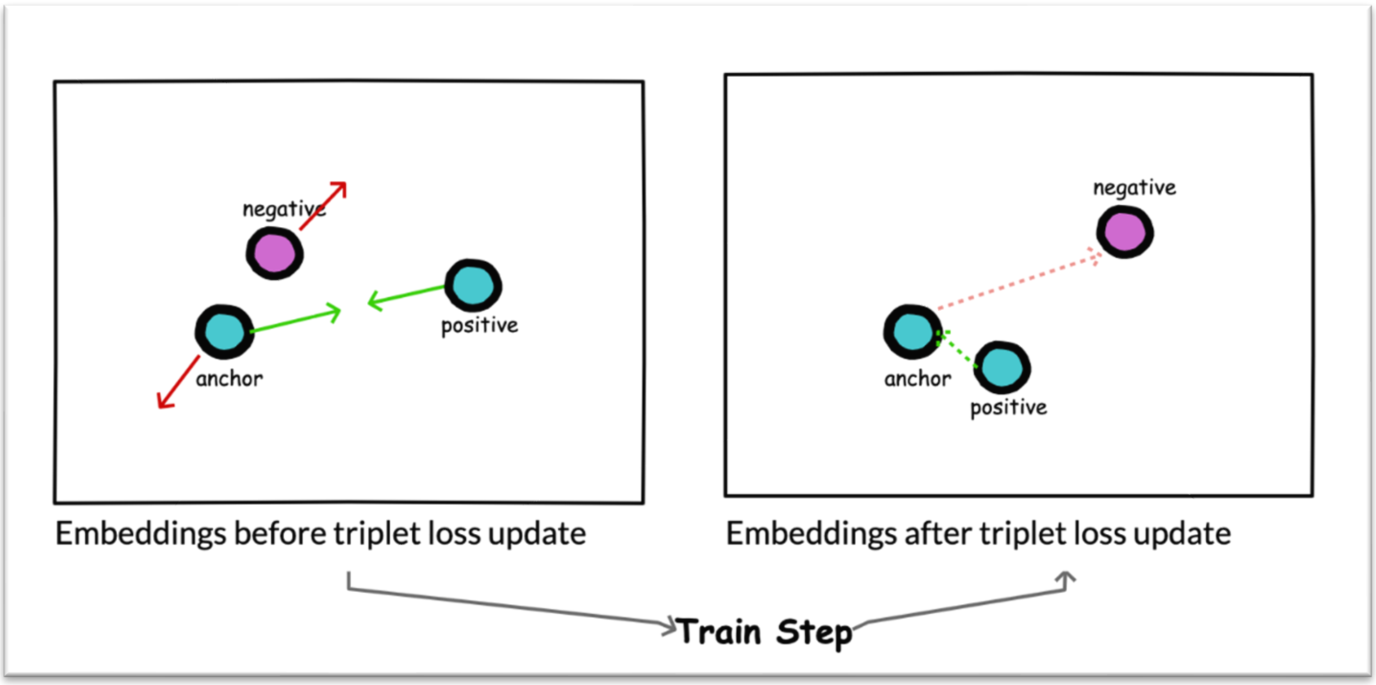

negative
positive
anchor
negative
anchor
positive
Embeddings before triplet loss update
Embeddings after triplet loss update
Train Step

Figure 1: Triplet loss [5]

As outlined in the FaceNet paper, mislabelled or poorly imaged faces can dominate the positives and negatives. To combat this, triplets are generated using a method called online triplet mining. Online means the triplets are selected during each training iteration from batches in the training data. On the contrary offline means selecting all triplets at the start of training from the whole train dataset. Online minimises the chances of mislabelled data skewing training, while also requiring less computation and memory compared to offline triplet mining.

To maximise the convergence, Google proposed the selection of hard triplets. The idea is to select triplets that are challenging for the model to learn from. They contain data points that are close in the embedding space and may result in smaller distances, making it harder for the model to distinguish between faces. By selecting hard triplets, the model is forced to learn more discriminative and meaningful embeddings. Leading to better capturing of subtle differences between similar and dissimilar data points. This can lead to faster convergence and improved performance of the model.

2.2 What detection models can be used for detecting the faces?

Most facial recognition pipelines first detect the faces before starting classification or recognition. For this purpose object detectors are commonly used. These models are used for detecting faces which in turn are used as input image for the facial recognition model.

2.2.1 YOLOv7

YOLOv7, being launched in 2022, is the cutting edge of object detection and considered one of the most accurate object detector [6]. YOLOv7 presents methods to enhance model performance without increasing training costs. For example re-parametrization. This is a technique used to modify the parameters of a trained model to improve its performance and can be done at model or module level. Model re-parameterization can be achieved by averaging weights from multiple models trained with the same settings or from different epochs. Modular re-parameterization divides the training process into modules, with outputs ensembled for the final model. [7]

The YOLO architecture is a fully connected neural network consisting of three main components:

- Backbone
- Head
- Neck

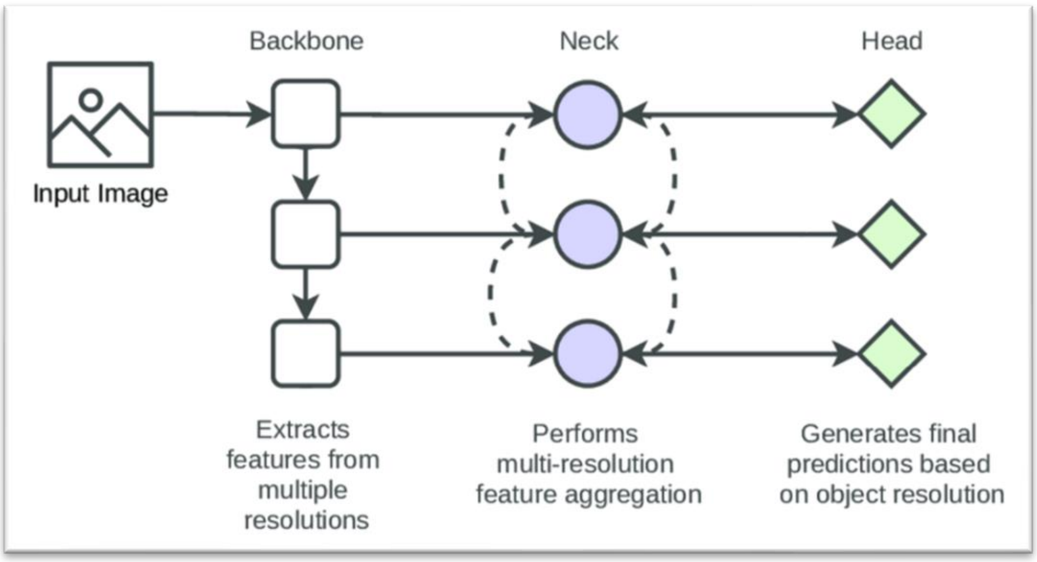

Backbone
Neck
Head
Input Image
Extracts features from multiple resolutions
Performs multi-resolution feature aggregation
Generates final predictions based on object resolution

Figure 2: Showcase three components. [8]

2.2.1.1 Backbone

The backbone of a object detection model is used for feature extraction. In YOLO this backbone consists of the Extended Efficient Layer Aggregation Network. E-ELAN is a technique that enhances the performance of object detection models without changing their fundamental structure. Instead of modifying the way information flows through the model, E-ELAN uses group convolution to increase the number of features being processed and combines them in a way that improves the model's ability to learn important features. This approach optimizes the use of parameters and calculations, ultimately leading to better detection accuracy. [9]

2.2.1.2 Neck

In object detection, a neck connects the feature extraction backbone to the object detection head. The neck is a set of intermediate layers that fuses the features extracted by the backbone in a way that is useful for the head. It is designed to perform operations such as feature aggregation, feature transformation and feature selection.

2.2.1.3 Head

The head performs a prediction using the extracted features. In the case of YOLOv7 it predicts both bounding box coordinates and a confidence score for the classes. YOLOv7 consists of two heads, a lead head for final output and an auxiliary head for training in middle layers. The model also features a Label Assigner mechanism that generates soft labels by considering network predictions and ground truth, rather than relying solely on hard labels based on given rules. The soft labels are optimized to consider the quality and distribution of prediction output and improve the deep network training process. [10]

2.2.1.4 YOLOv7-tiny

As the name suggests this is a version of YOLOv7 designed specifically for the edge, which means it is optimized for running on mobile computing devices or edge devices. Unlike other YOLO versions, YOLOv7-tiny uses leaky ReLU as the activation function [10]. Leaky ReLU prevents the "dying ReLU" issue, where neurons with negative biases become inactive. This is done by giving a small slope to the negative values, allowing all neurons to contribute to the network's output therefor increasing performance. [11]

2.2.2 SSD-MobileNetV2

The MobileNetV2 architecture is a lightweight neural network that is specifically designed for use in embedded devices [1]. In contrast to YoloV7, MobileNetV2 is optimized for efficiency and is well-suited for applications with limited computational resources such as a Raspberry Pi or Mobile phones.

The SSD-MobileNetv2 model consists of two distinct parts. The first part is the MobileNetv2 model, which is trained for image classification. The second part is the SSD, which replaces the classification component of the model with a mechanism for outputting bounding boxes making it well suited for object detection.

2.2.2.1 MobileNet-v2

MobileNet is designed to be efficient in terms of memory and computation, making it well-suited for use on devices with limited resources like a Raspberry Pi. In this model MobileNet is used as the backbone to extract features. Meaning the fully connected layer at the end normally used for classification is removed. This backbone extracts features, but leaves the spatial structure of the image as output.

The advantages of MobileNetV2 are a smaller number of parameters and lower computational requirements then most CNN architectures. Additionally, MobileNet uses depthwise separable convolutions which can be implemented more efficiently on mobile and embedded devices. These depthwise seperable convolutions are the middle layer of the Residual Blocks used in MobilenetV2. The first layer is the expansion layer, a 1x1 convolution layer with the purpose to expand the number of channels in the data. After this comes depthwise convolution and finally the Bottleneck layer, this layer makes the number of channels smaller. As the name suggest this block makes use of a residual connection to help with the flow of gradients through the network. [12]

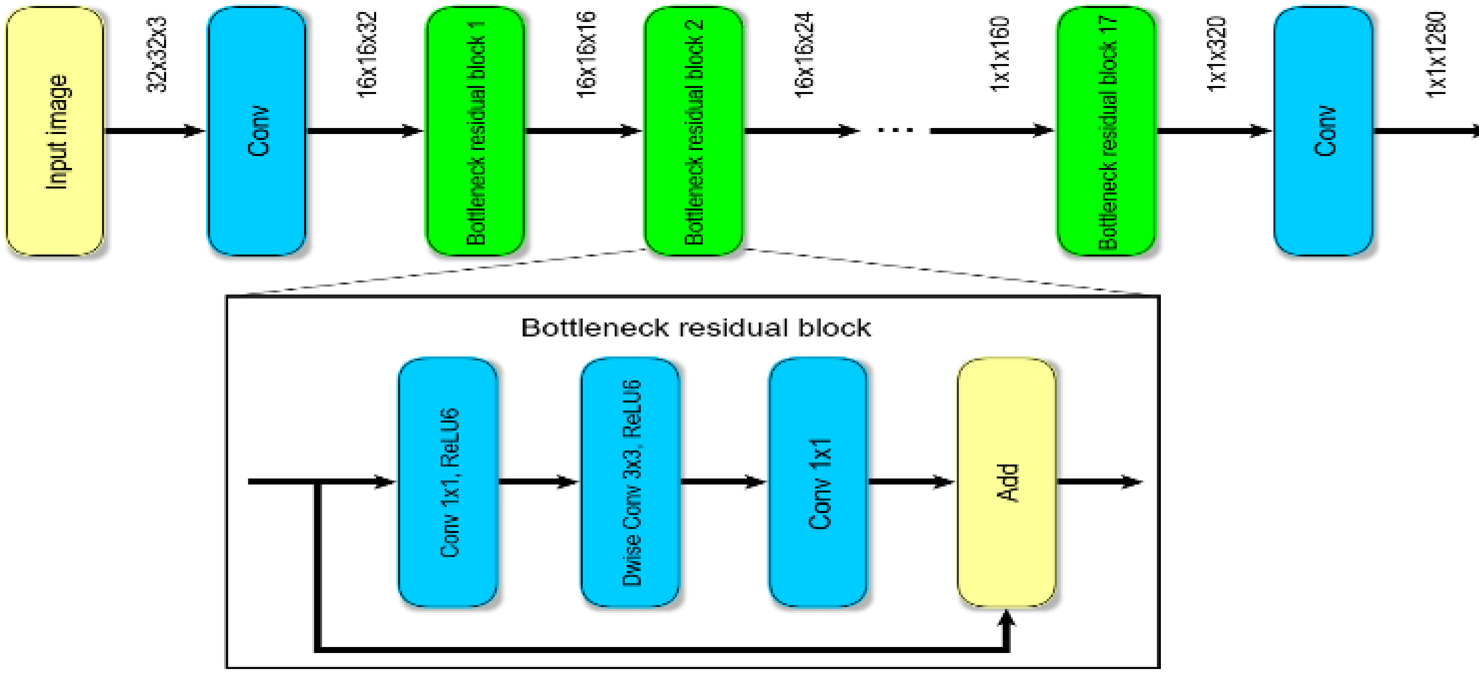


Figure 3: MobileNetV2 Architecture [12]

Depthwise separable convolutions consist of two main components, depthwise convolution and pointwise convolution. As seen in the example below a normal convolution is done on a RGB image of size 12x12x3 with a 5x5 kernel. The results is an 8x8 output image in width and height. Since the image has three channels, the convolution needs to have three channels as well. This means a total of 75 multiplications (5x5x3=75) need to be done every time the kernel moves. The resulting output image has dimensions of 8x8x1.

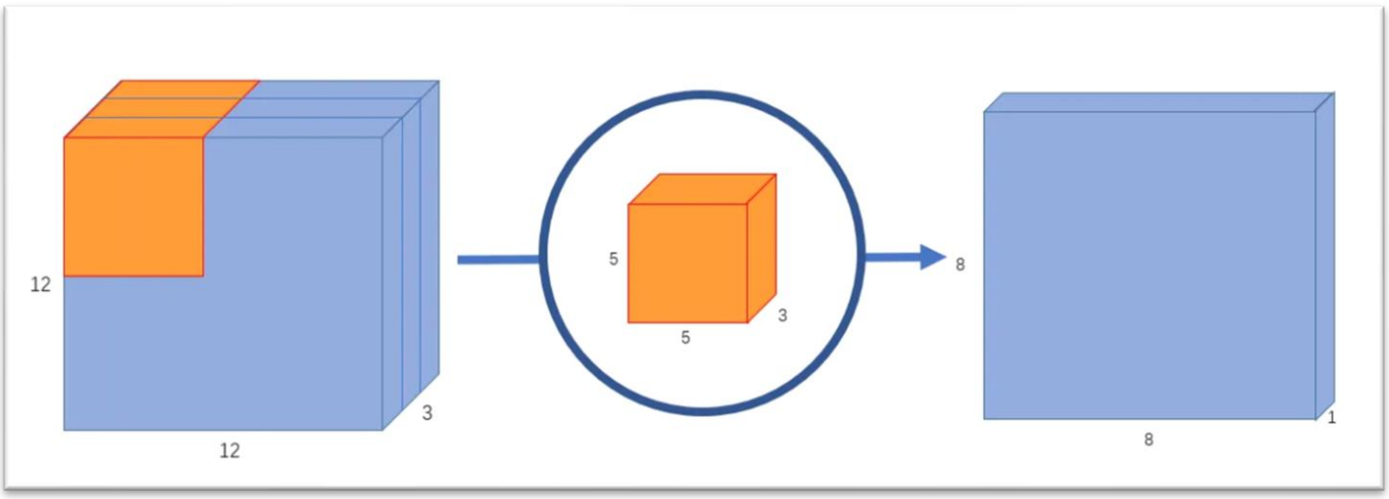


Figure 4: normal convolution on 12x12 image with 3 channels [13]

In Depthwise convolution each kernel iterates over only one channel at a time, rather than processing all channels together as in traditional convolution. Meaning that for a 5x5 kernel only 25 multiplications are needed, resulting in a total of 25 multiplications per channel. The output for each channel is a separate 8x8 feature map which then get stacked together as seen in the image below.

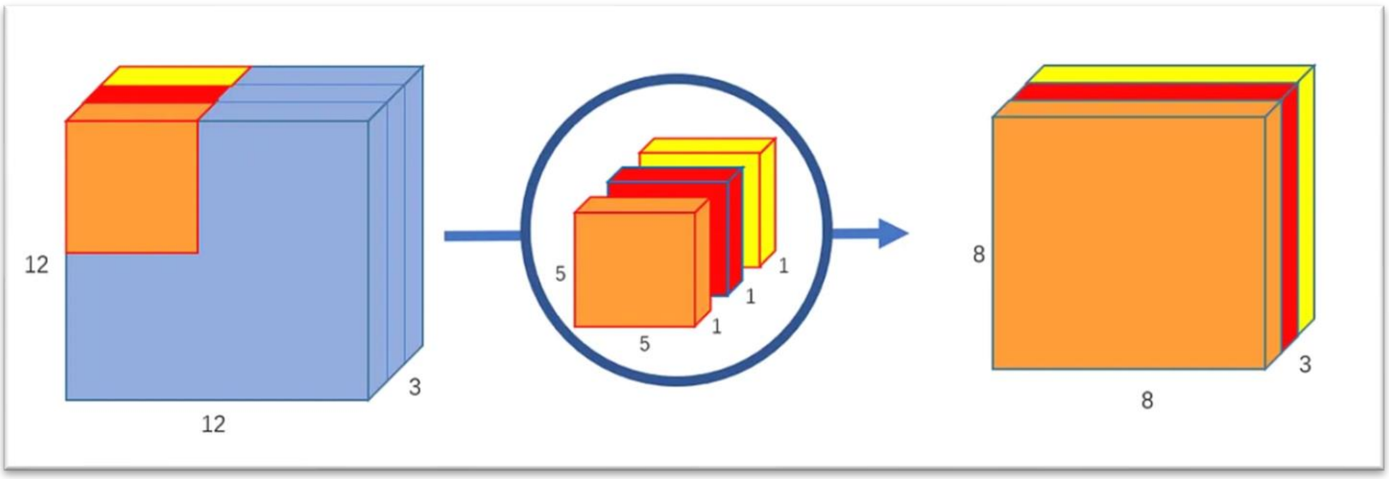


Figure 5: Depthwise convolution on 12x12 image with 3 channels [13]

To achieve the same output as normal convolution, the channels must be reduced to one. This is were pointwise convolution comes in. Pointwise convolution uses a 1x1 kernel with as many channels as the input so in the case of the example this is three channels. This 1x1x3 kernel goes through every single point using a set of filters to linearly combine the feature maps. [13]

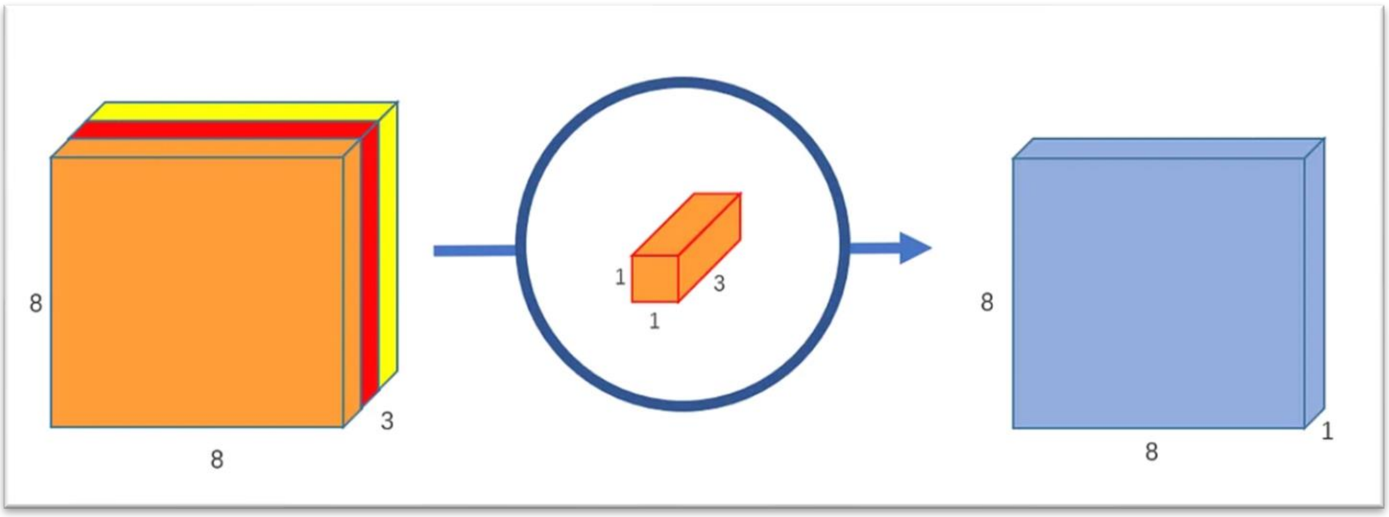


Figure 6: Pointwise convolution on output Depth wise convolution [13]

2.2.2.2 Single Shot Detector

The single shot detector head is a set of one or more convolutional layers that predict bounding boxes instead of classifying. It employs a set of predefined bounding boxes, referred to as anchor boxes to predict the location and class of objects in an input image. The SSD head combines the predictions from these anchor boxes with a non-maximum suppression algorithm to produce the final detection results. This method of detection takes the extracted features and uses them to identify the location of objects in the image.

The SSD works by breaking down the input image into a grid of cells and running object detection on each cell. If no objects are detected, the cell is classified as the background class. Then using default bounding boxes, called anchor boxes, to predict the location and class of objects within each cell. During training, the algorithm matches the appropriate anchor box with the bounding boxes of each ground truth object within an image. The anchor box with the highest degree of overlap with an object is used to predict that object's class and location. [14]

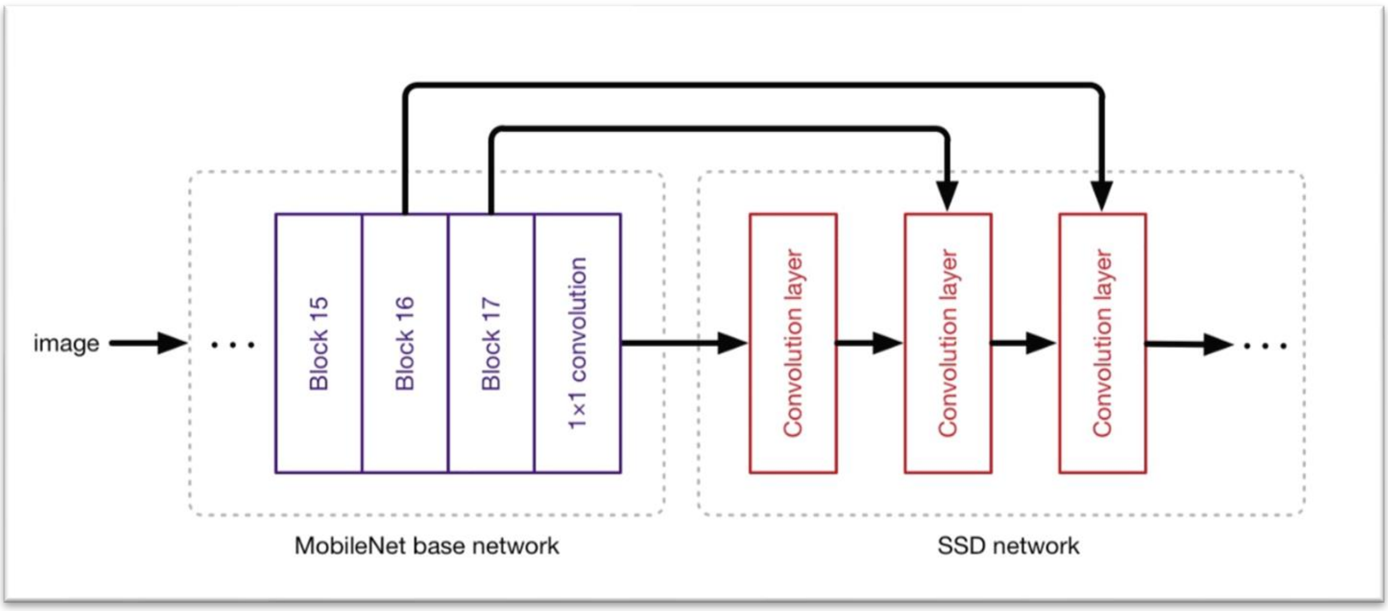


Figure 7: MobileNet base with SSD head [15]

2.2.2.3 Neck

Some SSD-MobileNetv2 models use a neck to connect the backbone with the SSD head. An example of this is FPN-Lite. This is a lightweight version of the Feature Pyramid Network. This works by combining features from multiple scales, providing the SSD with additional contextual information that can help improve the accuracy of the object detections.

2.2.3 Conclusion

When comparing both object detectors, it is clear that there is a trade-off between the higher precision of YOLO and speed of MobileNet. When selecting a model for embedded devices without GPU, MobileNet comes on top with a smaller model size compared to the YOLOv7-tiny model and depthwise convolution layers designed for faster interference. The convenience of it being a TensorFlow model should also be considered. This makes the transition to TensorFlow Lite much easier. For unrestricted devices YoloV7 is the clear winner. MobileNet’s depthwise convolutions are not directly supported in GPU firmware. Meaning when a GPU is available YOLO would be the better choice. [16][17][18]

	YOLOv7-tiny	**YOLOv7**	**MobileNetV2**
Framework	Pytorch	Pytorch	TensorFlow
Parameters	6.2M	36.9M	4.3M
COCO mAP	38.7%	51.4 %	22.1%
Designed for	GPU	GPU	CPU

2.3 What facial recognition technology is best scalable to pets?

The main difference in facial recognition between humans and pets is the different unique characteristics. The facial features of humans and pets are distinctively different. Human faces have a consistent structure with mostly similar facial features across individuals and ethnicities. Pet faces on the other hand exhibit a wider range of shapes, sizes and structures depending on the species, breed and individual characteristics. Even from just one dog to the other there can be a major difference in facial features.

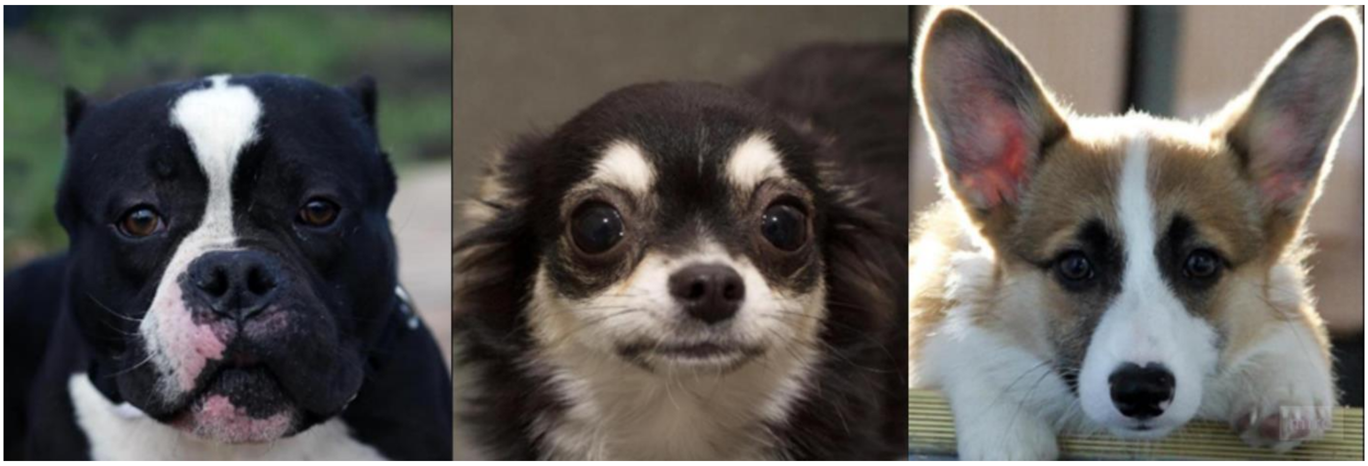


Figure 8: Example of facial differences in images from training data

2.3.1 Transfer learning

The first option is using transfer learning to train pre-trained models on the new pet data. Hereby making use of the features already learned from large datasets of human faces. This way there is no need to train a model from scratch, potentially leading to faster convergence.

In the context of facial recognition on pets, feature extraction is not that useful. The features learned from human data don't always translate well to pets. Fine-tuning by retraining all layers using the pre-trained weights as starting points could proof useful as pets also have eyes, a nose and a mouth. This might speed up the process of finding good features in the new data. However fine tuning is significantly slower then feature extraction as all layers need to be retrained.

2.3.2 Custom model

As there are no public facial recognition models pretrained on pets, it could be a good idea to train a custom model using techniques found in popular facial recognition models. This section will compare model architectures, loss functions and techniques to train a custom model for pets.

2.3.2.1 Architecture

Except for the older methods most facial recognition models use a variant of a convolutional neural network. These are composed of multiple convolution layers each of which detects more complex features. As pet faces have a big variability in features and the appearance of their faces can vary greatly within the same breed, it is advised to use architectures with good feature extraction like ResNet or DenseNet. These architectures ease gradient flow and feature reuse by better preserving important features needed for good predictions.

ResNet was introduced in a paper from 2015 called "Deep Residual Learning for Image Recognition" [19]. The key innovation from this paper is the introduction of residual connections. In a standard neural network the output of one layer is passed as input to the next layer in a sequence. In ResNet however the output of some layers is passed not only to the next layer but also to a layer further down the chain bypassing intermediate layers. This shortcut connection is done via element-wise addition, allowing the flow of information to be retained and reused throughout the network rather than being lost or distorted as it passes through multiple layers [19]. This also helps alleviate the vanishing gradients problem as a network gets more and more layers the gradients of the parameters with respect to the loss become very small, making it difficult for the network to learn. Residual networks avoid this problem by using these shortcut connections to carry gradients throughout the extent of deep networks. [20]

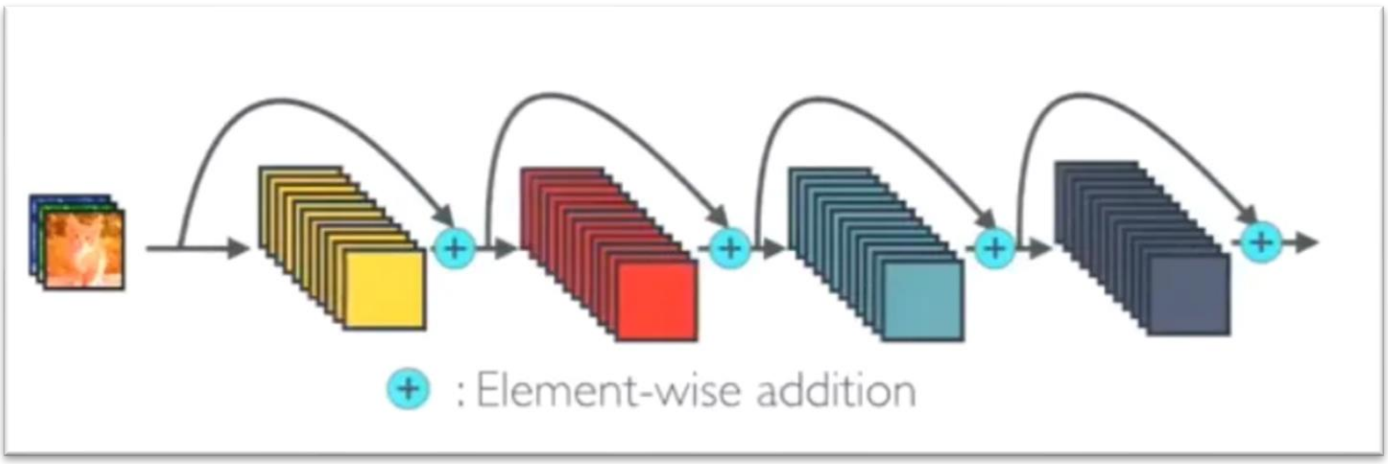


Figure 9: ResNet concept [21]

DenseNets where introduced in 2018 to further improve the information flow between layers [12]. Similar to ResNet they use residual connections, the difference is how they connect layers and leverage residual connections. In contrast to ResNets, they don’t combine features through summation (element-wise addition) instead they combine features by concatenating them.

As seen in the example image below, in DenseNet each layer obtains input from all preceding layers, thereby getting all prior collective knowledge [21].

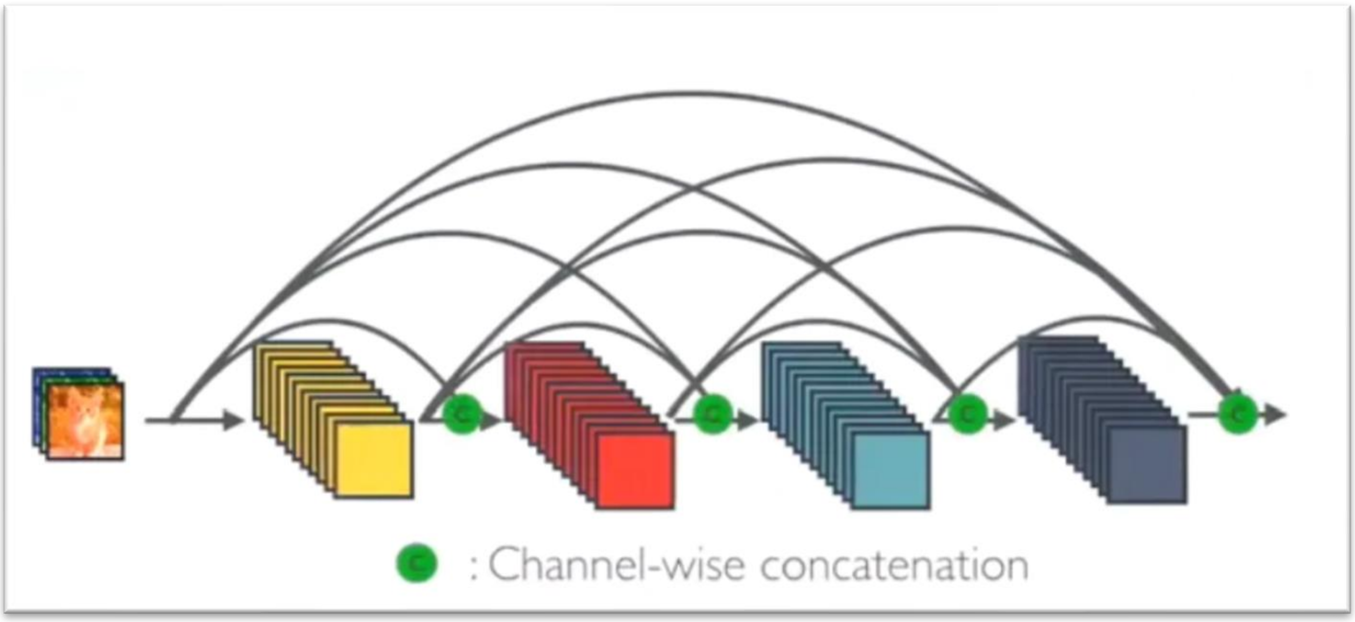


Figure 10: DenseNet concept [21]

The DenseNet architecture offers many advantages. The first of which is the ability to reuse feature maps through its dense connections. Hereby reducing interdependence between layers by reusing feature maps from different layers, allowing for compact and differentiated input features. DenseNet also addresses the aforementioned vanishing gradients problem. Furthermore by using features from all layers DenseNet achieves better performance and model robustness on standard datasets [13]. In addition DenseNet also requires fewer parameters than traditional convolutional networks, as there is no need to relearn redundant feature-maps [22].

One disadvantage to the concatenating of feature maps is that it results in redundant data replication. As the network gets deeper, the model parameters increase linearly, leading to significant computation and memory overhead during training [23].

2.3.2.2 Loss function

Loss Function is one of the most important components of a machine learning model. This section will go over the advantages and disadvantages of Triplet loss and ArcFace loss.

A comparison of both loss functions was done using a ResNet model on the TinyImageNet dataset [24]. To mine the triplets needed for Triplet loss, MultiSimilarityMiner was used from the PyTorch Metric Learning library. [25] The model was trained on a 2080ti GPU for 150 epochs on 100.000 images with a batchsize of 64.

Triplet loss scored better in Adjusted Mutual Information and Normalized Mutual Information meaning that the embeddings obtained from the Triplet loss provided a more accurate representation of the similarity between two clusters.

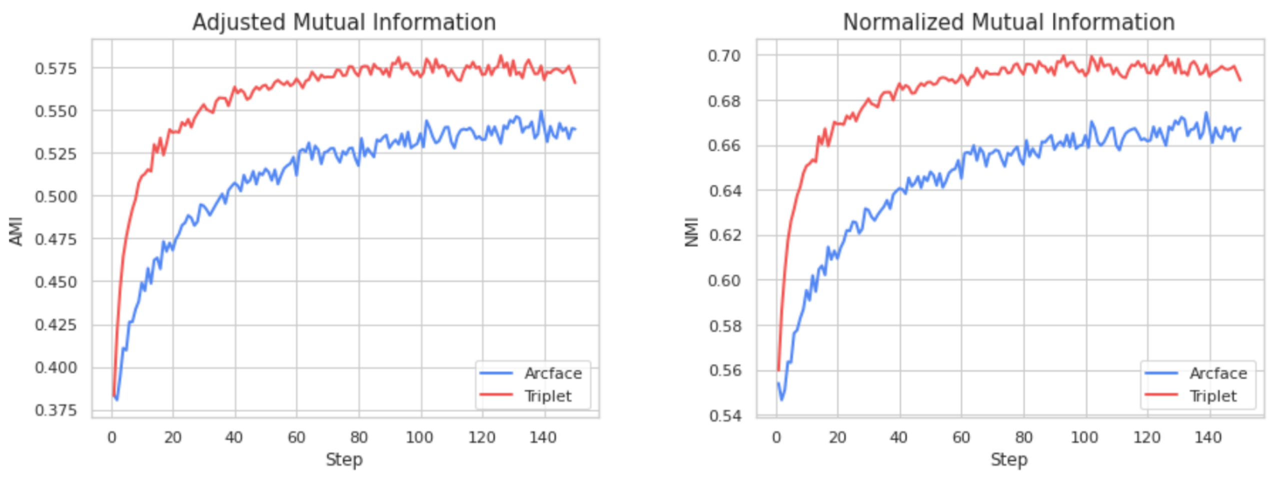


Figure 11: Mutual information scores [24]

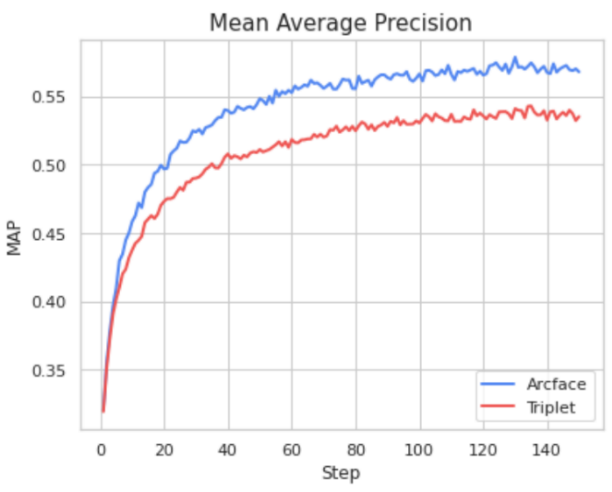


Figure 12: Map score [24]

On the other hand Arcface performed better in precision. This is an important metric in facial recognition tasks as it measures the accuracy of correctly identifying and verifying individuals based on their facial features. Both mAP and Mutual information are useful evaluation metrics assessing different aspects of the models performance. mAP focuses on the accuracy of object detection and image retrieval, while Mutual information assesses the quality of clustering.

A higher Mutual information can however be useful for creating embedding databases used for comparing with future predictions, A model with better clustering performance, means that the embeddings of individuals in the database are more accurately grouped together based on their similarity. This might lead to more reliable grouping and storage of facial embeddings.

2.3.2.3 Conclusion

Facial recognition techniques with high performance on humans will most likely translate in high performance on animals, the only difference is that for animals, no pretrained models are available. Just like humans, pets have distinctive facial features that a model can learn and use to tell faces apart. Furthermore facial recognition on animals is no new concept, many papers have proven the ability to translate this problem to animals. Some examples are; lemurs [26], bears [27], chimpanzees [28] and cows [29]. In all papers that used a deep neural network a custom model was build using CNN architectures some of which used ResNet ore DenseNet.

2.4 What is the minimal hardware needed for embedded real-time predictions?

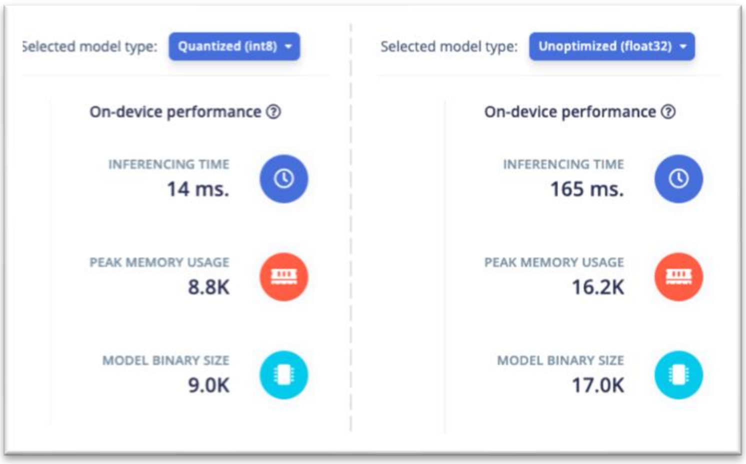
To answer this question, real-time must be defined in the context of this paper. Real-time can mean many things to many people, in this paper real-time is defined as “computer vision at rates sufficiently high to effect practical visually-guided control or decision making”. [30] meaning the predictions must be adequate speed for the pet door to react and open. In most cases this means several frames per second.

2.3.1 Think embedded

Before selecting hardware it is recommended to think embedded, meaning selecting the right lightweight network, by prioritizing lower computational needs instead of accuracy. [31]

Optimizing this model as much as possible is recommended, quantization is an effective method to reduce performance requirements with minimal accuracy decrease. Processors that can support dynamic quantization and efficiently leverage other techniques such as sparsity (limiting the number of non-zero weights) can achieve significant performance improvements while maintaining high accuracy. [31] Quantization is a method used to store tensors at lower bit widths. In a quantized model, some or all of the operations are executed using reduced precision typically in the form of integer values instead of full precision floating point values. This results in a more compact model and enhanced interference time. [32]

As seen in the image below, quantization from 32bit floating point values to 8bit integer values has a huge impact on interference time. Using half the memory the quantized model achieves 10 times lower interference. [33]


Selected model type: Quantized (int8)

On-device performance

INFERENCING TIME 14 ms.

PEAK MEMORY USAGE 8.8K

MODEL BINARY SIZE 9.0K

Selected model type: Unoptimized (float32)

On-device performance

INFERENCING TIME 165 ms.

PEAK MEMORY USAGE 16.2K

MODEL BINARY SIZE 17.0K

Figure 13: comparing quantized and floating point model [33]

Another option commonly used for embedded devices is pruning. This is the practice of removing unnecessary weights or connections to lower the model size therefore making it faster. There are several ways to prune a model:

- Weight pruning by removing individual weights from the model that are not contributing to the model's output.
- Neuron pruning by removing entire neurons from the model if they are not significant to the model's output.
- Structural pruning involves removing entire layers or connections from the model.

These can also be done in conjunction improving speed by lowering the size of the model. Pruning has the extra benefit of reducing overfitting, similar to dropout by removing connections the model is less likely to memorize training data. [34]

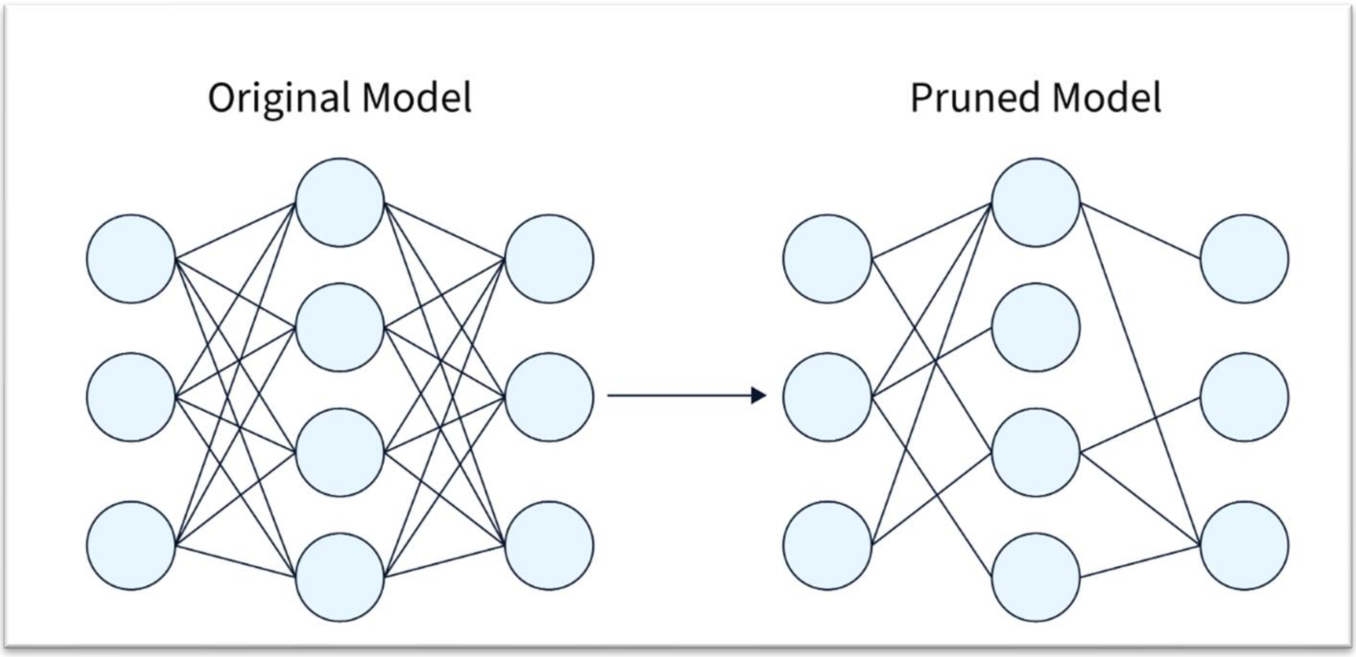


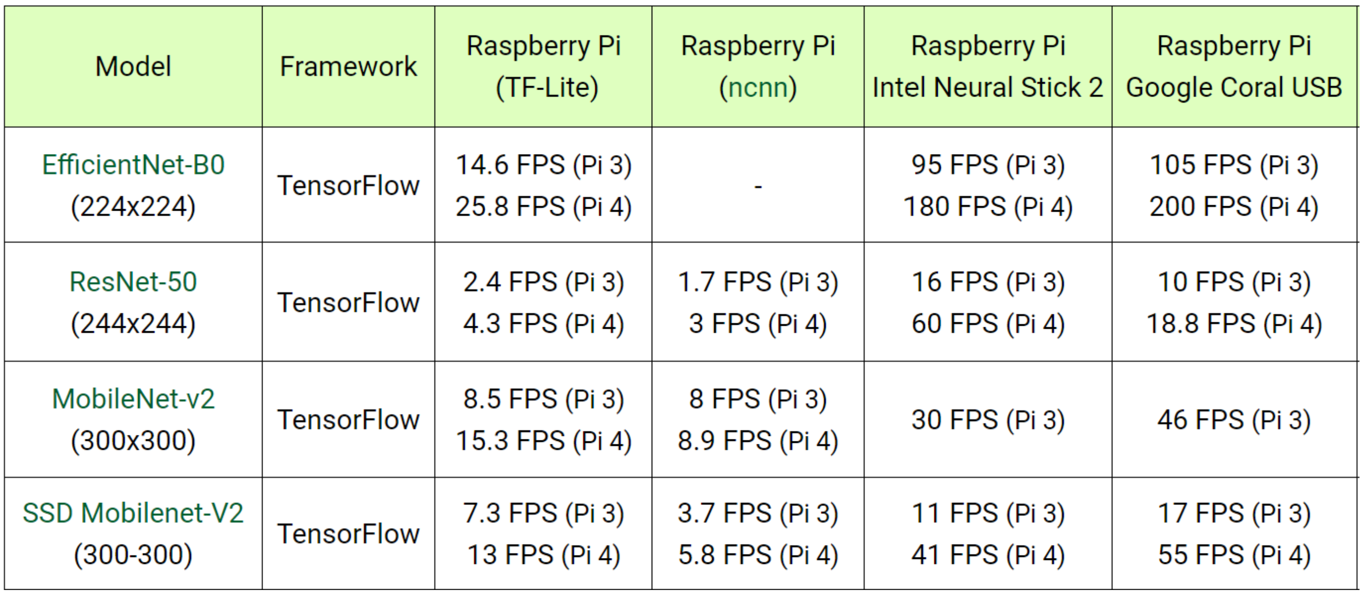
Figure 14: Example of pruning reducing complexity model [34]

2.3.3 Hardware

The major challenges in predicting on embedded devices like phones, microcontrollers and single-board computers are memory size and processor speed. Due to limited resources no different hardware was tested in the proof of concept.

2.4.1.1 Raspberry pi

Raspberry pi is a well known brand of single board computers commonly used in low level embedded computer vision. Any model with lower computing power then a Raspberry Pi 3 B+ should be ruled out, even fully optimized facial recognition models have a hard time on these older models. The table below is a comparison of Raspberry Pi 3 B+ and Raspberry Pi 4B using some popular models for embedded devices. The right two columns show the boosted performance a USB accelerator gives [35].

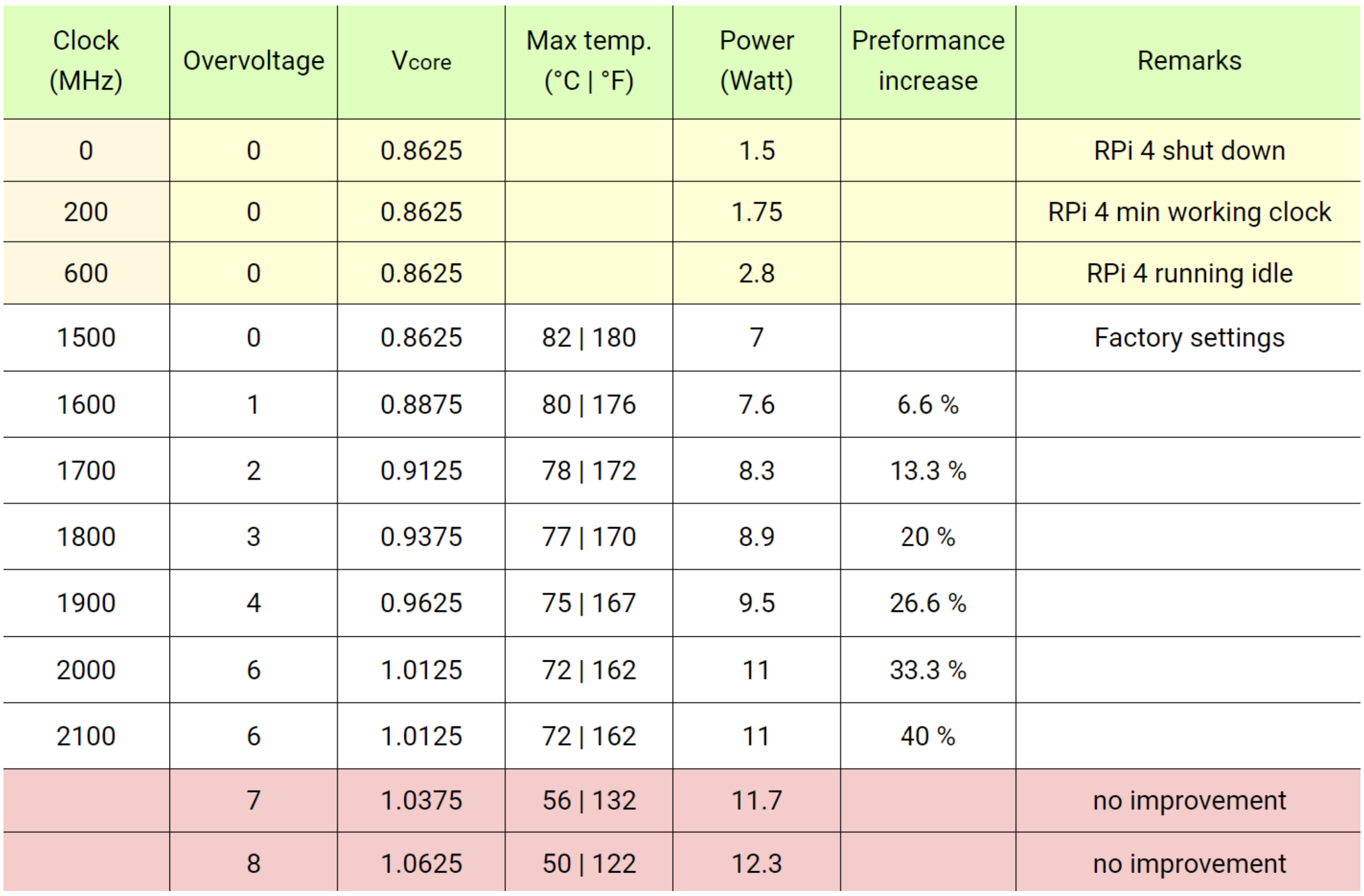


Model	Framework	Raspberry Pi (TF-Lite)	Raspberry Pi (ncnn)	Raspberry Pi Intel Neural Stick 2	Raspberry Pi Google Coral USB
EfficientNet-B0 (224x224)	TensorFlow	14.6 FPS (Pi 3) 25.8 FPS (Pi 4)	-	95 FPS (Pi 3) 180 FPS (Pi 4)	105 FPS (Pi 3) 200 FPS (Pi 4)
ResNet-50 (244x244)	TensorFlow	2.4 FPS (Pi 3) 4.3 FPS (Pi 4)	1.7 FPS (Pi 3) 3 FPS (Pi 4)	16 FPS (Pi 3) 60 FPS (Pi 4)	10 FPS (Pi 3) 18.8 FPS (Pi 4)
MobileNet-v2 (300x300)	TensorFlow	8.5 FPS (Pi 3) 15.3 FPS (Pi 4)	8 FPS (Pi 3) 8.9 FPS (Pi 4)	30 FPS (Pi 3)	46 FPS (Pi 3)
SSD Mobilenet-V2 (300-300)	TensorFlow	7.3 FPS (Pi 3) 13 FPS (Pi 4)	3.7 FPS (Pi 3) 5.8 FPS (Pi 4)	11 FPS (Pi 3) 41 FPS (Pi 4)	17 FPS (Pi 3) 55 FPS (Pi 4)

Figure 15: comparison raspberry pi versions with and without USB accelerators [35]

2.4.1.2 Overclocking raspberry pi

Chip manufacturers carefully balance performance, power consumption and heat generation. This often leads to chips being operated at lower clock rates than their maximum capabilities. Overclocking is a practice that involves adjusting settings to unlock additional performance and push the chip beyond its default specifications [36]. In case of the Raspberry Pi 4B the base frequency is 1500 MHz anything above that is considered overclocking. As seen in the table below overclocking can lead to a significant performance boost up to 40% at 2100 MHz.



Clock (MHz)	Overvoltage	V_{core}	Max temp. (°C | °F)	Power (Watt)	Preformance increase	Remarks
0	0	0.8625		1.5		RPi 4 shut down
200	0	0.8625		1.75		RPi 4 min working clock
600	0	0.8625		2.8		RPi 4 running idle
1500	0	0.8625	82 | 180	7		Factory settings
1600	1	0.8875	80 | 176	7.6	6.6 %	
1700	2	0.9125	78 | 172	8.3	13.3 %	
1800	3	0.9375	77 | 170	8.9	20 %	
1900	4	0.9625	75 | 167	9.5	26.6 %	
2000	6	1.0125	72 | 162	11	33.3 %	
2100	6	1.0125	72 | 162	11	40 %	
	7	1.0375	56 | 132	11.7		no improvement
	8	1.0625	50 | 122	12.3		no improvement

Figure 16: Performance and temperature depending on clock rate [37]

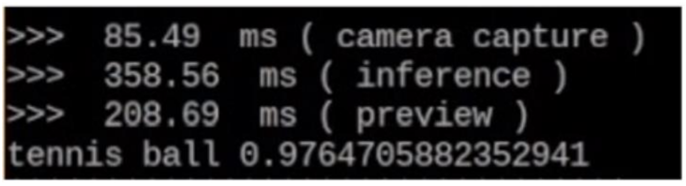
As clock rate increases so does power consumption and additional heat. Having a cooler to cool the Pi is recommended. Luckily there is a safety measure in place to protect the hardware when overheating, when the CPU gets to a temperature close to 85 °C the Pi will start throttling (reducing performance) as it approaches this threshold in order to prevent overheating. [38]

2.4.1.3 USB Accelerators

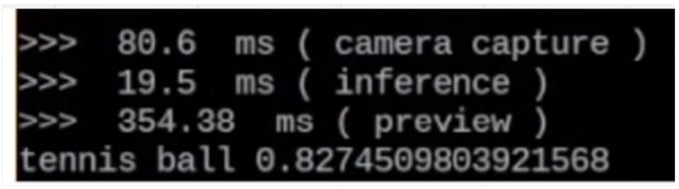
USB hardware accelerators are devices that enhance the performance of USB-connected devices. It is a specialized hardware component that can speed up data transfer and processing by offloading certain tasks from the device to the accelerator. Such device's performance is usually measured in terms of the number of floating-point operations it can perform in a second, commonly referred to as FLOPS. For the Raspberry pi two popular accelerators are commonly used to boost performance.

First the Intel Movidius Neural Compute Stick, this is a USB device that is designed specifically for deep learning applications. It greatly accelerates neural network processing on the Raspberry Pi and is compatible with popular machine learning frameworks like TensorFlow and Caffe. [39]. The Movidius stick can reach up to 100 giga FLOPS per second of computing power. It is possible to use multiple sticks, giving extra performance with each added stick. [40]

Secondly the Coral USB Accelerator is commonly used. This is another USB device designed for accelerating machine learning applications. It uses the Edge TPU (Tensor Processing Unit) to accelerate inference for TensorFlow Lite models. The TPU coprocessor is capable of performing 4 tera FLOPS. For comparison 1 tera FLOPS is equivalent to 1,000 giga FLOPS. [41]



```
>>>  85.49  ms ( camera capture )
>>>  358.56  ms ( inference )
>>>  208.69  ms ( preview )
tennis ball 0.9764705882352941
```



```
>>>  80.6  ms ( camera capture )
>>>  19.5  ms ( inference )
>>>  354.38  ms ( preview )
tennis ball 0.8274509803921568
```

Figure 17: Interference time on Raspberry pi model 3: without Coral(left), with Coral(right) [42]

2.4.1.4 Jetson Nano

The Jetson Nano from Nvidia is a good option when it comes to edge computing, It has the same amount of RAM compared to the Raspberry Pi 4B but in contrary to the Pi it was designed with AI inference in mind. Coming with a Nvidia GPU And quad core CPU. Additionally it also comes with a camera attached, reducing the need to buy extra parts. As well as the Pi it has USB inputs for possible accelerators and GPIO pins to send signals out. [43]. Nvidia also introduces NVIDIA JetPack SDK, a software development kit specifically designed for Jetson devices. It provides a full development environment on Linux with libraries for GPU computing, multimedia and computer vision. [44]

2.4.1.5 Conclusion

The minimal required hardware will vary greatly depending on the model architecture, size and the extra processes the application needs to do outside of predicting. In the context of facial recognition using a complex CNN network and considering 20 FPS as real-time, a raspberry pi 4 is recommended but a raspberry pi 3 B+ can suffice for some smaller optimized models. In addition a USB accelerator can provide a significant performance boost. If price is not an issue then the jetson Nano would be a better option. The build in GPU can provide extra resources, to possibly use less lightweight models for better precision.

2.5 Is it possible to train a model embedded?

This research focuses on computer vision models, which generally require more resources due to the complexity of tasks they perform. Training computer vision models on an embedded device is possible, however the speed of this will largely depend on the device used and the model architecture.

A Raspberry Pi 4 is compatible with frameworks like TensorFlow, Keras and PyTorch allowing for easy training. Unfortunately even the newest Pi models have limited processing power and memory compared to high-end AI development machines or regular CPU desktops. Due to this lack of power the Pi is only capable of handling small-scale AI projects. With the Pi 4 having 4GB of RAM the training process will slow down significantly for projects with larger datasets. [45]

A Jetson Nano although intended for inference has been proven to work with transfer learning [46]. A jetson developer published the results of training a pre-trained ResNet18 model using 5000 images for a binary image classification problem. In this project one epoch took around 7-8 minutes. When scaling this to 100 epochs it would take upwards of 12 hours. Given this estimated training time using the Jetson Nano overnight is a viable option.

2.6 Is it practical to train a model embedded?

Although training a model embedded on a device is feasible in certain cases, it is generally not the preferred option. Training on a embedded device with limited processing power and memory takes significantly longer then training the model first before deploying it on a raspberry pi [47]. Generally, embedded devices are a good option for inference in situations where cloud connections are not optimal. When scaling to larger projects using big datasets it is always recommended to train on cloud or GPU powered machines.

3 Technical Research

3.1 Introduction

This chapter will go over the process of making the proof of concept pet door in detail. The objective of this proof of concept is to demonstrate the feasibility of automating a pet door by using facial recognition technology. the proof of concept was made to recognize three individual dogs and unlocked a pet door if any of these got recognized. The test subjects can be seen in the image below.

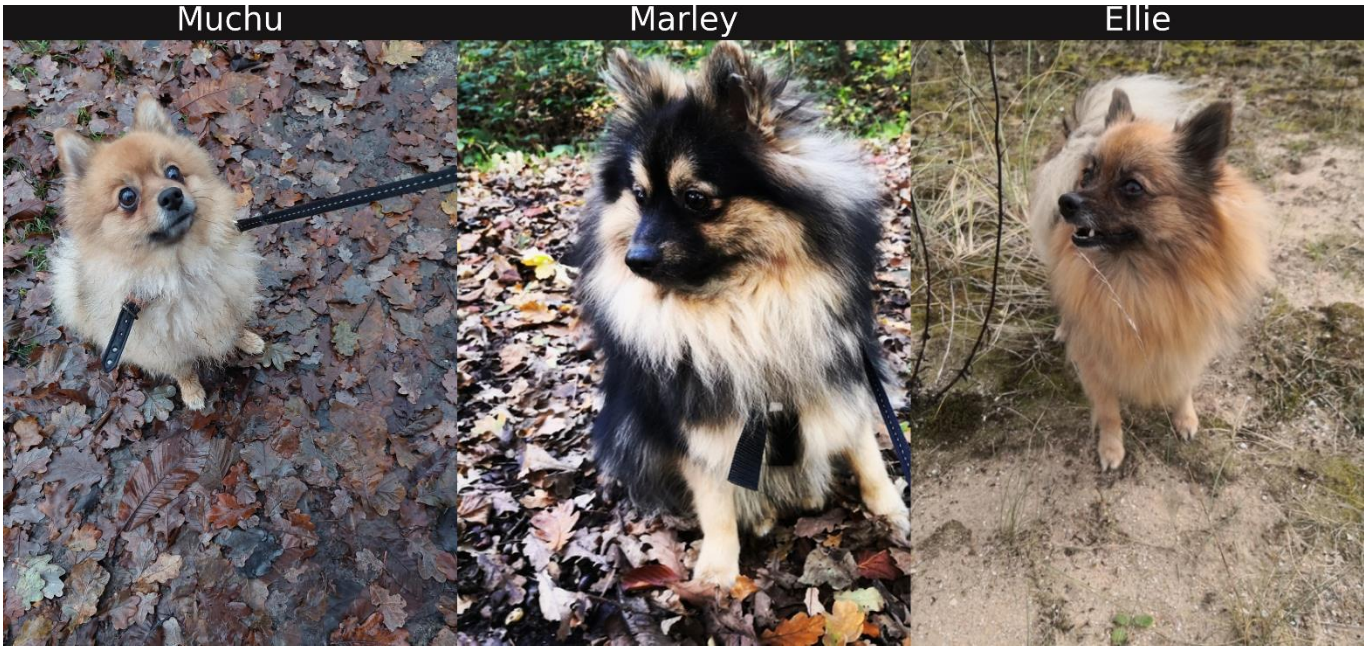


Figure 18: test subjects

3.2 Hardware

To have a working proof of concept some hardware is required. The big challenge of this project is making a full facial recognition application run on the limited computing power of a Raspberry Pi 3 Model B+ with 1GB Ram and a 1.4GHz 64-bit quad-core processor [48]. This device was chosen for its availability at the time.

Besides the Pi, a camera is required to capture the images. For this a Raspberry Pi Camera Module 2 was used, This camera can film at a resolution of 1920x1080 but for this project only a resolution of 640x480 was used. Lastly to control the pet door, two small servo motors were attached to the sides, allowing for the locking and unlocking.

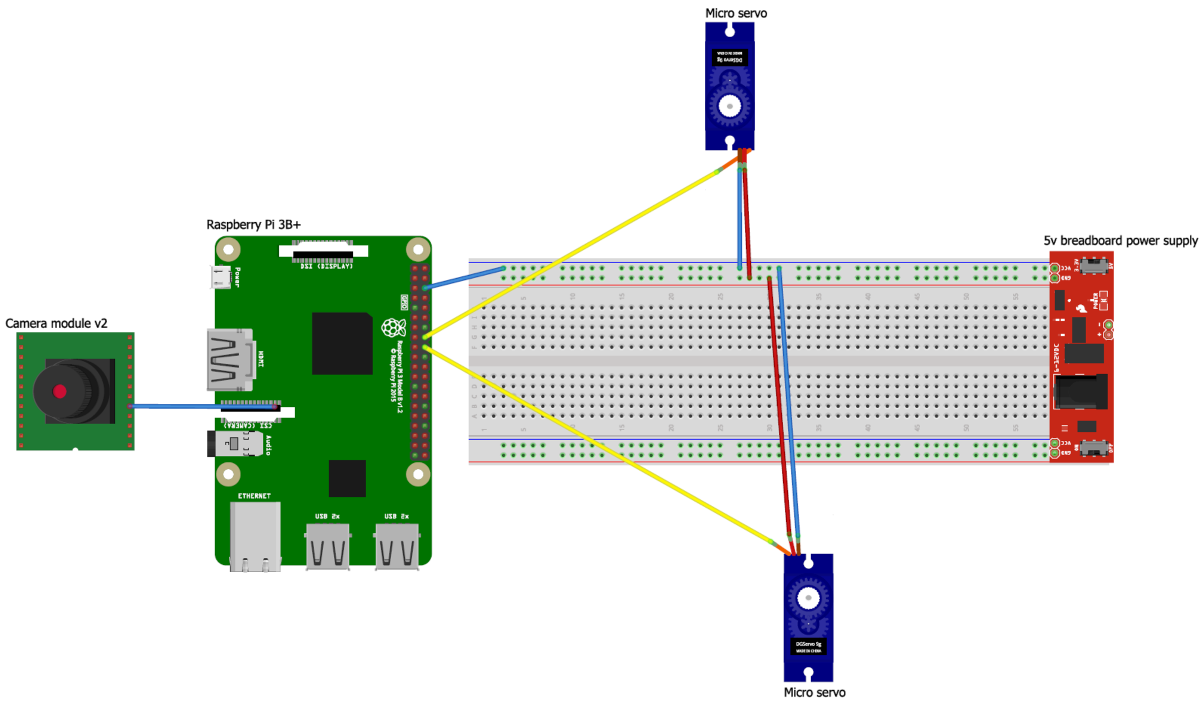


Figure 19: visualisation wiring

3.3 Face detection

Before recognition the face must be detected in an image, this detection is then cropped and used as input. For this task an object detector called MobileNetV2 was chosen for its easy TensorFlow lite implementation and design for embedded devices.

3.3.1 Data

To train any model, data is needed and more data usually means better performance. The data was collected from two publicly available databases, the cat and dog dataset [2], and the animal faces dataset [3]. Additionally, some pictures of the test subjects were added to improve the model's ability to detect them accurately. To label the data, a tool called VOTT was used. The model was trained in iterations with additional data being labeled every few hundred images, based on the model's performance. In total 664 faces were labeled during the training process. The bounding box coordinates generated by VOTT were saved in the format of x, y, width, and height. However, MobileNet required a different format, namely xmin, ymin, xmax, and ymax. As a result, a conversion process was implemented to convert the coordinates into the required format after labeling.

Furthermore to ensure consistency, both the images and bounding boxes were resized to match the resolution of the Raspberry Pi camera. In cases where the aspect ratio of the images did not match that of the camera some padding was added in the form of black borders to prevent stretching and distortion of the faces during training. This was done to ensure that the trained model would be able to accurately detect faces in images captured by the Raspberry Pi camera.

3.3.2 Model

A pre-trained MobileNetV2 model was used from the TensorFlow garden. This is a public library that provides a variety of pre-trained models for transfer learning purposes. The MobileNetV2 model chosen had been trained on the COCO 2017 dataset, which includes numerous pet labels proving useful for detecting pet faces.

The model used was designed for object detection consisting of 3 parts: [49]

- MobileNet-v2 as the backbone network used for feature extraction
- SSD head (Single Shot Detection) for detection
- FPN-Lite (Feature Pyramid Network) as the neck

3.3.3 Training

Transfer learning a model from TensorFlow garden usually requires generating a TFRecord from the labels and images. These are binary files that efficiently store large amounts of data, making them useful for training machine learning models on large datasets and commonly used by TensorFlow. They can be used to store both input data and corresponding labels or annotations for training. The MobileNetV2 model was then trained for 40 epochs with each epoch consisting of 1000 steps. To speed up this training process a service called Google Colab was used. For a small fee Colab provides access to powerful GPUs for faster training.

3.3.4 Scoring

The most common metric to score a object detection model is Mean Average Precision. This mAP score compares the ground-truth bounding boxes with the predicted bounding boxes. This is calculated by first computing the Average Precision for each class separately. AP is a measure of precision and recall, taking into account both false positives and false negatives. It is calculated by plotting a precision-recall curve and computing the area under the curve. AP reflects how well the model is able to accurately detect objects of a specific class.

After calculating AP for each class, the mAP is obtained by averaging the AP values across all the classes. This way, mAP provides a single value that represents the overall performance of the object detection model by taking into consideration the detection accuracy for multiple classes. [50].

The model trained in this project only predicts one binary class, face or no face. So the mAP is equal to the original AP score. The MobileNetV2 model achieved a mAP/AP of almost 95 suggesting that the object detection has a high level of accuracy and precision in detecting faces.

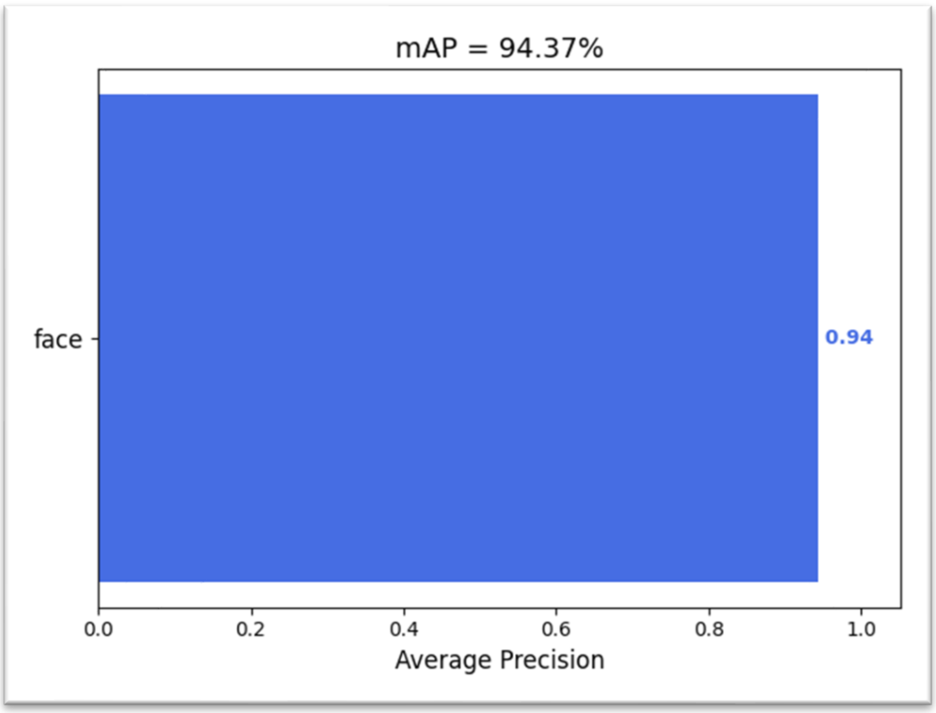


Figure 20: mAP/AP score of MobileNetV2 model on 135 validation images

In the image below a comparison is shown between the true bounding boxes (green) and the predicted bounding boxes (red) of the MobileNetV2 model. As visible the true and predicted bounding boxes are very close.

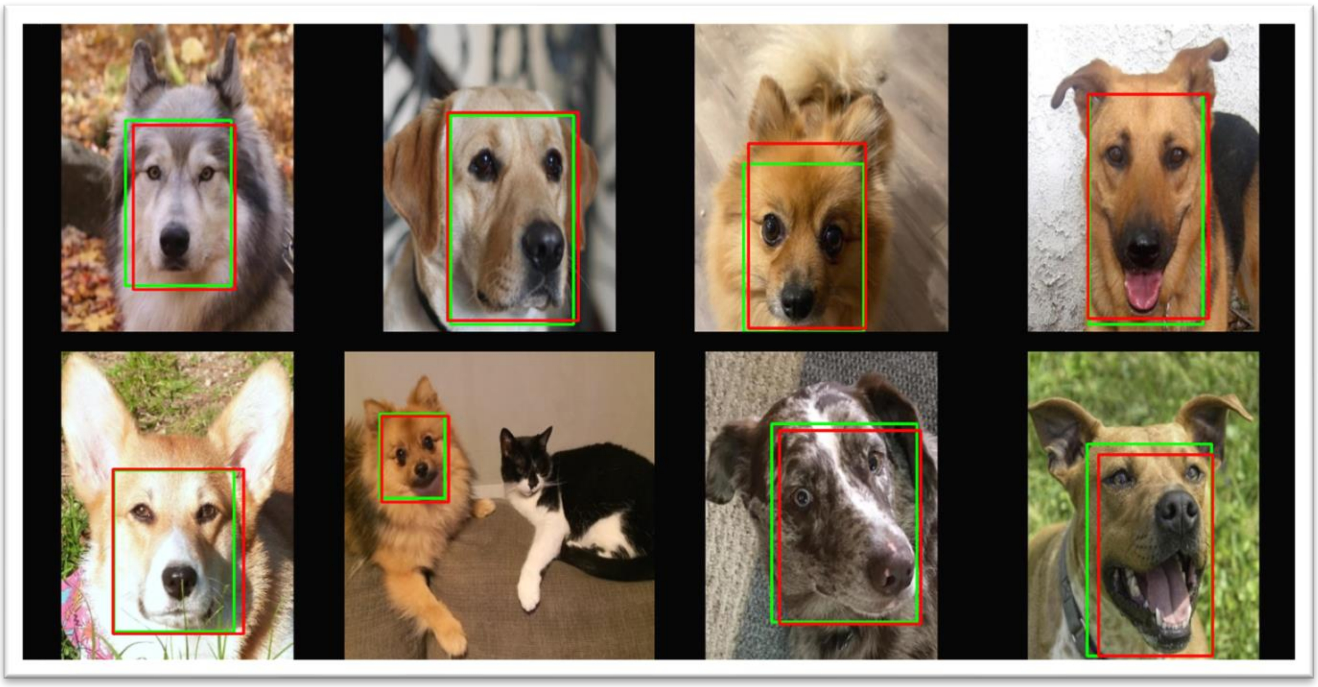


Figure 21: predicted bounding boxes vs true bounding boxes

3.4 Face recognition

The next step after detecting the faces is recognizing. For this task a custom model was build using techniques such as the earlier mentioned residual blocks and triplet loss to accurately create embeddings of the faces.

3.4.1 Data

The data used consists of around 150 images of the test subjects. As seen in the image below the dataset was fairly balanced. The data consists of different angles to closely match live data from a real world application. The faces were cropped and resized to 256x256 using the Auto preprocessing tool described further one in this thesis.

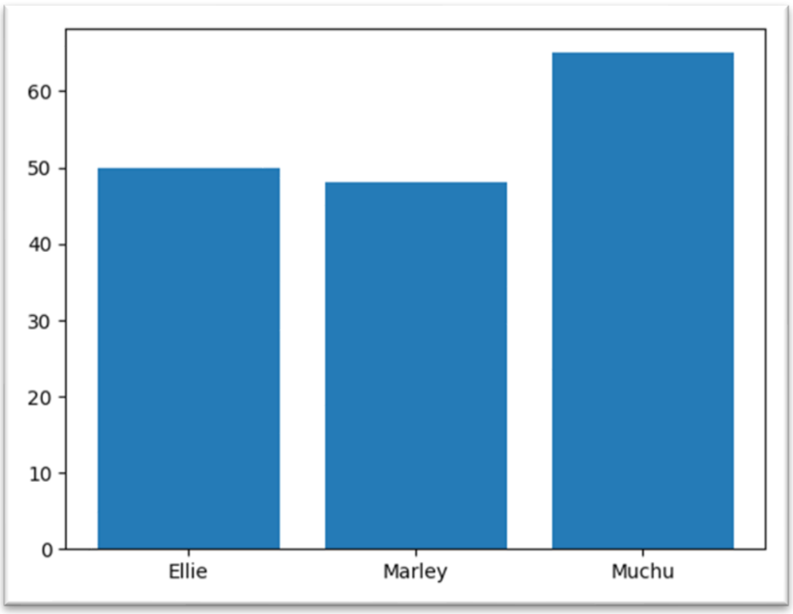


Figure 22: Data for each class

3.4.2 Model

Following is a full description of the model architecture and the purpose of each part. This model was trained using Triplet loss as loss function.

As input The model takes an RGB image of size 256x256. The first layer is a convolutional layer with 16 filters, each of size 7x7, and a kernel of 2x2 for down-sampling. Batch normalization is applied to the output of this layer to normalize the inputs. A max pooling layer with a pool size of 3x3 is then applied to reduce the spatial dimensions of the feature maps. Next are the residual blocks, the number of neurons in each sequential residual block are 16, 32, 64, 128, and 512.

Within each residual block, two convolutional layers are applied, each followed by batch normalization and ReLU activation. The first convolutional layer has a kernel of 2x2, while the second convolutional layer has a kernel of 1x1. An element-wise addition operation is then performed between the output of the first convolutional layer and the output of the second convolutional layer, which forms the residual connection.

After the last residual block, a global average pooling operation is applied to reduce the spatial dimensions of the feature maps to a single value per channel. The output is then flattened to a 1D vector and passed through a dropout layer with a dropout rate of 0.5 for regularization.

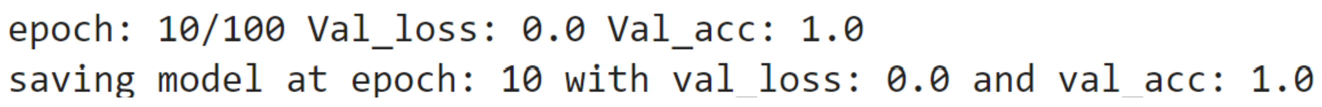
The final layer is a fully connected dense layer with a size of 128 which represents the embeddings of the input image. Finally the output is normalized using L2 normalisation. In total the model consist of just under six million parameters. [51]

3.4.3 Triplet generation

Two triplet generators were implemented. The first one was used for selecting random triplets for validation purposes. This was called at the end of each epoch to mine random triplets from the test set to then validate the performance. The second triplet generator used the concept of online adaptive triplet mining, online meaning the triplets were mined from a batch of the training data given each epoch instead of the whole training set at the start of training. Adaptive meaning the triplets get harder as the loss value drops, a hard triplet is one in which the negative image is closer to the anchor image in embedding space compared to the positive image. This makes it hard for the model to distinguish between the anchor and negative pairs.

3.4.4 Training and scoring

The model was trained for 100 epochs each consisting of 100 steps using a batch size of 30, the model with the lowest triplet loss on the validation set was saved. After only ten epochs the model had reached max convergence with a loss of exactly zero this was probably due to the model only being tested on 33 images total of three dogs.



```
epoch: 10/100 Val_loss: 0.0 Val_acc: 1.0
saving model at epoch: 10 with val_loss: 0.0 and val_acc: 1.0
```

Figure 23: max convergence

the model is able to predict the test set with an accuracy of 100%. without adding a distance threshold. In practice a threshold is used so not every detected face is seen as one of these three dogs. For example when using a threshold of 0.5 any new embedding created that is not closer then 0.5 Euclidean distance to the known faces will not be recognized. Before this threshold any face detected would just be matched to the closest embeddings, no matter how large the distance. When adding this threshold to the prediction the accuracy dropped from 100% to 88%.

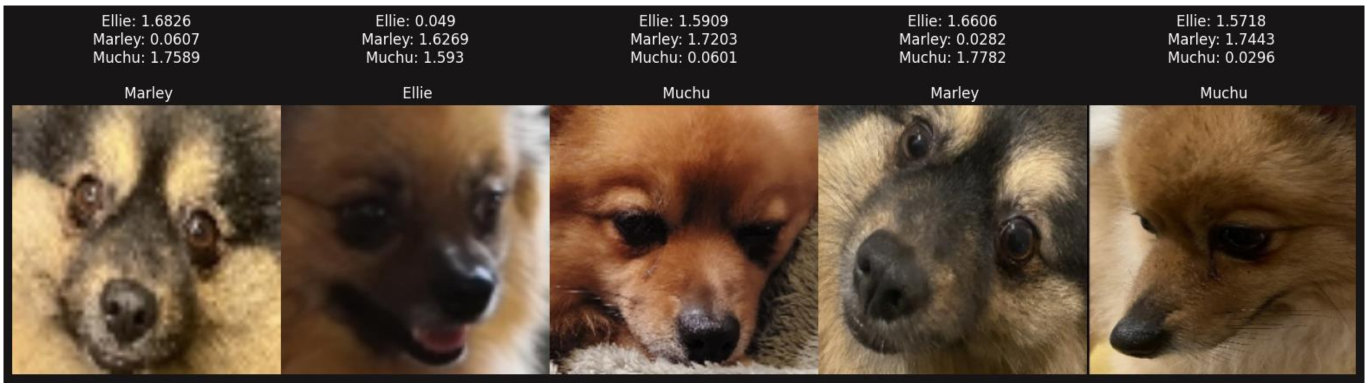


Figure 24: example of predictions using Euclidean distance

3.4.5 Databases

In facial recognition, databases are often used to store and manage facial embeddings or representations of faces. In the case of this project embeddings are used for Face identification meaning when a new face is captured by the detection model, the embedding predicted in the recognition model is compared against the embeddings stored in the database. This way it can verify the individual pet.

In the case of this project the output embeddings were saved in a separate JSON file for each dog. To enhance the model's prediction accuracy, both the training and test set embeddings were added to this JSON database, as having more embeddings increases the likelihood of correctly recognizing the dog. These embeddings were then loaded on the Raspberry Pi for use in the recognition.

3.5 Using camera Raspberry Pi

To use the camera on the Raspberry Pi, a library called OpenCV was used. This library allows for easy video capture and streaming of results. Using OpenCV it was also possible to show the boxes with labels on the stream. Although streaming the result is not necessary for this task it proves useful in showcasing the performance of the models.

3.6 Deploying models

Before deploying these models to the Raspberry Pi, both were converted to TensorFlow Lite format. TensorFlow Lite is a lightweight version of TensorFlow designed for inference on embedded devices and optimized for speed. For running the inference on the Raspberry Pi, three classes were written, one for each model and a third for the camera as mentioned above. In the main code, after initializing the models and loading in the embedding databases, a loop was programmed to run the detection model on every frame. Only when the detection model detected a face was the recognition model called. By using the bounding boxes output of the MobileNetV2 model, the face was cropped for the recognition model. Which in turn generated embeddings for the new cropped face. Those embeddings were then compared to the databases and the label with the lowest Euclidean distance was returned if that distance was under the threshold. Otherwise, the face was not recognized. To use this to control a door a check was implemented in the beginning of the loop, if the label was recognized as an allowed pet the loop was paused and the door opened. This pause is configurable and should be set to allow pets enough time to enter.

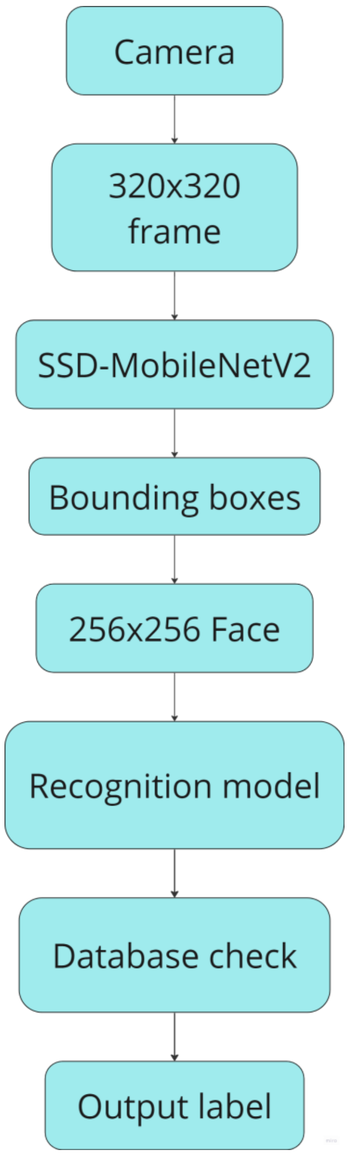


Figure 25: Detection and recognition flow

3.7 Auto Preprocessing

Initially the detection part used YOLOv7 as the model, this was later replaced by MobileNetv2 for easier conversion to TensorFlow lite and better inference speed on embedded devices such as the Raspberry Pi. As the YOLO model had excellent performance on new images it was repurposed for a auto preprocessing pipeline. This served the development of the facial recognition model as it needed cropped faces from all test subjects. By simply providing a folder of pet images the script would output all the cropped faces in the requested aspect ratio using the YOLO predictions. This meant that the only task now was adding each image to the right label. Some false positives also need to be filtered out. As seen in figure 26, the model also detected human faces, one reason for this could be the pre-trained features.

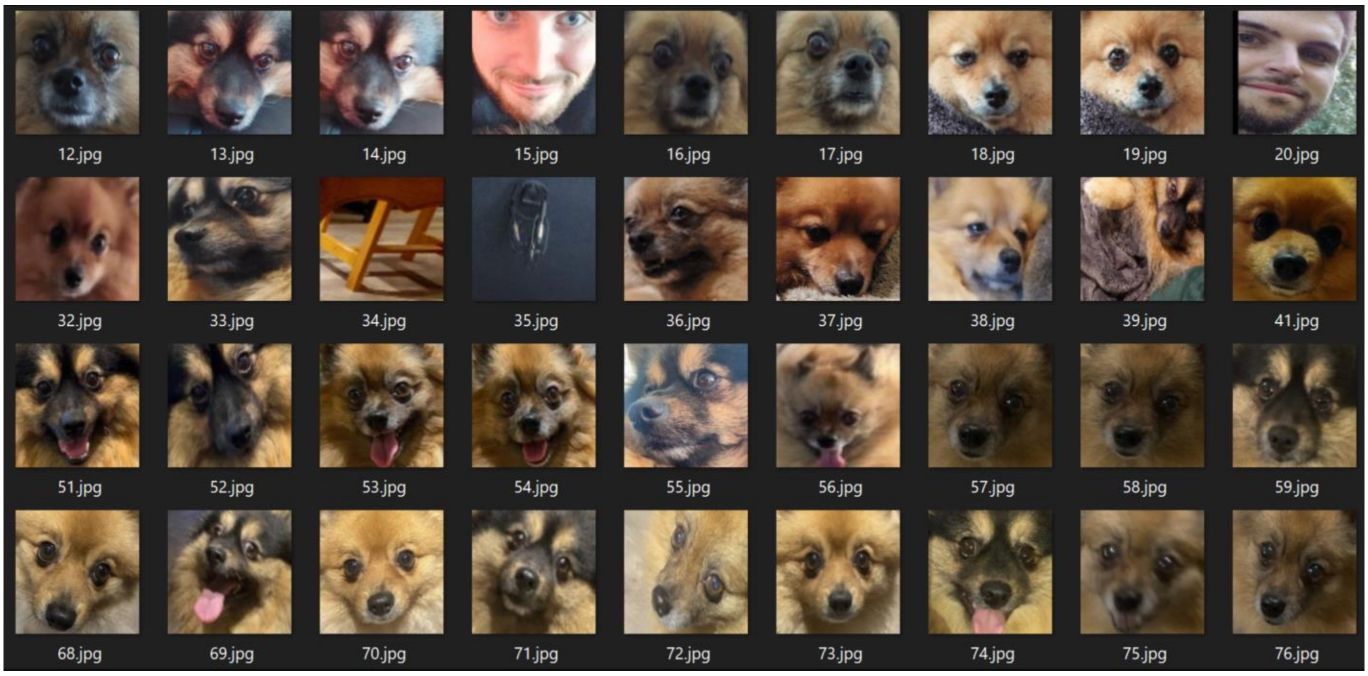


Figure 26: YOLOv7 output of cropped faces

3.8 Result

The result is a functional prediction pipeline that utilizes the camera feed from the Pi Camera Module to detect and recognize pet faces. Subsequently, when a recognized face is detected the pipeline triggers the activation of two servomotors serving as a lock. It is important to note that this is only a proof of concept, in reality stronger servos would be required, as any dog can jump through this lock.

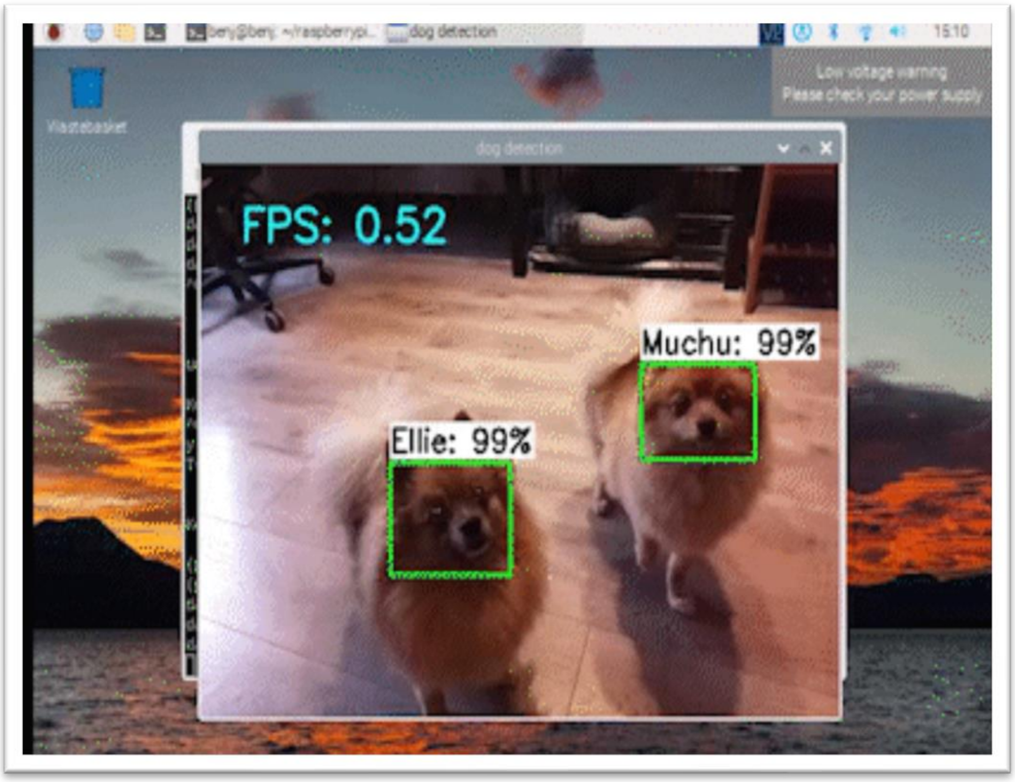


Figure 27: Snapshot of Raspberry Pi prediction feed

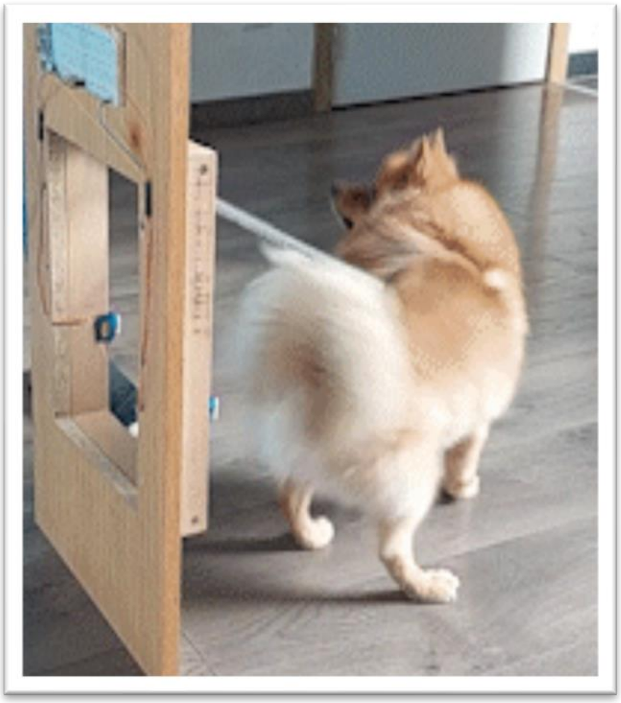


Figure 28: example of proof of concept in use

3.9 What is the minimum camera resolution needed?

The camera resolution used in the proof of concept is 640x480, this is then resized to a resolution of 320x320 to fit the MobileNetv2 model for face detection. This model takes 320x320 images as input so reducing the resolution beyond could negatively impact the accuracy, as the images would need to be stretched to fit the model input. At this time no MobileNetv2 pre-trained model with a lower input resolution is available at the TensorFlow garden. It is however possible to train a model with a custom input resolution form scratch.

The recognition model was originally trained on images of 256x256. Considering that a face when detected takes up around 1/10th of the full image and the detection model gives 320x320 images as output, a resolution of 32x32 was tested. In the image below is a snippet of results when retraining the model on this resolution. It is shown in figure 29 that the model is still accurate, scoring 100% on the test date. Keep in mind that the test data consists of only 33 images and this perfect score does not translate fully to real world application.

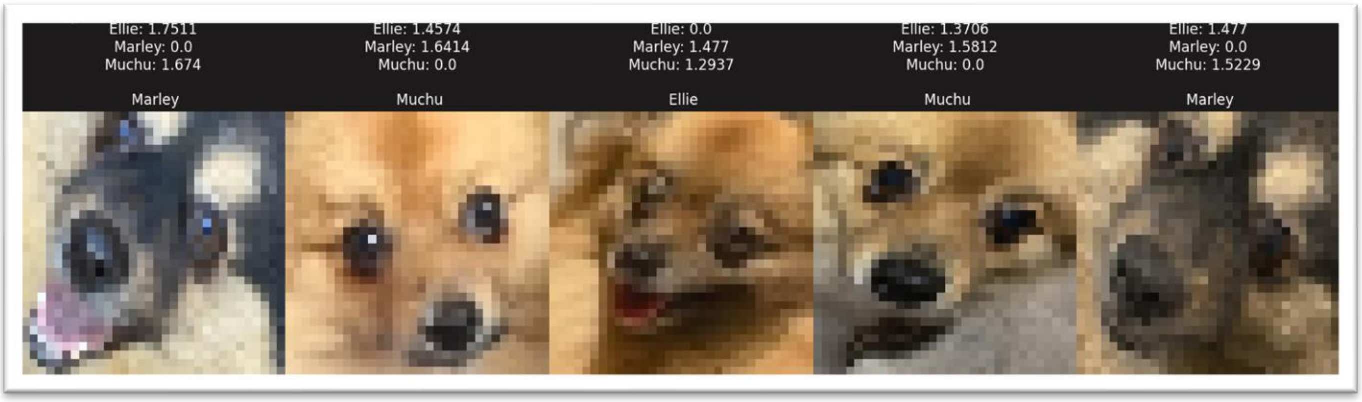
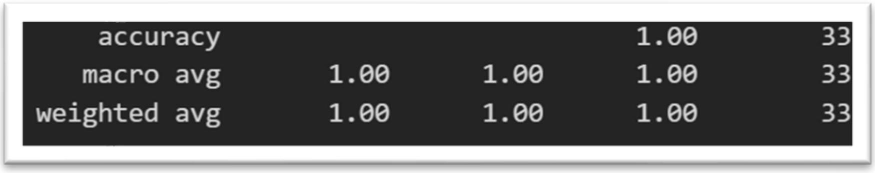


Figure 29: Results Face Recognition for a resolution of 32x32.



```
    accuracy                        1.00        33
   macro avg       1.00      1.00      1.00        33
weighted avg       1.00      1.00      1.00        33
```

Figure 30: Classification report 32x32 resolution

The performance of the model significantly deteriorates when further lowering to a resolution of 16x16, as evident from the classification report in Figure 32. There is a notable decrease in accuracy for the labels Ellie and Muchu, while Marley continues to be recognized correctly 100% of the time, likely due to his distinct black fur. Furthermore when comparing the distances shown in figure 29 and 31 a clear drop between classes is seen, showing that the model has a harder time differentiating between faces.

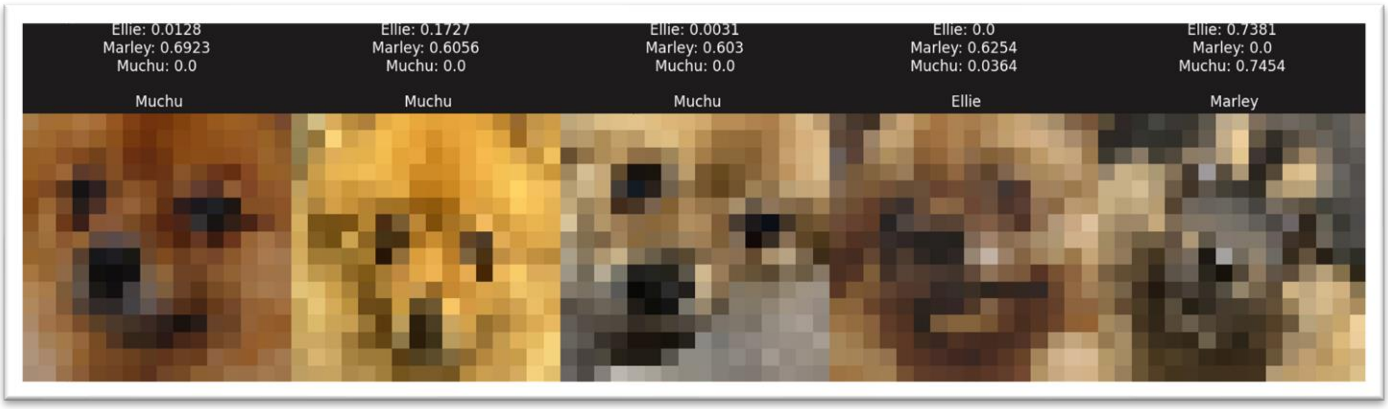
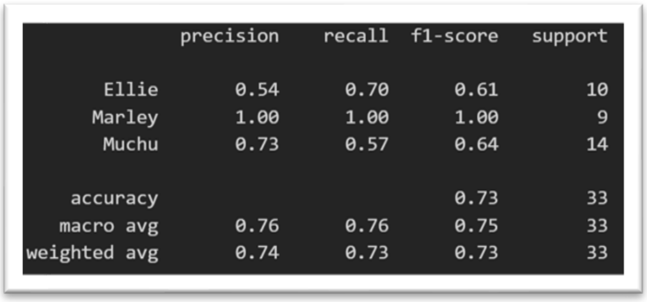


Figure 31: Results Face Recognition for a resolution of 16x16.



```
              precision    recall  f1-score   support

       Ellie       0.54      0.70      0.61        10
      Marley       1.00      1.00      1.00         9
       Muchu       0.73      0.57      0.64        14

    accuracy                           0.73        33
   macro avg       0.76      0.76      0.75        33
weighted avg       0.74      0.73      0.73        33
```

Figure 32: Classification report 16x16 resolution

4 Reflection

4.1 Strengths and weaknesses.

4.1.1 Strengths

The face recognition model used is lightweight and works well for this specific use case. The triplet loss function proved useful in delivering an accurate and robust model, even when lowering the resolution to 32x32 pixels it could still recognise features and deliver quality embeddings. The object detection models both YOLOv7 and MobileNetv2-SSD could precisely detect faces if the face was close and front facing, it also worked surprisingly well on faces further away and faces slightly facing away.

4.1.2 Weaknesses

Even though the MobileNetv2 model was designed for embedded devices, the object detection part proved a computational challenge for the Raspberry Pi. With only 0.6 FPS when detecting faces. This is not considered real-time which was one of the goals for this project. The model trained in this project had a additional problem of not detecting faces if the lighting was not bright enough. This is due to limited training data of different light conditions.

A weakness of the YOLOv7 auto pre-processing project is that it does detect false positives, for example if an image features human faces it will most likely be detected as a dog face. This leads to extra work when preparing the data for the recognition model.

4.2 Is it useable?

4.2.1 project

Based on the current state of the project the detection and recognition speed is not optimal. However, with some optimizations in the models and implementation it could become a fun gadget for smart home enthusiasts or even a safety measure for those living in areas with wildlife that could enter using regular pet doors.

Furthermore even if the project is optimized to reach a speed of over 10 FPS when deployed, it is still only trained to recognize the test pets used in this project. Therefore, to make it a practical and usable product a pipeline needs to be developed to train new weights followed by a web-interface or app for the clients to easily input images for each pet.

These images would then need to be pre-processed using the YOLOv7 pre-processing scripts to train a new recognition model. The client would also need an easy way to upload these new weights to the embedded device.

The locking mechanism used as proof of concept would need to be replaced by a more robust locking or opening mechanism, that is still safe for the pets.

4.2.2 Methods

MobileNetv2 is a widely used detection model for image classification and object detection. it is very useful for any company looking for a computer vision solution on any embedded device. some example use cases in embedded devices are:

- Drones for real-time object detection and tracking to help search and rescue missions or monitoring wildlife populations.

- Doorbells for detecting and notifying when a person is at the door.

MobileNet’s ability to efficiently perform object detection and classification makes it a powerful tool for a wide range of embedded applications.

As the facial recognition methods used in this paper are derived from Facenet. It will be mentioned in this section, FaceNet is a state of the art technology that is still widely used for all sorts of face recognition applications. For example it can be used in security to detect wanted people at airports, in home automation by setting different heat settings depending on the people in the house and many more. FaceNet can also be used for less ethic use cases such as mass surveillance and data collection or the clustering of faces as input for a generative adversarial network to create deepfakes often without consent.

4.2.2 Implementation hurdles

If any business want to implement this project some hurdles must be considered.

Training a new model on the raspberry pi is not practical. Therefore, a service needs to be set up that allows users to train models using their own pet images. The trained models can then be uploaded to the Raspberry Pi for use in the application. This requires a scalable solution to train models on demand, although the recognition model is not that computationally expensive, good scalability still needs to be considered.

Another challenge is data privacy regulations, such as GDPR. If users need to upload their images for the application to work, it is important to ensure clients are aware of how their data is being used and to be transparent about the data collection. To protect the data appropriate measures should be implemented, such as encryption and secure storage.

An additional consideration is the safety of the pets when developing a pet door that opens and closes automatically. The door mechanism should be designed in a way that does not cause any harm or discomfort to the pets when entering or exiting the house. The door should also be sturdy and durable enough to withstand any weather conditions or rough use by the pets. For this it may be necessary to conduct thorough testing and research to ensure that the door mechanism is safe before deploying it in a real-world scenario.

4.3 Alternatives and suggestions

4.3.1 Non-technical

Stijn Hooghe who has a bachelor in animal care suggested to look into Pavlovian conditioning to train the dogs to face the camera.

The detection model detects faces more accurately when they look straight into the camera. If a pet looks away from the camera it can be detrimental for the performance, leading to unreliable embeddings resulting further in wasted computation time. Unlike humans, pets are unwilling participants. Pets don’t realise they have to look into the camera unless they are taught this. This is where conditioning comes in.

Ivan Pavlov observed that dogs instinctively salivated upon being presented with food. He referred to this as the unconditioned response. In his experiment, he rang a bell each time his dog was given food. Eventually, the dog began to associate the bell with the food and would salivate at the sound of the bell alone. This learned behaviour is referred to as the conditioned response.

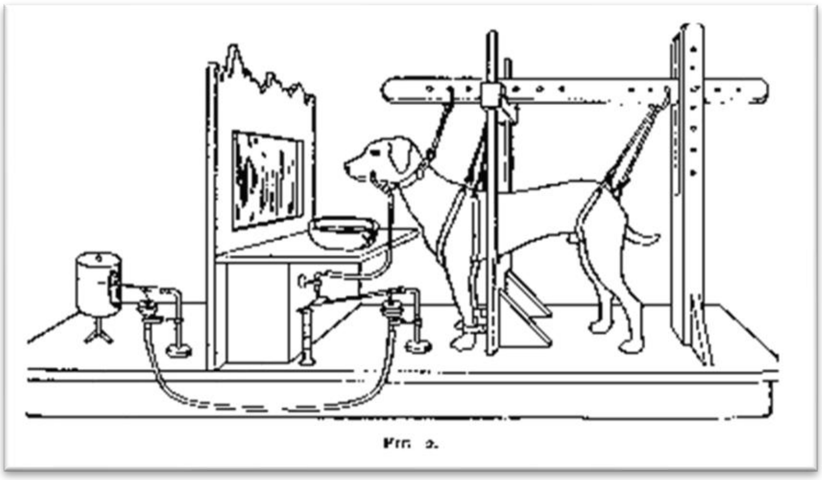


Figure 33: Ivan Pavlov's research setup [52]

This concept of conditioning can be applied to train the pets to look at the camera. For instance, when the pet looks at the camera a clicker can be used to create a sound and at the same time the animal gets positively reinforced for looking towards the camera, by immediately receiving a treat. This process can be repeated until the clicker and reward become associated with facing the camera. However, it is important to gradually phase out the rewards to prevent the dog from stopping the behaviour altogether once the treats are no longer given. [52]

Once the dog has been successfully conditioned, a motion detector can be added to trigger the sound of the clicker when the dog is near the camera triggering the dog to face the camera.

4.3.1 Technical

Some suggestions came up in a consultation with Arne Vanbruaene a software developer who is actively researching face detection for embedded devices. The first suggestion was to use an Edge TPU or switch to a microcontroller that has one built-in. This could significantly improve inference speed. TPUs are designed to accelerate machine learning workloads and can perform matrix multiplication operations much faster than traditional CPUs or GPUs. This makes them an ideal choice for running deep learning models on embedded devices with limited computing power.

The second suggestion was to look into movement detection to lower the amount of idle processing cycles when no pet is in the camera vision. One technique he suggested that would prevent the need of buying and adding sensors is background subtraction.

Background subtraction is a common method used in computer vision to detect moving objects in a sequence of frames from a static camera. The goal is to separate the image foreground (moving object) from the background (stationary object) for further processing. There are several algorithms available for this process one commonly used algorithm is BackgroundSubtractorMOG. This is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. This algorithm models each pixel in an image as a mixture of Gaussian distributions, where each distribution represents the probability distribution of the pixel's intensity value in the background. By using a weighted sum of these distributions, the algorithm constructs a representation of the overall background at each pixel. [53].

As the video sequence progresses, the algorithm adapts the model by adjusting the weights and parameters of the Gaussian distributions. The algorithm compares each new frame of the video sequence with the background model to determine whether each pixel belongs to the foreground (moving object) or the background (stationary object).

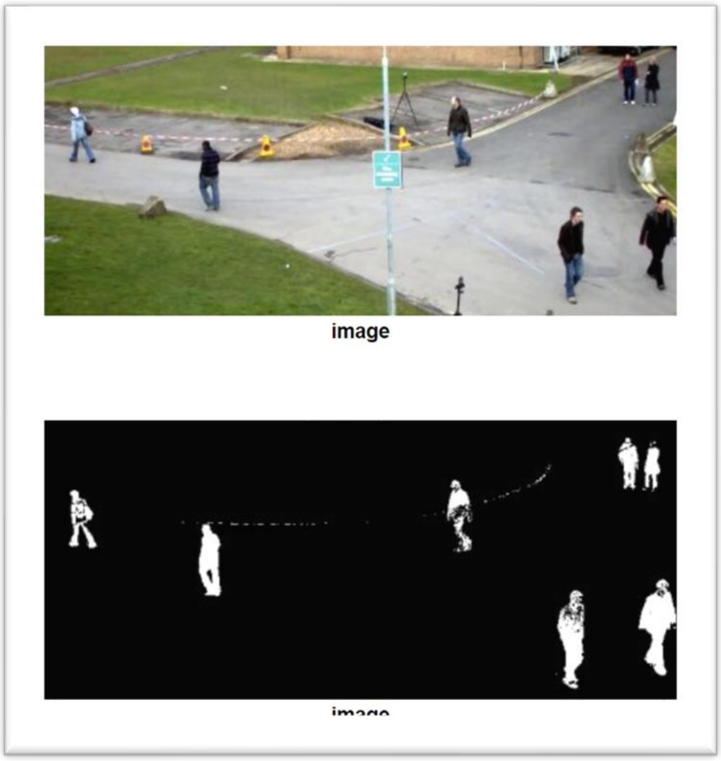


Figure 34: result background subtraction [53].

For the detection and recognition of the faces, FOMO was suggested. Like MobileNet this model is also designed to run on highly constrained devices [54]. FOMO uses a convolutional neural network where the final layer is a per-region class probability map. This map is a two-dimensional grid that represents the image divided into multiple regions. The network is trained on a custom loss function to predict the probability of an object for each of these regions. Delivering a sort of heatmap as seen in the image below. [55]

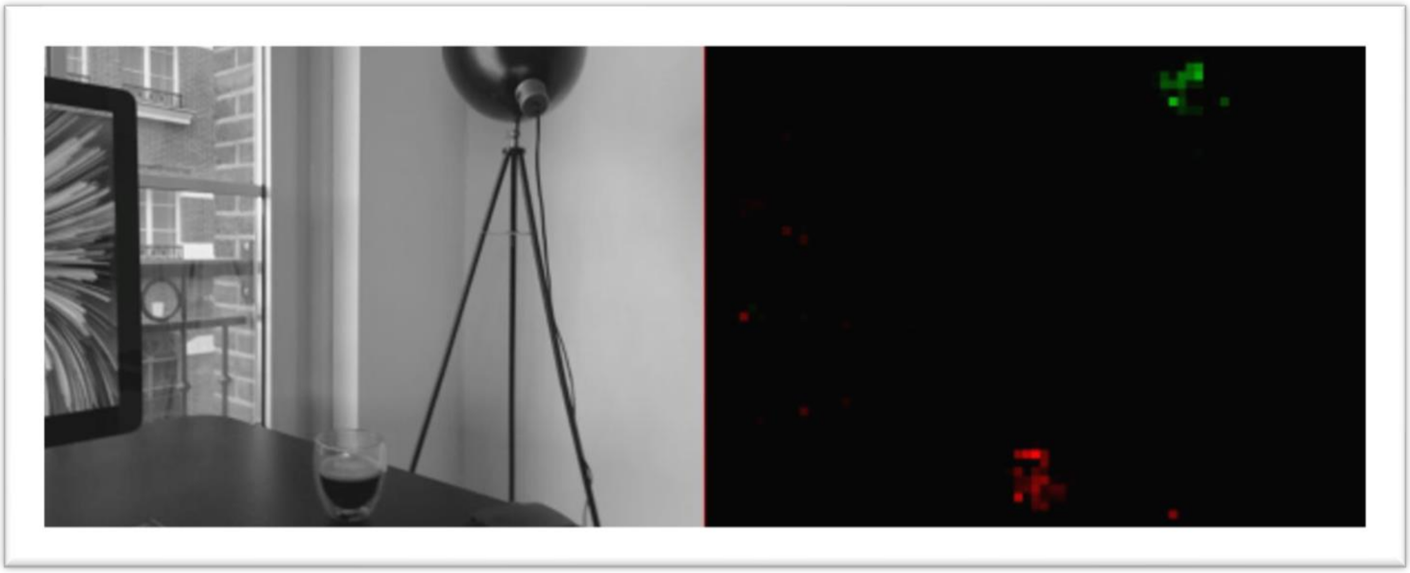


Figure 35: Heatmap produced by the output layer [54]

The size of the heatmap depends on where the previous layers are cut off meaning a 1:1 input to output ratio would give a classification for each pixel. FOMO trains on centroids instead of bounding boxes, this decision was made after realizing that bounding boxes are merely the expected output from object detection models and are not a requirement. Most industries don't need to know the size of the object only the position. Predicting only the centroid instead of the bounding boxes requires significantly less computing power. According to the creators, FOMO is much faster then the original MobileNet-SSD model. A video showcase was published on YouTube where beer bottles are detected at a rate of 60 FPS on a Raspberry Pi 4 [56]. For the purpose of this proof of concept the size and bounding boxes are needed to crop the input images for the recognition model, however if it is possible to calculate the bounding boxes using this centroid it could be a good alternative.

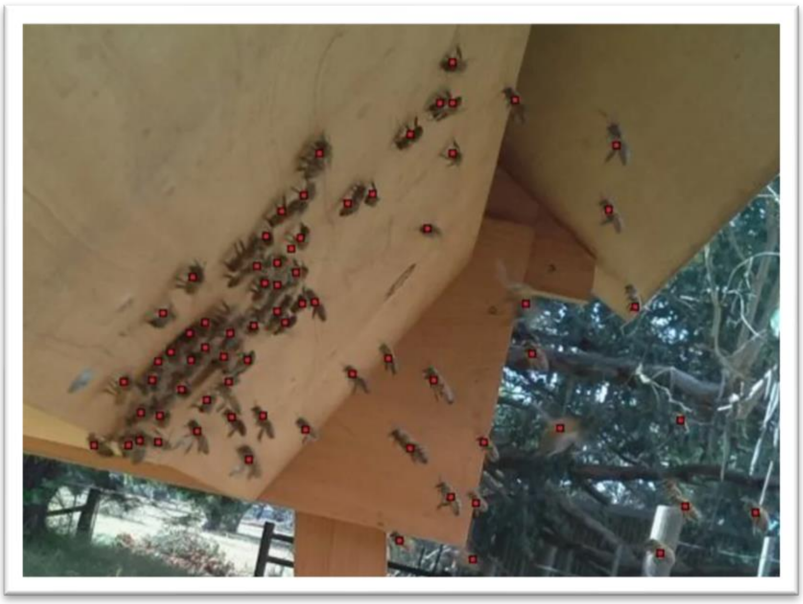


Figure 36: Centroids predicted to count bees. [54]

Another suggestion from Arne was to look into SIFT/SURF. Both are supported by OpenCV and are computer vision techniques used for feature detection, description, and matching in images.

SIFT works by identifying key points in an image that don't change when scaled, rotated and illuminated. Once these key points are detected, SIFT computes a local feature vector for each key point that describes the key point's appearance and geometry. These key point are found by searching for local maxima in the difference of Gaussian scale (DoG) space. This space in turn is constructed by first convolving the image with a gaussian kernel at multiple scales. The kernel is a filter that smooths the images while preserving the edges leaving a series of blurred version of the original image at different scales.

Next the DoG operation is performed by subtracting each blurred image from its neighbouring image in the scale space, resulting in a series of DoG images. The DoG images emphasize the regions of the image where there is a significant change in intensity across scales, and suppress the regions where the intensity is relatively constant. In other words, it highlights the edges and other high-contrast features in the image. [57]

To find key points each pixel in the image is compared with its 8 surrounding pixels as well as nine pixels in the next scale and nine pixels in the previous scales.

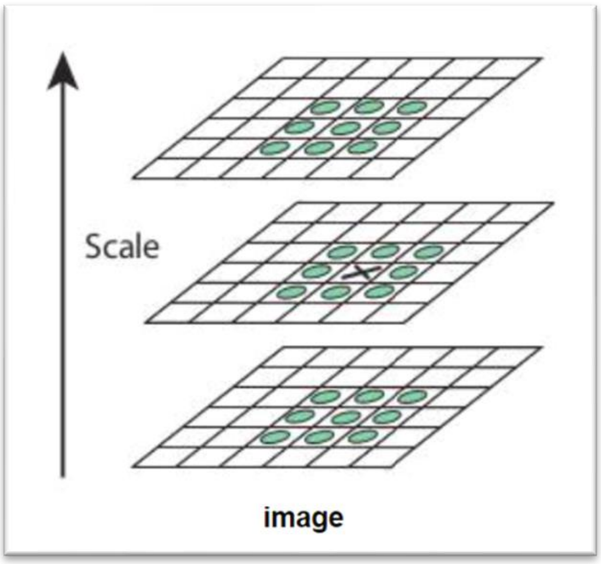


Figure 37: potential key point compared with neighbouring pixels [57]

If the pixel has the highest or lowest intensity value compared to all these surrounding pixels, it is considered a local extrema and is identified as a potential key point. Not unlike how the faces are detected in this project SIFT matches the key points and their feature vectors between two facial images. finding the key points that have the most similar feature vectors. [58]

SURF on the other hand is a faster version of SIFT. Instead of using the Difference of Gaussian approach used by SIFT, SURF uses the Hessian matrix to detect potential interest points. The determinant of the Hessian matrix is calculated at each pixel in the image at multiple scales. If the determinant exceeds a certain threshold, the pixel is considered a potential key point. [59]

If SIFT or SURF can reach similar accuracy for a facial recognition, it would prove a solid option for this project, this is because they are computationally less expensive then a neural network like the ResNet used in this project.

Some further optimization suggestions were mentioned in a consultation with a consultant data scientist from Barco named Ali Adiby. Ali suggested the use of a profiler to find out if it was indeed the models that were causing low FPS or if there are other bottlenecks in the code. Following this advice a profiler named Cprofile was ran on the detect script for about one minute. As visible in the image below the true bottleneck is the MobileNetV2 detection model. The detection takes up 55 of the 63 total seconds the application ran. The interpreter invoke method is the prediction method of this model.

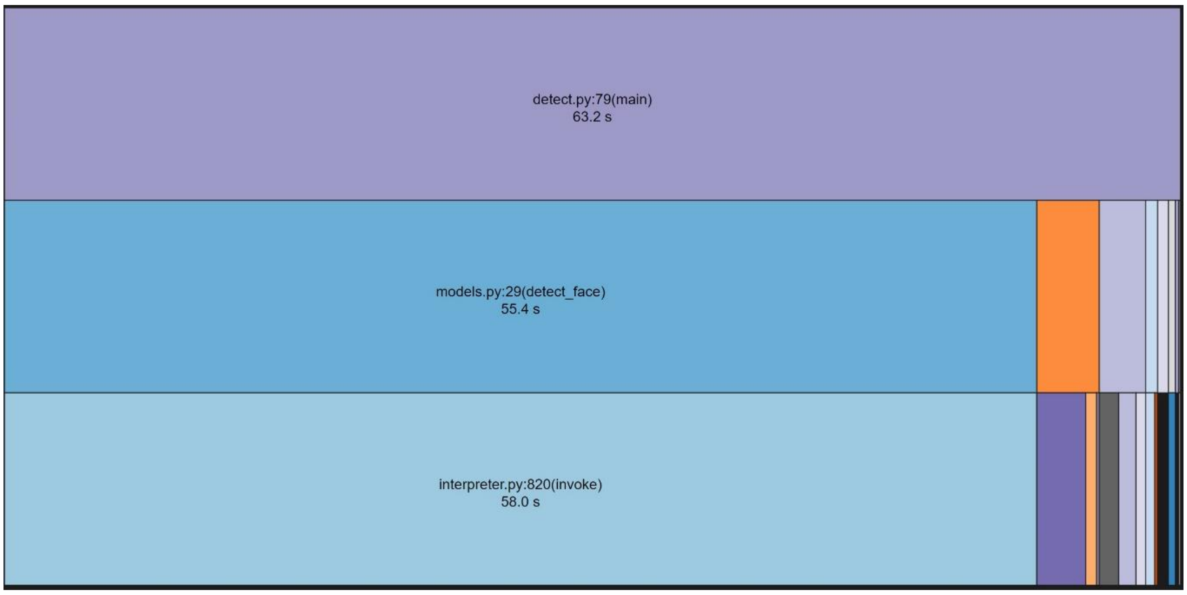


Figure 38: Profiler results visualised using snakeviz

In this consultation some further suggestions were made to decrease the detection time, one of which was batch detection. This is the practice of collecting frames in a batch of a determent amount of images. Only when the batch is full a detection is performed on the full batch, preventing the need to detect each frame. Another way is to simply only detect x amount of frames. It is often redundant to run detection on each frame. When adding a parameter so that detection only runs every 25 frames no noticeable difference is seen visually in the detection. On the other hand the FPS raises significantly. As seen in the image below the difference can also be noticed in the profiler. Were the detection method now only takes up half of the total time. Removing the bottleneck.

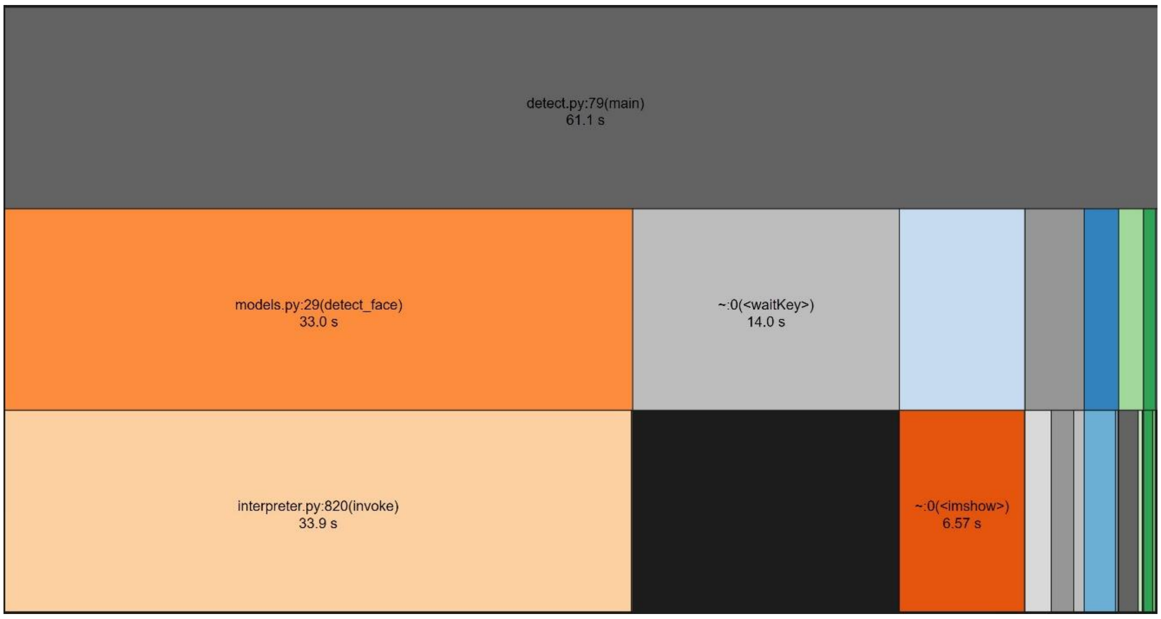


Figure 39: Profiler results after reducing detection from every frame to every 25

4.4 Is there any social, economic or socio-economic value?

For people that don't want a normal pet door for safety reasons this door can be of social value. With this project it is no longer needed to monitor pets or let them out to use the toilet which at times leads to unpleasant surprises. For these people it could serve by relieving the stresses of a unsafe pet door while also adding the benefit of enjoying social occasions without worrying. This also leads to a potentially reduced energy use for heating and cooling the house as it alleviates the need for humans opening the door leading to less heat or cold escape. The biggest benefit is preventing break-ins in high crime areas where large enough pet doors could be used as entry points.

During the consultation with Ali Adiby, he mentioned the benefit of creating jobs. If a demand for this kind of product is big enough, jobs will form for maintaining and developing these devices. He also briefly touched upon positive sides to mental health. This device can be a reason for people to get pets as it mitigates some security risks and worries mentioned above. Owning pets, especially dogs have shown an increase to overall mental health [60]. Another side of this is it can reduce worries, for example pets can now escape house fires if the owner previously only used regular doors. Additionally a remote lock system can also be implemented to prevent dogs from going outside and barking at the neighbours having dinner outside in term improving social connections. All of the above lead to a better enjoyment of social activities and increase in mental health.

4.5 Further research

An alternative suggestion for further research would be to simplify the task of pet detection and recognition by focusing on image classification instead. By training a model like MobileNetv2 on a large dataset it may be possible to directly classify an image as containing a known pet or not. This approach simplifies the model and potentially increases its inference time. Additionally it could address the challenge of pets not always facing the camera at the right angle. One hurdle of this approach is the increased amount of data needed. A solution like this would need images of all angles sides and light conditions.

Another area for research is model fusion, which involves incorporating recognition layers into the Object detection model architecture, with additional processing layers in between to reshape the bounding box output into cropped faces. This approach could reduce the prediction time by performing only one prediction instead of two every time a face is detected. Moreover it would eliminate the need to load two separate models into memory simultaneously, being more memory efficient.

5 Advice

In this section advice will be given for anyone wishing to implement, build upon or try a similar project. An example of a similar project is automatically opening of a garage door or car barrier when the correct license plate is recognized.

5.1 Should the same technologies be used?

On a Raspberry Pi 4 these technologies with some modifications are a great option. MobileNetV2-SSD is initially designed to have lower computational requirements to work well on embedded devices, while still achieving good accuracy given enough quality training data. On the recognition side a ResNet with triplet loss remains a widely used technique for facial recognition. Providing the ResNet does not have too many layers hereby reducing the time needed for predictions. Both models are easily converted too Tensorflow lite, further improving the inference time on a Raspberry Pi.

In a consultation with Dr. Abdelkrim Belhaoua, a highly educated researcher in the field of image processing, and computer vision, he advised the use of a Jetson Nano. Suggesting that with jetsons increased processing power, framework and easy implementation it is a good candidate for this project. Additionally this device has a GPU build in which can be utilized for better performance. Like the Raspberry Pi, Jetson also has GPIO pins to control the servo motors. He further mentioned that a YOLOv7-tiny model can reach up to 15 FPS on a Jetson Nano, even more when looking at the new state of the art YOLOv8-nano model. The conclusion is that a Jetson Nano albeit more expensive provides better inference times and the possibility to use generally better scoring models like YOLO while utilizing the build in GPU.

5.2 Recommendations

5.2.1 Optimizing interference time

The biggest factor in real-time recognition on a embedded device is the computational needs of the application therefore some recommendations are:

- Quantizing
- Pruning
- Model Fusion
- Using lower resolution

As explained in the research Quantizing and pruning will greatly reduce the inference time for a small loss in performance. Model fusion means combining both predictions into one, so removing the step of getting bounding boxes from the MobileNetV2 model and using these to crop a face for input to the other model. This process can be simplified by adding the recognition layers to the detection model, while also adding a layer in between these model that extracts the area of interest serving as input for the recognition. In short creating one neural network from two making it a single prediction task. As seen in the research lowering the resolution of the camera doesn’t necessarily mean big accuracy drops. Lower resolution does mean reduced input size and less computations and memory needed to process the images. Lowering the camera resolution gradually can lead to a good balance of speed and accuracy. If it is possible set the camera output resolution to the same resolution as the model. This will alleviate the need to resize the input reducing time needed to detect.

5.2.2 Optimizing performance

If inference speed is not a problem I would advise adding alignment to the facial recognition pipeline. Alignment works by aligning or normalizing facial images to a standard pose or orientation. Reducing the variability caused by pose and light. By normalizing the faces it can improve accuracy and robustness, it does however require some extra processing.

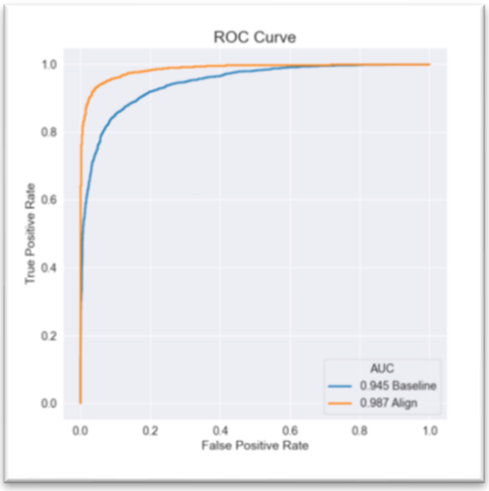


Figure 40: ROC curve with and without alignment. using FaceNet [61]

A good example of a face alignment pipeline was introduced in a paper called “DeepFace: Closing the Gap to Human-Level Performance in Face Verification”[62] published by Facebook's AI research team. The pipeline can be seen in the image below.

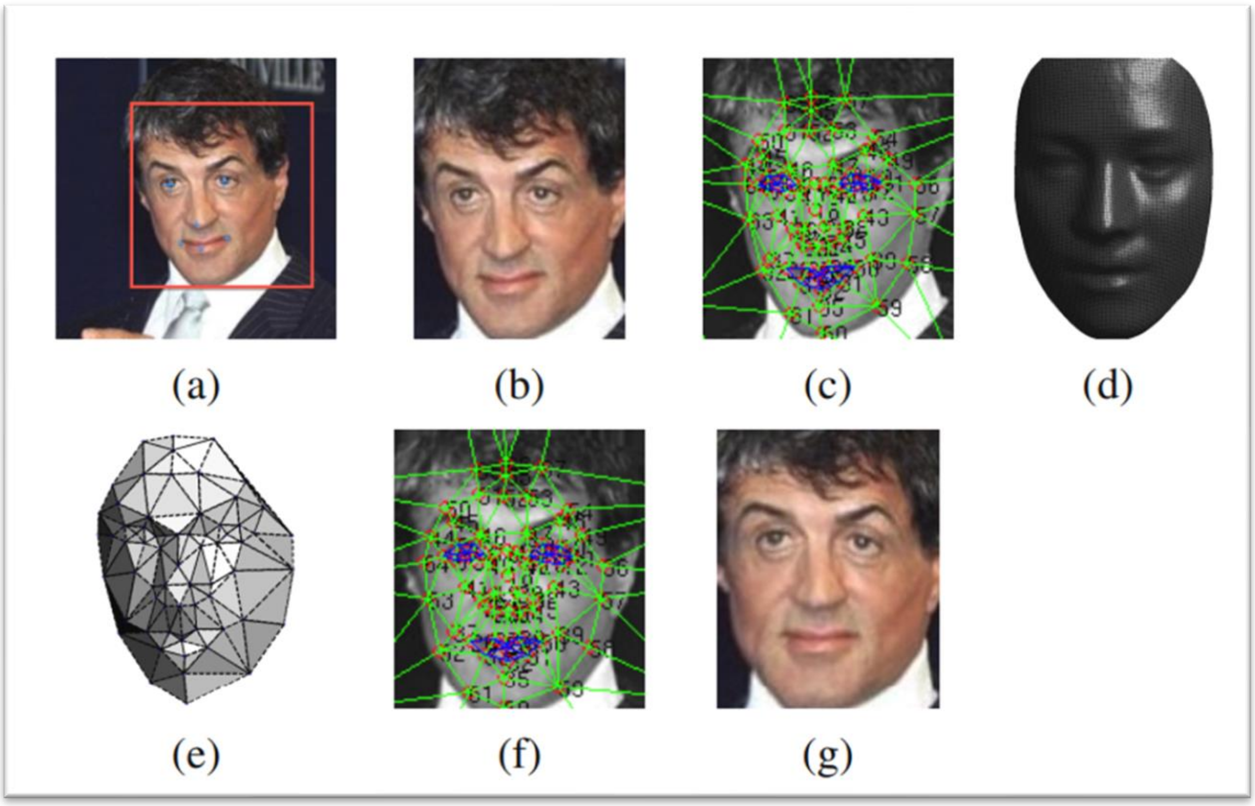


Figure 41: Face Alignment pipeline [62]

The first step is to detect the face in the first place, then 6 fiducial points are placed on the key facial landmarks. These are the eyes nose mouth corners and mouth center (a). Using these points the face is then cropped and aligned to a 2D plane resulting in a frontal view of the face (b). In (c) another 67 fiducial points are placed on more detailed facial features. These points are used to create a Delaunay triangulation, which is a way of partitioning the face into non-overlapping triangles which are the green lines on the image. Following that a 3D shape is created using these 67 points (d). The visibility of the triangles in the 3D model is then calculated based on their orientation with respect to the fitted camera. Triangles that are more visible appear darker in the visualization (e). Extra points are predicted using this 3D model to further refine the alignment (f). Finally the result is a front facing aligned face (g).

5.3 Practical advice implementation

The POC in this paper is only tested and trained on images with good lighting, when it gets dark the POC is unable to detect and recognize faces. If it is intended to implement this as a real world application, the device should work at all times of the day. One suggestion for this is to add a light source and a light/movement sensor. Using these it is possible to turn on a light depending on the data from the sensor. For this to work new training images need to be added mimicking these lighting conditions. Dr. Abdelkrim Belhaoua suggested the possibility of using infrared images instead. This would address the issue of varying light conditions. However it would also introduce certain drawbacks such as less distinct features for recognition and an increase in costs due to the higher price of infrared cameras.

As mentioned earlier dogs may not have the best understanding of the device, it is recommended to provide them with a clear indication to tell them they are recognized and can enter. One way to achieve this is with a consistent sound notification.

Dr. Abdelkrim Belhaoua pointed out another practical concern being the device itself. It is necessary to address factors such as power consumption, cooling and protection of the device. The camera and computing components must be protected against all weather conditions. Additionally proper cooling should be implemented to prevent overheating, when selecting a cooling system it is suggested to select one that doesn't generate excessive noise as it could annoy owners or scare the dogs.

In terms of consumer and environment the total power consumption of the device becomes an important factor and should be considered. Another consideration is providing the device with power, as attaching a visible cable to the door is not aesthetically pleasing or practical. Dr. Abdelkrim Belhaoua suggested selling the pet door as a feature integrated within a normal door. This approach allows users to avoid worrying about device attachment and potentially enables hiding the power cable discreetly with the door's attachment mechanism.

5.4 Use the auto pre-processor tool

In this thesis a YOLOv7 model was trained on thousands of pet faces, with this model a script was written to extract the bounding boxes of every detected face and crop an image into a desired shape. This can easily be used to create good training data for any facial recognition model. Saving allot of time and tedious manual labelling. The only task remaining is to classify the faces to the right pets.

5.5 Collect diverse data

One thing that can greatly improve the recognition and detection model is using images in all sorts of lighting environment and angles. For example the model trained in this thesis had a hard time detecting faces when the lighting dropped from bright to dimly lit. collecting this data is not an easy task and it is unlikely to account for all possible conditions the model will meet in a real world application.

5.6 Further advice

Dr. Abdelkrim Belhaoua suggests that when embarking on a project, it is important to prioritize the project goal before selecting models and gathering data. For instance, if the objective is to develop a recognition model that achieves a frame rate of 10 FPS, the next logical step would be to choose the appropriate device and model. In this case, it is preferable to consider the device first since certain devices offer a wider range of model choices. However, if the goal is solely focused on detection, it becomes possible to opt for a model like FOMO and be less stringent in device selection.

As business advice Dr. Abdelkrim Belhaoua suggests developing an mobile app to connect with the device, however this does mean the device requires internet connection. An app like this could send a notification to the owner if there pet goes outside or inside. Notifying when unknown creatures or humans are detected would be especially valuable. This app could serv both as security and connection to the pet. Making it possible to call dogs inside and close remotely could also be of benefit, for example when the dog is a nuisance for the neighbors by barking, simply calling the dog inside using some kind of trigger and closing the door would solve this.

6 Conclusion

Multiple approaches exist for addressing facial recognition on an embedded pet door. When weighing up these approaches the choice between accuracy and interference time is still prevelant. As AI on edge devices grows in popularity and the devices get more powerful the need to choose between inference and precision decreases and models like MobileNetV2 and YOLOv7 start running faster on embedded devices.

In conclusion, the use of embedded facial recognition in the way demonstrated in this thesis is a viable solution, as it can recognize pets and unlock a door without the need for internet connection. When not every frame is detected it is able to run smoothly. The proof of concept in this thesis successfully implements two neural networks for detection and recognition. Suggestions for further research include several optimization techniques such as quantization, pruning, lowering resolution and the combining of multiple neural networks into one.

With those implemented a similar project could achieve close-to-real-time prediction speed. However it is worth noting that while MobileNetV2 is designed for embedded devices, the model did not perform optimally on the Raspberry Pi even after being converted to Tensorflow Lite. Therefore future studies could explore other techniques for object detection and facial recognition to improve performance on embedded devices such as FOMO, SIFT, SURF. As well as other more powerful devices such as the Jetson Nano.

Before this project can be used commercially, several things must be considered. Firstly there must be a way for pet owners to train new iteration of the model using their own pet images. As training embedded is not feasible it necessary to train on the cloud. For this data privacy needs to be carefully considered as users would need to upload their pet images. Protecting users personal information is a priority.

Additionally, in the proof of concept the door uses locks that are too weak. Therefore, an alternative method of locking or opening/closing the door must be considered while also keeping the safety of the pet in mind.

Finally this thesis provides a foundation for future research in this area, and the techniques suggested in this work can serve as a useful starting point for building more advanced pet door automation systems. The thesis also serves as a demonstration that recognition can be done successfully on animals other than humans. Additionally proving that with limited data a facial recognition solution can be deployed on an edge device.

7 References

[1] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, ‘MobileNetV2: Inverted Residuals and Linear Bottlenecks’. arXiv, Mar. 21, 2019. Accessed: Apr. 07, 2023. [Online]. Available: http://arxiv.org/abs/1801.04381

[2]‘Cats-vs-Dogs’. https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset (accessed Apr. 07, 2023).

[3] ‘Animal Faces’. https://www.kaggle.com/datasets/andrewmvd/animal-faces (accessed Apr. 07, 2023).

[4] F. Schroff, D. Kalenichenko, and J. Philbin, ‘FaceNet: A Unified Embedding for Face Recognition and Clustering’, in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2015, pp. 815–823. doi: 10.1109/CVPR.2015.7298682.

[5] ‘Deep Metric Learning for Signature Verification’. https://blog.fastforwardlabs.com/2021/06/09/deep-metric-learning-for-signature-verification.html (accessed Apr. 07, 2023).

[6] ‘A Guide to the YOLO Family of Computer Vision Models’, Data Phoenix, Feb. 12, 2023. https://dataphoenix.info/a-guide-to-the-yolo-family-of-computer-vision-models/ (accessed May 15, 2023).

[7] ‘The evolution of the YOLO neural networks family from v1 to v7. | by Maxim Ivanov | Deelvin Machine Learning | Medium’. https://medium.com/deelvin-machine-learning/the-evolution-of-the-yolo-neural-networks-family-from-v1-to-v7-4d4fab3c4db7 (accessed May 15, 2023).

[8] F. Kateb, M. M. Monowar, Md. A. Hamid, A. Ohi, and M. Ph. D., ‘FruitDet: Attentive Feature Aggregation for Real-Time Fruit Detection in Orchards’, Agronomy, vol. 11, p. 2440, Nov. 2021, doi: 10.3390/agronomy11122440.

[9] C.-Y. Wang, A. Bochkovskiy, and H. Liao, YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. 2022. doi: 10.48550/arXiv.2207.02696.

[10] G. Boesch, ‘YOLOv7: The Most Powerful Object Detection Algorithm (2023 Guide)’, viso.ai, Feb. 21, 2023. https://viso.ai/deep-learning/yolov7-guide/ (accessed May 15, 2023).

[11] S. Rallabandi, ‘Activation functions: ReLU vs. Leaky ReLU’, MLearning.ai, Mar. 26, 2023. https://medium.com/mlearning-ai/activation-functions-relu-vs-leaky-relu-b8272dc0b1be (accessed May 15, 2023).

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, ‘MobileNetV2: Inverted Residuals and Linear Bottlenecks’. arXiv, Mar. 21, 2019. Accessed: Apr. 21, 2023. [Online]. Available: http://arxiv.org/abs/1801.04381

[13] C.-F. Wang, ‘A Basic Introduction to Separable Convolutions’, Medium, Aug. 14, 2018. https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728 (accessed Apr. 21, 2023).

[14] ‘How single-shot detector (SSD) works?’, ArcGIS API for Python. https://developers.arcgis.com/python/guide/how-ssd-works/ (accessed Apr. 21, 2023).

[15] ‘MobileNet version 2’. https://machinethink.net/blog/mobilenet-v2/ (accessed Apr. 21, 2023).

[16] M. Oršić, I. Krešo, P. Bevandić, and S. Šegvić, ‘In Defense of Pre-trained ImageNet Architectures for Real-time Semantic Segmentation of Road-driving Images’. arXiv, Mar. 20, 2019. Accessed: May 29, 2023. [Online]. Available: http://arxiv.org/abs/1903.08469

[17] ‘MobileNetV2: The Next Generation of On-Device Computer Vision Networks’, Apr. 03, 2018. https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html (accessed May 29, 2023).

[18] ‘YOLOv7 Paper Explanation: Object Detection and YOLOv7 Pose’. https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/ (accessed May 29, 2023).

[19] K. He, X. Zhang, S. Ren, and J. Sun, ‘Deep Residual Learning for Image Recognition’. arXiv, Dec. 10, 2015. Accessed: Apr. 12, 2023. [Online]. Available: http://arxiv.org/abs/1512.03385

[20] A. Veit, M. Wilber, and S. Belongie, ‘Residual Networks Behave Like Ensembles of Relatively Shallow Networks’. arXiv, Oct. 26, 2016. Accessed: Apr. 21, 2023. [Online]. Available: http://arxiv.org/abs/1605.06431

[21] S.-H. Tsang, ‘Review: DenseNet — Dense Convolutional Network (Image Classification)’, Medium, Mar. 20, 2019. https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803 (accessed Apr. 21, 2023).

[22] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, ‘Densely Connected Convolutional Networks’. arXiv, Jan. 28, 2018. Accessed: Apr. 21, 2023. [Online]. Available: http://arxiv.org/abs/1608.06993

[23] T. Zhou, X. Ye, H. Lu, X. Zheng, S. Qiu, and Y. Liu, ‘Dense Convolutional Network and Its Application in Medical Image Analysis’, BioMed Research International, vol. 2022, p. e2384830, Apr. 2022, doi: 10.1155/2022/2384830.

[24] ‘PyTorch Metric Learning: An opinionated review.’, Pento. https://www.pento.ai/blog/pytorch-metric-learning (accessed Apr. 21, 2023).

[25] ‘PyTorch Metric Learning’. https://kevinmusgrave.github.io/pytorch-metric-learning/ (accessed Apr. 21, 2023).

[26] D. Crouse et al., ‘LemurFaceID: a face recognition system to facilitate individual identification of lemurs’, BMC Zoology, vol. 2, no. 1, p. 2, Feb. 2017, doi: 10.1186/s40850-016-0011-9.

[27] M. Clapham, E. Miller, M. Nguyen, and C. T. Darimont, ‘Automated facial recognition for wildlife that lack unique markings: A deep learning approach for brown bears’, Ecology and Evolution, vol. 10, no. 23, pp. 12883–12892, 2020, doi: 10.1002/ece3.6840.

[28] ‘Chimpanzee face recognition from videos in the wild using deep learning | Science Advances’. https://www.science.org/doi/10.1126/sciadv.aaw0736 (accessed May 29, 2023).

[29] H. Gong et al., ‘Facial Recognition of Cattle Based on SK-ResNet’, Scientific Programming, vol. 2022, pp. 1–10, Nov. 2022, doi: 10.1155/2022/5773721.

[30] ‘Real-Time Computer Vision | Computer graphics, image processing and robotics’, Cambridge University Press. https://www.cambridge.org/be/academic/subjects/computer-science/computer-graphics-image-processing-and-robotics/real-time-computer-vision, https://www.cambridge.org/be/academic/subjects/computer-science/computer-graphics-image-processing-and-robotics (accessed Apr. 21, 2023).

[31] M. Nadeski, ‘Bringing machine learning to embedded systems’, 2019.

[32] ‘Quantization — PyTorch 2.0 documentation’. https://pytorch.org/docs/stable/quantization.html (accessed Apr. 21, 2023).

[33] ‘Make Deep Learning Models Run Fast on Embedded Hardware’, Apr. 29, 2020. https://www.edgeimpulse.com/blog/make-deep-learning-models-run-fast-on-embedded-hardware (accessed Apr. 21, 2023).

[34] N. Malingan, ‘Quantization and Pruning’, Scaler Topics, May 11, 2023. https://www.scaler.com/topics/quantization-and-pruning/ (accessed May 15, 2023).

[35] Q-engineering, ‘Deep learning with Raspberry Pi and alternatives in 2023 - Q-engineering’. https://qengineering.eu/deep-learning-with-raspberry-pi-and-alternatives.html (accessed Apr. 21, 2023).

[36] ‘How to overclock and stress-test your Raspberry Pi’, ZDNET. https://www.zdnet.com/article/how-to-overclock-and-stress-test-your-raspberry-pi/ (accessed Apr. 21, 2023).

[37] Q-engineering, ‘Overclocking the Raspberry Pi 4 - Q-engineering’. https://qengineering.eu/overclocking-the-raspberry-pi-4.html (accessed Apr. 21, 2023).

[38] P. F. I. the lead author, owner of R. com M. goal is to help you with your R. P. problems using detailed guides, tutorials I. real life, and I. a L. system administrator with a web developer experience, ‘Raspberry Pi Temperature: Limits, monitoring, cooling and more – RaspberryTips’. https://raspberrytips.com/raspberry-pi-temperature/ (accessed Apr. 21, 2023).

[39] ‘Deep Learning on a USB Stick’, Hackster.io. https://www.hackster.io/news/deep-learning-on-a-usb-stick-29c117cf93e2 (accessed May 15, 2023).

[40] ‘Movidius: A stick for neural-networks from Intel for $79.’, Elektor, Jul. 27, 2017. https://www.elektormagazine.com/news/movidius-a-neural-networking-stick-from-intel-for-79 (accessed May 15, 2023).

[41] ‘USB Accelerator’, Coral. https://coral.ai/products/accelerator/ (accessed May 15, 2023).

[42] ‘Google Coral USB Accelerator performance with Raspberry Pi 3B, 3A+, 4B’. https://helloworld.co.in/article/google-coral-usb-accelerator-performance-raspberry-pi-3b-3a-4b (accessed May 15, 2023).

[43] ‘Jetson Nano’, NVIDIA Developer, Mar. 06, 2019. https://developer.nvidia.com/embedded/jetson-nano (accessed May 29, 2023).

[44] ‘JetPack SDK’, NVIDIA Developer, Oct. 14, 2014. https://developer.nvidia.com/embedded/jetpack (accessed May 29, 2023).

[45] ‘AI and Machine Learning Tools with Raspberry Pi 4’, Pi Australia. https://raspberry.piaustralia.com.au/blogs/ai-and-machine-learning/ai-and-machine-learning-tools-with-raspberry-pi-4 (accessed May 15, 2023).

[46] D. Franklin, ‘Deploying Deep Learning’. May 15, 2023. Accessed: May 15, 2023. [Online]. Available: https://github.com/dusty-nv/jetson-inference/blob/bd4098c7a3c9ea5d912356a3cc02409421cf3c15/docs/pytorch-cat-dog.md

[47] ‘Machine Learning: Using Tensorflow on the Raspberry Pi - Blog - Artificial Intelligence and Machine Learning - element14 Community’, Mar. 25, 2020. https://community.element14.com/technologies/ai-machine-learning/b/blog/posts/machine-learning-using-tensorflow-on-the-raspberry-pi (accessed Apr. 21, 2023).

[48] R. P. Ltd, ‘Buy a Raspberry Pi 3 Model B+’, Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/ (accessed Apr. 21, 2023).

[49] ‘MobileNetV2 SSD FPN’. https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/mobilenetv2-ssd-fpn (accessed Apr. 21, 2023).

[50] ‘Mean Average Precision (mAP) Explained: Everything You Need to Know’. https://www.v7labs.com/blog/mean-average-precision (accessed Apr. 21, 2023).

[51] BenjaminVierstraete2, ‘Automated dog door using facial recognition technology’. Jan. 28, 2023. Accessed: Apr. 21, 2023. [Online]. Available:

https://github.com/BenjaminVierstraete2/Facial_Recognition_dogdoor/blob/3f5c06e3357afc1e74d9c49fd434b07cfc95c5e2/FaceRecognition/models/model_plot.png

[52] ‘Classical conditioning’, Wikipedia. Apr. 10, 2023. Accessed: Apr. 21, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Classical_conditioning&oldid=1149093162

[53] ‘OpenCV: Background Subtraction’. https://docs.opencv.org/4.x/d8/d38/tutorial_bgsegm_bg_subtraction.html (accessed Apr. 21, 2023).

[54] ‘FOMO: Object detection for constrained devices’. https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices (accessed Apr. 21, 2023).

[55] B. Dickson, ‘FOMO is a TinyML neural network for real-time object detection - TechTalks’, Apr. 18, 2022. https://bdtechtalks.com/2022/04/18/fomo-tinyml-object-detection/ (accessed Apr. 21, 2023).

[56] 60 fps Object Detection on Raspberry Pi 4 with FOMO, (Mar. 28, 2022). Accessed: Apr. 21, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=o2-o3wEmxaU

[57] ‘Difference of Gaussians’, Wikipedia. Jan. 27, 2023. Accessed: Apr. 21, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Difference_of_Gaussians&oldid=1135968382

[58] ‘OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)’. https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html (accessed Apr. 21, 2023).

[59] D. Tyagi, ‘Introduction to SURF (Speeded-Up Robust Features)’, Data Breach, Apr. 07, 2020. https://medium.com/data-breach/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e (accessed Apr. 21, 2023).

[60] ‘Americans Note Overwhelming Positive Mental Health Impact of Their Pets in New Poll; Dogs and Cats’. https://www.psychiatry.org:443/news-room/news-releases/positive-mental-health-impact-of-pets (accessed May 29, 2023).

[61] ‘Why is Face Alignment Important for Face Recognition?’ https://www.analyticsvidhya.com/blog/2022/07/why-is-face-alignment-important-for-face-recognition/ (accessed Apr. 21, 2023).

[62] ‘DeepFace: Closing the Gap to Human-Level Performance in Face Verification - Meta Research’, Meta Research. https://research.facebook.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/ (accessed Apr. 07, 2023).

8 Attachments

8.1 Report IBM

On the 18th of January, I attended a guest lecture by Stefan Van Den Borre from IBM, IBM is one of the world's largest and oldest information technology companies. It has a significant presence in various sectors, including hardware, software, cloud, artificial intelligence (AI) and quantum computing.

The talk was on the topic of AI ethics. This refers to the principles and guidelines that govern the responsible development and use of artificial intelligence (AI) systems. It involves considering the ethical implications of AI technology and ensuring that it is developed and deployed in a manner that aligns with social values, respects human rights, and avoids harm.

When looking at fairness for example, he mentioned that fairness is not a fixed concept. It varies depending on different perspectives, contexts, and cultural norms. What is considered fair in one situation or society may not be viewed as fair in another. In the context of artificial intelligence and algorithmic decision-making, fairness becomes even more complex. Determining what is fair in algorithmic systems involves addressing biases. Bias and fairness are related so by maximizing fairness the biases change.

The talk mentioned an article which is a good example of this: “Google ‘fixed’ its racist algorithm by removing gorillas from its image-labelling tech”.

The article talks about a image recognition algorithm misclassifying black individuals as gorillas, Google's initial response was a promise to fix the problem thereby demonstrating a recognition of the need for more fairness. However, the action eventually taken was stopping the algorithms from identifying gorillas altogether. This approach might be viewed as a trade-off between mitigating the bias and restricting the functionality of the service.

Fairness also has many definitions, in the talk the presenter showed a list of different accepted definitions. Things like upbringing can change an individuals perception of fairness, the question then is what definition should AI researchers be focused on?

The following interesting topic was drift, which was still an unheard term for me at the time. Drift meaning the AI system is used for a different purpose then it was initially trained for. In the context of fairness this can mean that the model initially trained to be fair. This can lead to unfairness to the new data. To Address unfairness from drift would require ongoing monitoring and evaluation of the system and continuous efforts to update and retrain the model with new data to ensure that it remains aligned with evolving fairness criteria.

One way companies handle fairness is by training unaware models, meaning the sensitive data is not included in the training process. For example not including gender or race. A drawback of this is the loss of potentially valuable data.

IBM has developed a Toolkit that can help determine and examine the fairness of machine learning models. Some metrics are used for this:

- The statistical parity difference is the difference in the rates of favourable outcomes received by different groups.
- The equal opportunity difference is the difference in the true positive rates between different groups.
- The Average Odds Difference is the disparity in both the false positive rate and true positive rate between different groups.
- Disparate Impact is the ratio of rate of favourable outcome between different groups.

The following point made is that its hard to know on what bases an AI model makes decisions. For example a car prediction model trained solely on images of cars on a white background could interpret a white background as a car. Therefor classifying all images with white backgrounds as cars.

To understand the decision process of the model we need tools to help AI explain ability. IBM also developed an opensource toolkit for this. The two techniques used for this are LIME and Shapley values. LIME offers local explanations by simplifying complex models, while Shapley values attribute feature contributions fairly. Both methods enhance understanding, address bias, promote transparency, and foster trust in AI systems.

In the closing part of the presentation some examples were given for security risks, for example adding noise to an image without visually changing what is being imaged can trick googles inception v4 model to classify a snail as a mitten. In the context of things like autonomous driving this can rapidly turn dangerous. For example adding stickers to important traffic signs can make a car continue instead of stopping.

The final ethical question was who should a model save in a modified version of the trolley problem. Given the choice to either drive over 2 small children, an elderly man, and a wall. What should the model decision be. And where does the responsibility lay, the model, the driver, the developer?

8.2 Report SupportSquare

On the 27th of January, Mathieu Allert delivered an insightful talk centered around his extensive experience as an XR developer. While a significant portion of his presentation revolved around his educational background and professional journey, several valuable lessons could be learned from his talk. This report will delve into the key takeaways from Allert's talk, emphasizing the challenges associated with developing and bringing cutting-edge products to market, as well as the importance of mitigating risks and ensuring ethical business practices.

One of the fundamental lessons highlighted by Allert was the inherent complexity involved in developing and launching innovative products. He emphasized that such products often come with substantial costs, necessitating the need of high-paying customers to ensure financial sustainability. Moreover, the development of cutting-edge products inherently carries an element of risk, as market demand and acceptance can be unpredictable. To alleviate these challenges, Allert recommended diversifying income sources, such as creating fast proof-of-concept (POC) products or leveraging existing, well-established products for revenue. By doing so, developers can mitigate the risk of working without compensation and maintain a more stable financial foundation.

However, Allert cautioned against viewing POCs as the ultimate solution or final product. While POCs serve as valuable prototypes to demonstrate the feasibility of an idea, they may not always translate into full-fledged products. In some cases, the cost associated with developing the final product may deter further investment, especially if the market demand does not justify the expenses. Therefore, developers should be cautious and carefully evaluate the potential return on investment before proceeding with the development of a product based solely on a successful POC.

Another crucial aspect highlighted by Allert is the importance of including promises into contracts. In the world of development, where collaboration and partnerships are common, it is essential to have clear contractual agreements in place to protect all parties involved. By explicitly defining expectations, deliverables and timelines within a contract, developers can mitigate the risk of clients returning on their commitments. This proactive approach ensures that all stakeholders are held accountable and minimizes the potential for disputes or misunderstandings.

Furthermore, Allert touched upon the topic of crowdfunding as a potential avenue for start-ups in the XR industry. While crowdfunding platforms can provide valuable financial support and early validation for innovative projects, ethical considerations must be taken into account. It is crucial for developers to deliver on their promises and fulfil the expectations of their backers. Failure to do so can lead to reputational damage and legal complications. Therefore, while crowdfunding can be a viable option, it is essential for developers to carefully assess their ability to deliver on their proposed project before engaging in such campaigns.

8.3 Installation Manual

Hardware:

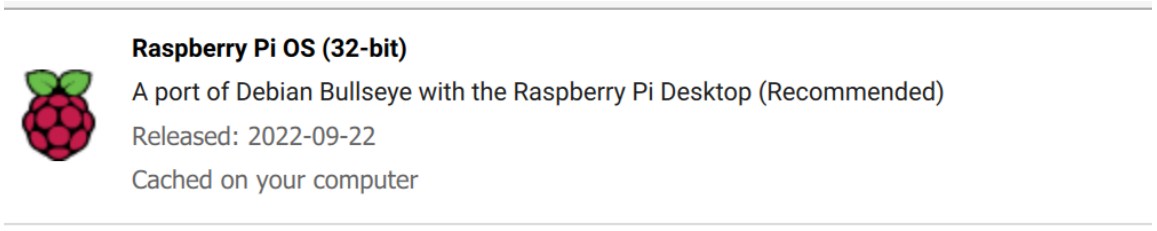
- Raspberry pi 3B+ or Raspberry pi 4
- Raspberry Pi Camera Module 2
- Micro sd card 16gb minimum
- 2 micro servo motors
- Suitable power supply (I used 5v breadboard supply)

Raspberry pi image setup:

This manual uses Raspberry Pi imager for simplicity and configurability

Insert your micro SD card into the SD card slot of a computer, then launch the Raspberry Pi Imager. Note that using this tool will erase any existing data on the SD card.

Choose following operating system:

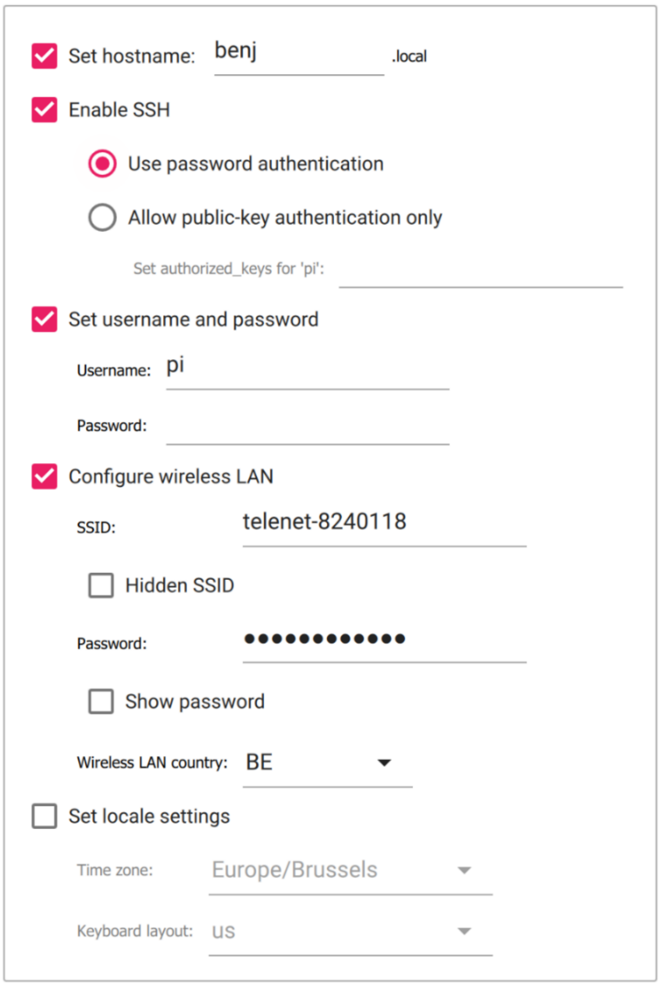


Raspberry Pi OS (32-bit)
A port of Debian Bullseye with the Raspberry Pi Desktop (Recommended)
Released: 2022-09-22
Cached on your computer

Once the Raspberry Pi Imager is open, select your SD card from the list of available devices. Then, click on the settings gear icon to access additional options.

It is crucial to enable SSH by checking the corresponding box as we will need it shortly. Additionally, set a new username and password for your device.

You may also choose to add a wifi connection or change the hostname if desired. Once you have made your selections, proceed to write the image to your SD card by clicking on the "Write" button.



- [✓] Set hostname: benj .local
- [✓] Enable SSH
  - [●] Use password authentication
  - [○] Allow public-key authentication only
  - Set authorized_keys for 'pi':
- [✓] Set username and password
  - Username: pi
  - Password:
- [✓] Configure wireless LAN
  - SSID: telenet-8240118
  - [ ] Hidden SSID
  - Password: ●●●●●●●●●●●●●
  - [ ] Show password
  - Wireless LAN country: BE
- [ ] Set locale settings
  - Time zone: Europe/Brussels
  - Keyboard layout: us

Raspberry pi setup:

Before proceeding with the connection, it is important to ensure that certain functionality is enabled on your Raspberry Pi. To do this follow next steps.

1. Connect an Ethernet cable from your computer to the Raspberry Pi.

2. Log in to the device using the following command in windows command promt:

   ssh <username>@<hostname>.local -p 22

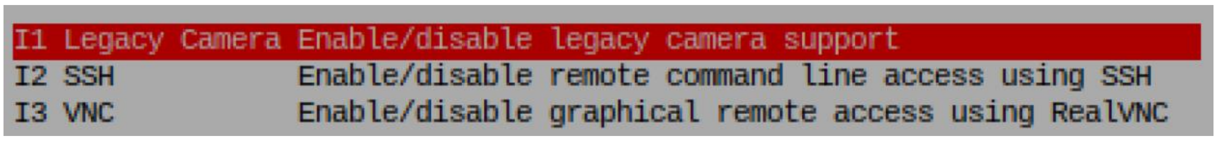
   Make sure to replace username with the correct username and hostname.local with the correct hostname of your Raspberry Pi.

3. Once logged in, enter the command:

   sudo raspi-config

   This will open the Raspberry Pi Configuration tool.

4. Go to the "Interfacing Options" and select the "Camera" and "VNC" options. Make sure to enable both.



```
I1 Legacy Camera Enable/disable legacy camera support
I2 SSH           Enable/disable remote command line access using SSH
I3 VNC           Enable/disable graphical remote access using RealVNC
```

5. After enabling the options, reboot your device for the changes to take effect. Now you can use the camera and vnc on your raspberry pi.

To connect to your Raspberry Pi, you have two options:

- Use a VNC viewer to remotely access the Pi's desktop.
- Connect the Pi to a screen using an HDMI cable to access the desktop directly.

Once you have access to the desktop, open the terminal and proceed with the next set of commands.

1. Update the pi:

   sudo apt-get update

   sudo apt-get dist-upgrade

2. clone needed code and files:

   git clone https://github.com/BenjaminVierstraete2/DogDoor_Pi.git

3. if u want u can rename it for easy typing:

   mv DogDoor_Pi <newname>

4. go to folder:

   cd DogDoor_pi

Optional:

install virtual env to allow the project to run in its own environment, will reduce chances of package conflicts if u have anything else on your pi, if not installing venv skip over next 3 commands.

```
sudo pip3 install virtualenv
python3 -m venv venv
source venv/bin/activate
```

5. install all needed packages:

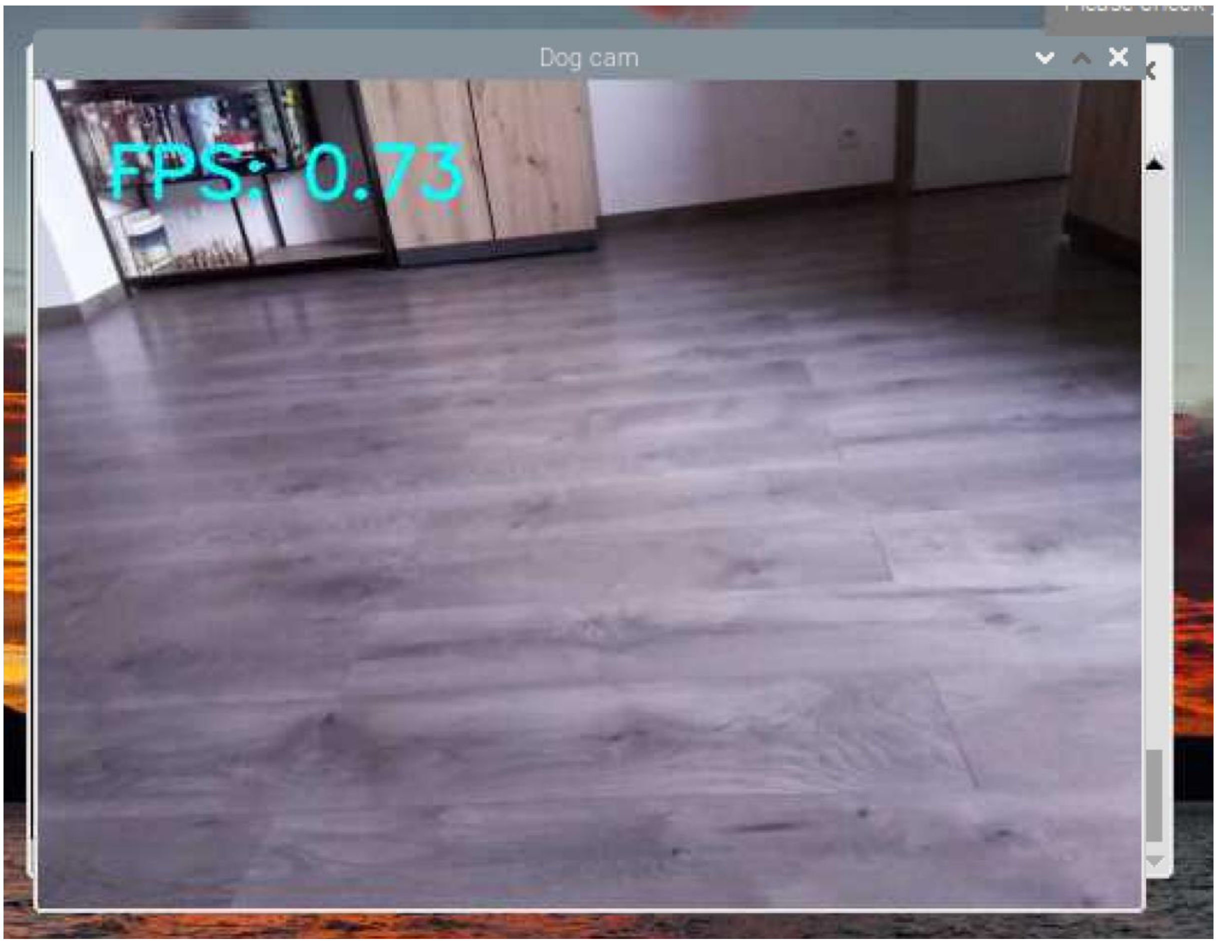
   bash requirements.sh

Test setup :

Run model to see if there are no errors:

python3 detect.py --view

this should give u a popup window like this:



Automate:

To make it truly embedded we need to automate it, meaning when u plug in your pi the detection starts.

We will do this by creating a service that will start on bootup

1. Open the terminal and enter the following command to create a service called "detect.service": **sudo nano /etc/systemd/system/detect.service**

2. Next, add the following lines to the file, then save it:

```
[Unit]
Description=Runs detection script on boot
[Service]
Type=simple
User=<username pi>
WorkingDirectory=/home/<username>/DogDoor_Pi
ExecStart=/bin/bash -c "source venv/bin/activate && python3 detect.py"
Restart=always
[Install]
WantedBy=multi-user.target
```

Remember to change "username" to your own username. If you did not create a virtual environment, you can remove the **source venv/bin/activate &&** part of the ExecStart line.

3. Now simply reload systemctl

   sudo systemctl daemon-reload

4. enable to run on boot:

   sudo systemctl enable my-script.service

5. test by either rebooting or running following:

   sudo systemctl start detect.service

Disable automation:

The service you created will start a no view version of the detect script that will run in the background of your pi.

if u want to view the detections live u can boot the pi and run **sudo systemctl stop detect.service** in terminal to stop the service and then run it manually using the –view argument

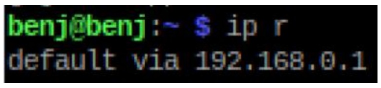
Static Ip

To remotely access the Raspberry Pi after the detection script has been installed, you will need to set a static IP address. This will allow you to connect to the Pi via SSH without the need for a HDMI or Ethernet cable. Setting a static IP address will ensure that the Pi always has the same IP address, making it easier to connect to it remotely.

Note: u will need to add a Wi-Fi network if not done in the image setup fase.

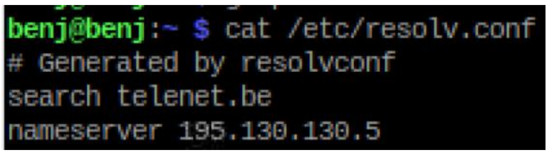
1. first we need to know your routers ip address open terminal and enter:

   ip r

   ```
   benj@benj:~ $ ip r
   default via 192.168.0.1
   ```

2. next we need to know our dns address:
   cat /etc/resolv.conf

   ```
   benj@benj:~ $ cat /etc/resolv.conf
   # Generated by resolvconf
   search telenet.be
   nameserver 195.130.130.5
   ```

3. go to the DHCP client configuration file using next command:

   sudo nano /etc/dhcpcd.conf

4. add following lines to the bottom of the file

   interface wlan0
   static ip_address=<ip address of choice>/24
   static routers=<routers adress>
   static domain_name_servers=<dns address>

   change the values in <> to the your values.

5. Reboot to apply changes

8.4 User manual

Automatic detection

If the Raspberry Pi was set up correctly according to the instructions in the installation manual, the only task remaining is to plug it in.

Manual detection

If you wish to run the model manually, you will need to connect to your Raspberry Pi using SSH or VNC Viewer for the desktop. The static IP you set up should be used for this purpose. To stop the automatic running of the model when the Raspberry Pi boots, you can open the terminal and enter the command:

```
sudo systemctl stop detect.service
```

Then navigate to the directory where the detect script is located by using the command:

```
cd <dir>
```

If you are using a virtual environment, you will need to activate it by running the command:

```
source <venvname>/bin/activate
```

Once the virtual environment is activated, you can run the script manually.

Running detect

The default command to run the script is

```
python3 detect.py
```

The detect script takes a few different arguments, below is a list of what they do and their default values.

Threshold:

The first argument for the script is the threshold value. The threshold represents the level of confidence required for the model to detect a face. This is used by the initial model that detects the faces of dogs. the face will only be used for recognition when the models confidence in detecting a face is higher than the threshold value. The default threshold value is 0.9 or 90% confidence. This means that by default, the model will only consider a face as detected if it is 90% confident that it is a face.

Distance:

The second argument for the script is the distance value. This value is used by the second model to determine if it recognizes the dog or if it is an unknown face. This model compares the detected face to all the faces in the database for each dog, and calculates the minimum distance (closest face) and the dog it belongs to. It will only consider the face to belong to that dog if the distance is smaller than this value. The default distance value is 0.1, which means that by default, the model will consider a face as belonging to a specific dog only if the minimum distance between the detected face and the faces in the database for that dog is less than 0.1.

Resolution:

The third argument for the script is the resolution value. It takes a resolution in the format of "widthxheight" and uses this resolution to stream the live video on the desktop. The default resolution is set to 576x432, which is a relatively low resolution. If you use an HDMI cable, you can increase this resolution to a higher value, such as 720x480 or 1080x720 for example. This will allow you to view the live video in a higher quality, but it will also require more processing power and bandwidth. You can adjust the resolution according to your needs and the capabilities of your Raspberry Pi and monitor.

View:

The fourth argument for the script is view. This argument determines whether the camera will be streamed or not. If this argument is not set, no video will be streamed and the detection will happen in the background. This means that the script will run without displaying a live video feed. This can be useful if you want to save resources on the Raspberry Pi or if you don't need to see the live video.

Unlike the other arguments this does not take a value, simply add –view to the command to show video.

Allowed:

The fifth argument for the script is the list of dogs that the recognition model will detect. This argument takes a list of one or more dog names separated by commas. The recognition model is trained on my personal dogs named Marley, Muchu and Ellie, so by default these dogs are in the allowed list. If you want the model to detect only specific dogs, you can provide the names of those dogs as the argument. For example, if you only want the model to detect Marley and Muchu, you can provide the argument "Marley,Muchu".

Opentime:

The sixth and last argument for the script is the time that the door must be open after the allowed dog is detected. This argument takes a value in seconds, and the default value is set to 10 seconds for testing purposes. In a real-world use case, you might want to set this value higher to allow the dog plenty of time to enter the home.

Below is an example of how u could change these arguments in the command.

In the example I call all arguments but u can leave out any u like, also keep in mind to type 2 dashes before each argument.

```
python3 detect.py –threshold=0.8 –distance=0.5 –resolution=720x480 –view –
allowed=muchu,elllie –opentime=50
```