UNIVERSITY OF TARTU

INSTITUTE OF COMPUTER SCIENCE

COMPUTER SCIENCE CURRICULUM

Villem Susi

Real-time Pet Identification Using Computer Vision On A Raspberry Pi

Bachelor's Thesis (9 ECTS)

Supervisor: Karl Kaspar Haavel, MSc

Tartu 2024

Real-time Pet Identification Using Computer Vision On A Raspberry Pi

Abstract:

This work investigates using deep convolutional neural networks to detect and identify domestic pets using camera-captured images on an edge device - Raspberry Pi. To do this, we use a pre-trained YOLOv5 to detect pets and a fine-tuned EfficientNet on a modified dataset for the identification of specific pets. Furthermore, a user interface is created to add additional pets, view logs, fine-tune the model and interact with the device. We describe a custom dataset of internet-sourced and author-provided cat and dog images. This work obtains three F1 scores with three different datasets on the identification task: 0.25, 0.67 and 0.84. The results highlight a need to improve the reliability of the identification process.

Keywords:

Image classifier, machine learning, transfer learning, pets

CERCS: P176 Artificial intelligence

Reaalajas lemmiklooma tuvastamine raalnägemisega *Raspberry Pi* peal

Lühikokkuvõte:

See töö uurib süvaõppega koduloomade tuvastamist ja identifitseerimist kaameraga tehtud piltidel kasutades *Raspberry Pi*d. Et seda saavutada, kasutatakse eeltreenitud YOLOv5 mudelit et loomi tuvastada ning täiendatud *EfficientNet* mudelit spetsiifiliste isendite identifitseerimiseks. Lisaks luuakse graafiline veebiliides, millega saab loomi lisada, logisid vaadata, mudelit edasi täiendada ning *Raspberry Pi*ga interakteeruda. Kirjeldatakse internetist ja autori erakogust kombineeritud andmestikku kassi ja koera piltidest. Selles töös saavutatakse kolm F1 tulemust kolmel erineval andmestikul identifitseerimis ülesandel: 0.25, 0.67 ning 0.84. Need tulemused peegeldavad vajadust parandada identifitseerimis protsessi usaldusväärsust.

Võtmesõnad:

Pildiliigitaja, masinõpe, siirdeõpe, lemmikloomad

CERCS: P176 Tehisintellekt

Table of Contents

Introduction

Human facial recognition with deep learning has advanced significantly in recent years. However, there hasn’t been as much focus on deep learning in the context of animal identification on a specimen level.

There are a lot of household pets who are used to spending their time outside of the house during the day or night of their own volition. Installing a door flap as their entry and exit point could pose a security threat, as it is accessible by anyone passing by. Constantly letting the pet in or out of the house might turn into an annoyance for the owner. This work proposes a camera-based identification software that operates a smart door which reduces the security implications of a door flap and automates the door opening process. We aim for an F1-score of 0.8 in the identification task and for the negative class to be predicted negative 100% of the time.

Existing similar literature is focused mainly on deep learning approaches with seemingly unlimited data. This work tries to unlock similar potential from as little training data as possible, as every user might not have an unlimited number of images of their pet.

The thesis has been organised into three major sections: Background, Method and Results. In the background section of the work, the author gives a general overview of the related topics and the related works in pet identification. In the method section, the author describes the steps taken to develop the resulting interface and the use of YOLOv5 for pet detection and EfficientNet for pet identification. In the results section, the author showcases the achieved results, compares these to the targeted goals and describes the details of the final interface and identification network.

1 Background

This chapter gives an overview of supervised learning from the classification perspective and how convolutional neural networks (CNN-s) can be used for it. After this, the focus shifts to the basics of computer vision, its usage in facial and object detection, and how the latter differs when applied to humans versus animals. Further, a summary is made about the concept of a Raspberry Pi and the intricacies of image classification on low-resource devices. Lastly, similar related works on pet recognition are opened.

1.1 Supervised Learning

Supervised learning in the context of machine learning means models that are trained with entirely correctly labelled data. This means that for each data point that's given to the model for training, the model would know with certainty, which output it should present for this input. This causes the model to train more effectively and for better accuracy. The main negative aspect of supervised learning is the need for the existence of labelled data. To obtain labelled data, unlabelled data must be labelled manually or automatically labelled data must be checked manually to confirm its correctness. Incorrect labelling has a high probability of resulting in an ineffective model, in which case the model might be unusable [1].

Classification in the context of machine learning is a process, during which a model attempts to predict the label of previously unseen data points. This is done by comparing the new data points against previously seen data and looking for similar patterns. Classification can be performed both supervised and unsupervised.

True positive rate (TPR), accuracy, precision, recall and F1-score are metrics that help evaluate a machine learning model's performance. In the following definitions, TP (true positive) is the number of times the positive class was predicted correctly, FN (false negative) is the number of times the negative class was predicted incorrectly, TN (true negative) is the number of times the negative class was predicted correctly and FP (false positive) is the number of times the positive class was predicted incorrectly [2].

$$TPR = \frac{TP}{TP + FN}$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$f1\_score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

1.2 Convolutional Neural Networks

Convolutional neural network (CNN) is a machine learning method which is used mainly for pattern recognition in images [3]. This is because CNN-s are effective in detecting patterns in large amounts of data points (an image consists of approx. thousands to millions of pixels, each of which might contain three values - one for each of the RGB colour values). For example, pinpoint the lighter and darker areas, distinguish contour lines etc. It is difficult to train a non-convolutional network to do pattern recognition in images because the relevant objects can have different locations, rotations, sizes and colours on individual images.

To learn the features of an image, the CNN uses a square matrix called a kernel or a filter which has a specific set of numbers in it (see Fig. 1). The kernel is moved step by step over the matrix of pixels of an image. At every step, the image pixel values are multiplied by the kernel values and added together. The resulting values are added to a new matrix (see Fig. 2). This new matrix is then used by the next convolutional layer. Since this matrix can contain negative numbers, a ReLU function can be used which converts every negative number to a zero to reduce the computational requirements [4]. CNN-s use pooling layers which also reduce the computational power necessary to train the network by reducing the size of the convolutional layers output matrix. This is also done by a kernel moving across the matrix taking either the maximum or average of the current pixels for the new value [5].

The alternating use of convolutional and pooling layers for feature learning and then classification layers describes the usual structure of a CNN.

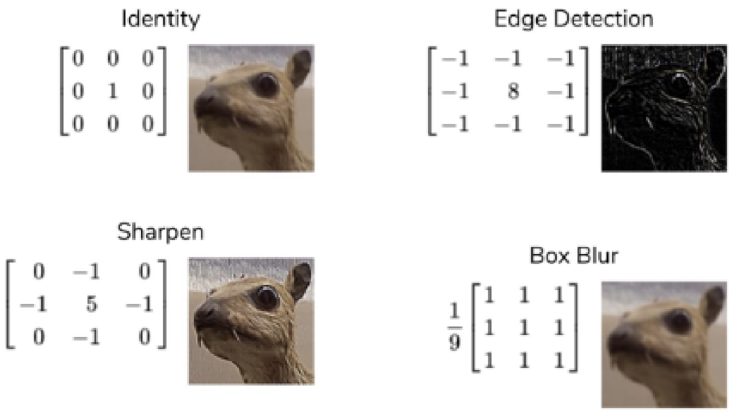


Figure 1. Examples of kernels used for different use cases [5].

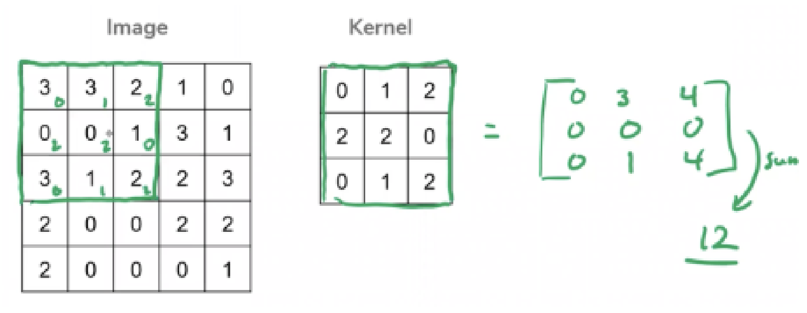


Figure 2. Convoluted feature extraction matrix operation [5].

CNN-s don’t have to be trained fully from scratch but can also be improved upon. This is called transfer learning. Transfer learning is a process in machine learning which helps fine-tune existing models for more specific use. It consists of loading the pre-existing model, freezing its layers so the training doesn’t affect those, adding new layers on top of the existing ones and finally training the new layers with new data [1]. The last step is fine-tuning which means unfreezing the entire model and training it with a low learning rate.

This work later mentions two terms: an epoch and an optimiser.

An epoch in machine learning is one cycle of training where data passes through the training algorithm.

An optimiser is an algorithm that tweaks the machine learning model in some predefined manner to achieve better results.

1.3 Facial Recognition

Facial recognition is a widely common phenomenon in today's society. Its goal is to identify a person based on the characteristics of their face.

One of the options to check, whether a person matches a given profile, is to analyse that person's facial geometrical features and compare these to the features of the given profile. For example, the distance between the eyes, the height of the face, facial shape, the positioning of the mouth, the nose and the ears to one another etc. (see fig. 3). By combining the results of these features a theoretically unique dataset is forged. This can be used to check whether it matches the original profile. The described methodology requires a direct view of the face in question for it to be possible to analyse every geometrical detail with high enough confidence. However this is very difficult in practice if the face in question does not belong to a person but instead a freely moving animal, since it might be difficult to motivate the animal to point its face straight towards the camera [6].

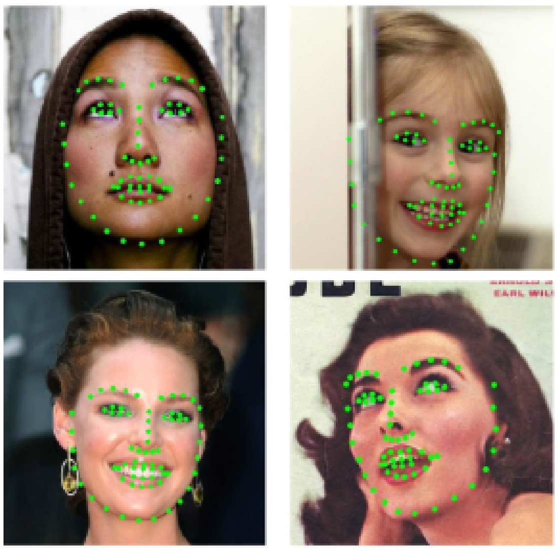


Figure 3. Facial point detection visualisation [7].

Another option to perform facial recognition is a deep learning approach. In this case, the problem is essentially an image classification challenge. For this verification, a neural network is trained with a lot of images of faces [8]. This network is then used to predict which face the given face is most similar to. This option has the significant disadvantage of requiring much more data for the initial training.

1.4 Neural Network Object Detection

Object detection is the process of identifying one or more objects in an image. The main idea consists of image classification and/or object localisation which when combined is called object detection. Object detection can be further developed to do image segmentation.

Image classification is the process of labelling an image based on the contents of that image and previously seen images of similar objects. For example, a picture of a table can be labelled as a “table” if the model was trained on images, of which some were also labelled “table”. The level of sophistication for the labelling depends entirely on the data that was used for training the model. This means that if the model saw images labelled “cat” it can only predict new images with the label “cat”. But if the training data distinguished a Maine Coon from a British Shorthair, then the model can also in theory make that distinction.

Object localisation is the process of identifying the presence of an object in an image and locating the rectangular smaller part of the image which contains that object, called a bounding box. The exact label of the identified object is not important in this case [9].

Object detection is the combination of image classification and object localisation. Its main idea is to locate and classify all of the objects of interest in an image. An object detection model takes an image as an input and outputs the coordinates of the bounding boxes for each object of interest in that image with the labels of those objects attached to the bounding boxes. This can be done in two ways: one-stage and two-stage detection. Two-stage detection methods prioritise the accuracy of the resulting bounding box and label by first estimating the general area of the objects in question and only then classifying these areas and identifying/refining the exact bounding boxes. Doing this in two steps results in the process taking a longer time but yielding better results. One-stage methods perform the latter without the initial general area proposal resulting in better speed but sacrificing accuracy [10].

One such one-stage option is the CNN-based You Only Look Once (YOLO) model, which analyses the image as a grid and labels each cell in that grid as the object that's most likely in that cell (see Fig. 4). Using that knowledge the YOLO model then estimates the bounding boxes of each object. YOLO has very fast real-time detection, being able to process images at 45-155 frames per second, depending on the size of the network variant and the capabilities of the hardware it runs on [11]. YOLO has multiple versions with increasing performance and size, the latest being YOLOv8[1].

[1] https://docs.ultralytics.com/ Visited 24.01.2024

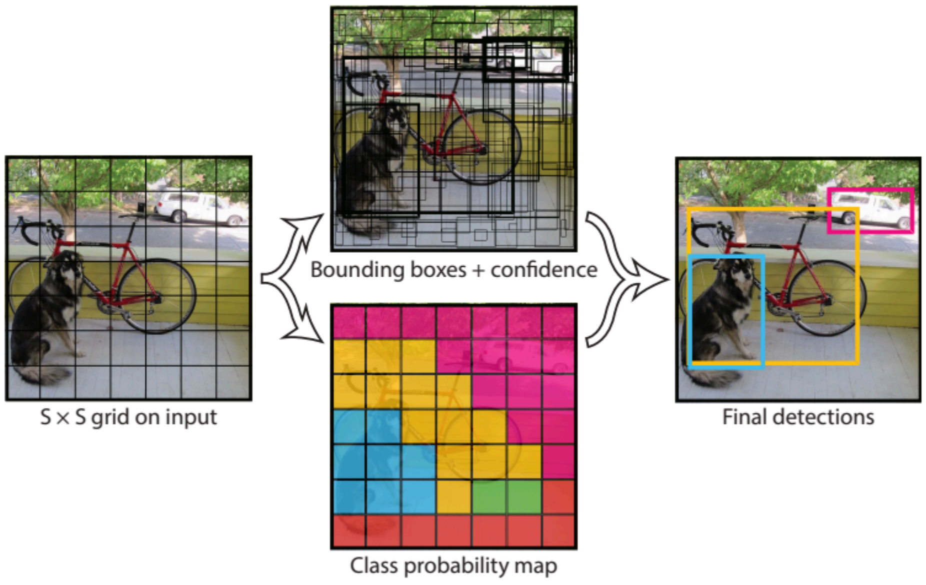


Figure 4. YOLO detection and localisation logic [11].

1.5 Raspberry Pi

Raspberry Pi is a low-cost, small-sized computer board. It has IO ports as well as GPIO pins which make it possible to use the Raspberry Pi with a computer monitor, a keyboard, and a mouse and control all manner of electronic components [12]. Raspberry Pi by default uses a Debian-based variant of the Linux operating system formerly called the Raspbian but known now as the Raspberry Pi OS[2] and has built-in software to aid in the initial setup. Raspberry Pi has many versions with varying computational power and number of ports, the most current being Raspberry Pi 5.

A Raspberry Pi is perfect to use as an IoT or an Edge device [12]. It is often more useful to complicate the system architecture with an Edge device and by doing so reduce the amount of data sent over the network. An autonomously operating device is also more secure, efficient and flexible [13]. The low cost of these devices limits the impact of losing or destroying one as it makes it cheaply replaceable [14].

1.6 Related Works

Optimising pet identification on a specimen level has also been attempted by others with different approaches and varying results. The following highlights some of those approaches.

According to Lan [15], one option is to first find the face of an animal using object detection methods and then do image classification on the cutout image of the face using a model

[2] https://www.raspberrypi.com/software/ Visited 26.01.2024

specifically trained on animal faces. This can be useful to reduce the unnecessary noise in the image, aiding the classification model. Lan first used the YOLOv3[3] model for finding the face boxes and then used five different models specifically trained on cat faces to accurately estimate the species of that cat. The best results (88.9% accuracy) were obtained with an implementation of the EfficientNet[4] model using the support vector machine (SVM) algorithm. EfficientNet is a CNN that uniformly scales all dimensions of the network with a constant coefficient, whereas other CNN-s do so with an arbitrary coefficient. This results in EfficientNet being efficient. SVM is an approach in machine learning that finds the best plane between data points in a n-dimensional coordinate system to separate one class from another. Using that plane, new data points can be classified by observing on which side of that plane the data point resides.

Klein [16] used transfer learning to train an implementation of YOLOv3 to detect the faces of cats. For identification and verification, Klein used transfer learning on a CNN based on the EfficientNetB2 model which was originally trained on ImageNet. For validation, Klein used 10,000 pairs of cat faces to verify whether or not the cats in both images of the pairs are the same specimen. This validation resulted in 95% accuracy and the identification task resulted in 81% accuracy.

Azizi et al. [2] collected data from various sources including pet owners. They used approximately 50,000 images over 7 datasets for training and testing. They performed analysis separately on the face and body, doing facial landmarking for the faces and feature extraction for both. They achieved approximately 98% accuracy in testing on the Asirra dataset from Kaggle.

[3] https://arxiv.org/abs/1804.02767/ Visited 03.12.2023
[4] https://arxiv.org/abs/1905.11946 Visited 03.12.2023

2 Method

This chapter describes the choices and the reasoning behind them that were made during the development of the software resulting from this work. It opens the model structure selection, data collection and augmentation, model training and the setup of the Raspberry Pi for user interfacing.

2.1 Data

To help the model predict with a high enough accuracy, it has to be trained on a large enough dataset. In practice, it can be difficult to obtain such data and for a generalised model, thus the dataset has to be artificially enlarged. This is called data augmentation.

2.1.1 Data Augmentation

Data augmentation is a process where new data is artificially generated based on existing data. It is a useful method to increase the generalisation capacity of a model by solving the problem of data scarcity [17]. To augment a dataset containing images it is possible to do the following: rotate, cut, tone, blur the images, remove areas from the images, merge multiple images into one etc. Any of the previously mentioned methods can also be combined. Shorten et al. [18] stated that for each method it is essential to consider the impact it may have on a model when trained. For example regarding number image classification, a rotated image of the number 6 might look like an image of the number 9 and be therefore incorrectly classified. For images of animals, most of these methods are safe to apply, meaning that the resulting image is most likely similar enough to the original to be correctly classifiable. The only exception is the merging of images, since this may result in images containing the features of multiple species and be therefore incorrectly classified. These images would have to be manually checked which takes a lot of time. The following paragraphs explain the different techniques most commonly used for image augmentation.

Transformations

Rotating the image to varying degrees, scaling, mirroring and translating are the core geometric transformations applied to images. In the context of animal images, all of these are relatively safe operations since no features are lost as it is normal for an animal to hold different poses on images. An example of applying geometric transformations on an image is seen in Figure 5.

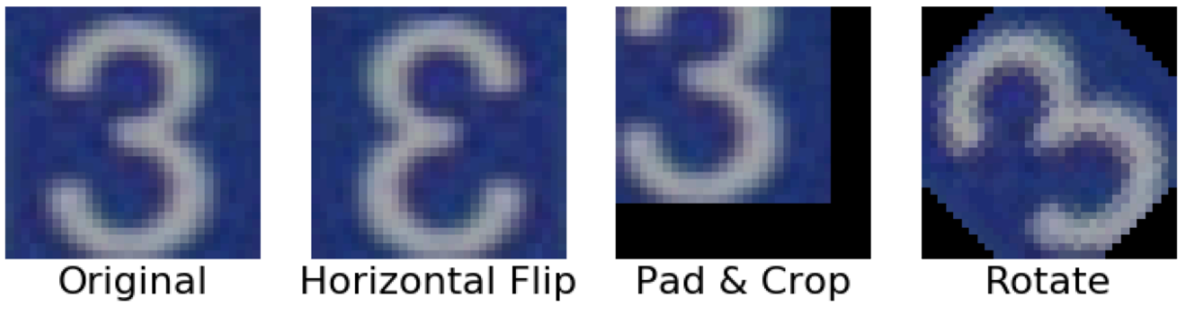


Figure 5. Example of simple geometric transformations [19].

Toning

Toning an image means changing the colour values of each individual pixel in the image in some way. For example, to make an image more red, the red value of each pixel should be increased. For this technique, the context also matters. For more generalised classifiers it might be sensible to convert the image into black and white or greyish tones to reduce the computational power necessary to train on that image. However, this will reduce the level of detail in the image and might reduce the level of specificity at which the model can be trained. For example, an animal classifier might require coloured images because colour can be a defining factor when comparing animals. An example of toning augmentations is seen in Figure 6.

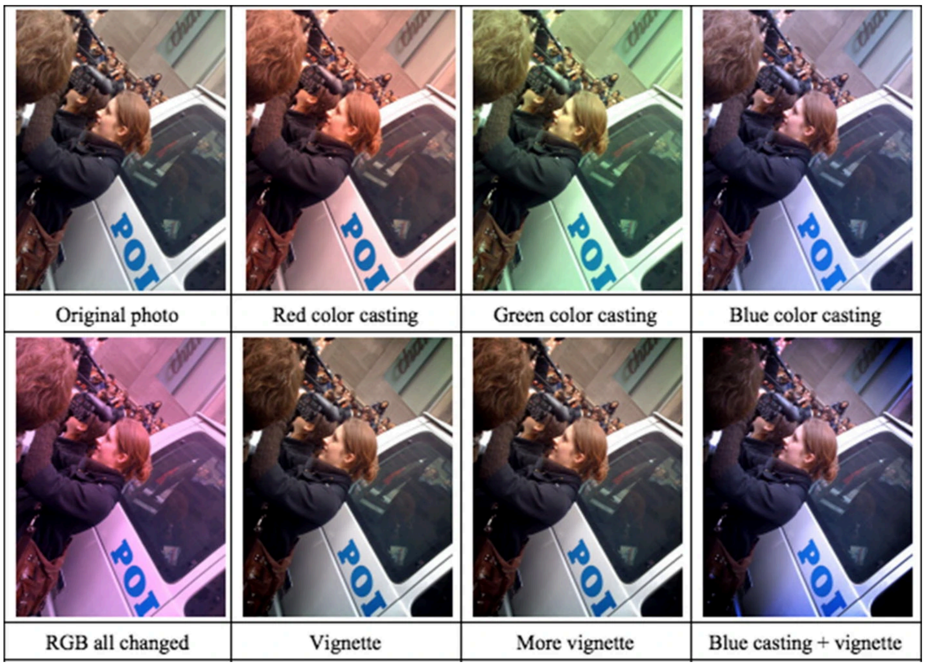


Figure 6. Example of toning augmentations [20].

Blurring and sharpening

To sharpen an image, each individual pixel is processed separately. Each pixel's colour values are changed to equal the average of the unchanged values of the surrounding pixels. The square containing the surrounding pixels is called a kernel. Each pixel in the kernel is also given a weight on how much it affects the new values. This changes the difference of the values for adjacent pixels resulting in a change of contrast. The direction in which contrast changes depends on the weights on the pixels and the level of contrast change depends on the size of the kernel as well as the weights. An example of blurring images is seen in Figure 7.

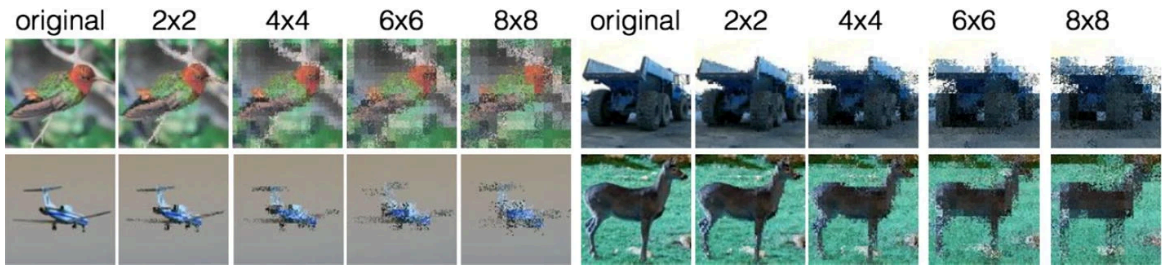


Figure 7. Example of blurring images with varying kernel sizes [18].

Erasing

To remove certain data from the image, a rectangular set of pixels in an image can be converted to be of the same colour. There is no limitation as to which colour should be chosen. This idea was developed by Zhong et al. [21] in 2017 and it helps simulate images in which objects are partially hidden. Erasing can be done randomly (see Fig. 8) and semi-randomly. Random erasing means to select the region completely randomly from within the image, whereas semi-random erasing has some awareness. For example, object-aware erasing selects the erasable region randomly from within the bounding box of the object in question [22].

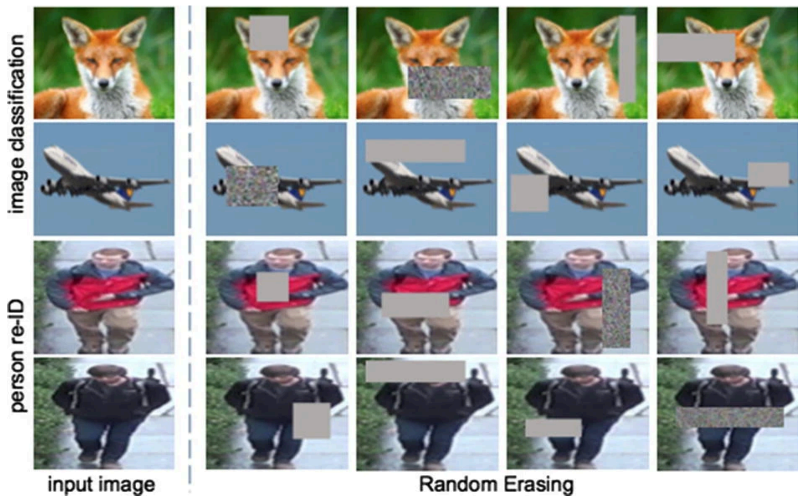


Figure 8. Example of random erasing augmentation [22].

2.1.2 Data Selection

Traditionally, a larger amount of sufficiently good data means better results in machine learning. For this reason, it was decided that it would be better to use a pre-existing dataset for the training rather than creating the dataset from the ground up. There are a few datasets available on the Internet focused on images of pets. Since this work focuses mainly on dogs and cats, all the datasets containing other types of pets were unfit for the task at hand. Of the remaining, two datasets were deemed as the best fit for having a sufficient diversity of species and number of images - the Oxford-IIIT Pet Dataset[5] and the Stanford Dogs Dataset[6]. Since the Stanford dataset contains only images of dogs, the Oxford dataset was chosen.

The Oxford-IIIT Pet Dataset contains 37 different species of cats and dogs with approximately 200 images per species. Each image is truthfully labelled and has the bounding box coordinates for the head. These coordinates were not used for this work because the model was designed for full-body identification. In conjunction with the Oxford dataset, approximately 20 images of one specific pet were added to the dataset. This was done with 2 different cats and 1 dog, each for a separate model. The low amount of images of the chosen specimen was a conscious decision based on the assumption that a random user of this product might not have hundreds or thousands of images of their pet to train the model with. Thus, the model has to be functional even with a low amount of training data. To reduce the difference of sizes of the class datasets only 1705 of the original 7400 images were kept. The selection was done randomly while keeping approximately the same amount of images for each species. These 1705 images were split with 1372 being in the training set, 264 in the validation set and 69 in the test set. An example of the images is seen in Figure 9.

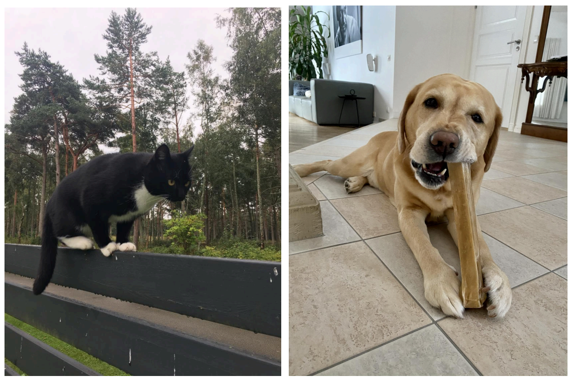


Figure 9. Example positive class images for two different pets.

[5] https://www.robots.ox.ac.uk/~vgg/data/pets/ Visited 05.01.2024
[6] https://www.kaggle.com/datasets/jessicali9530/stanford-dogs-dataset Visited 05.01.2024

For enhancing training data, every image was augmented. Each image was translated, rotated, toned, had its brightness and contrast adjusted etc. (see Fig. 10). The augmentation wasn't done separately beforehand but rather during the loading of the dataset during training. This keeps the dataset on the disk as small as possible.

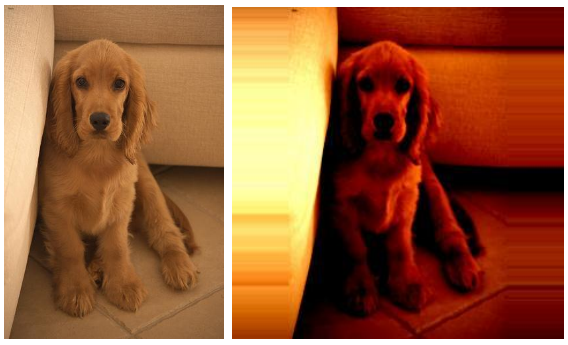


Figure 10. Training image before and after augmentations.

2.2 Implementation

The core idea for the resulting software is to be integrated into a smart pet door. The creation of the physical door is beyond the scope of this work. However, the design choices will be made with that ultimate goal in mind. The main concept is that the camera snaps an image every couple of seconds to keep itself up to date with its field of view while not computing excessively. The Raspberry Pi then performs object detection on that image. If it detects an animal in the image, it then immediately takes 9 more images and performs image recognition on all of them to make sure whether that animal is the correct specimen or not. As per the goals stated earlier, recognising the correct animal in at least 7 images out of 10 means a positive result. Less than that means a negative result. In the case of a positive result, the device sends a signal to the door to be opened. In the case of a negative result, the device returns to its original loop (see Fig. 11).

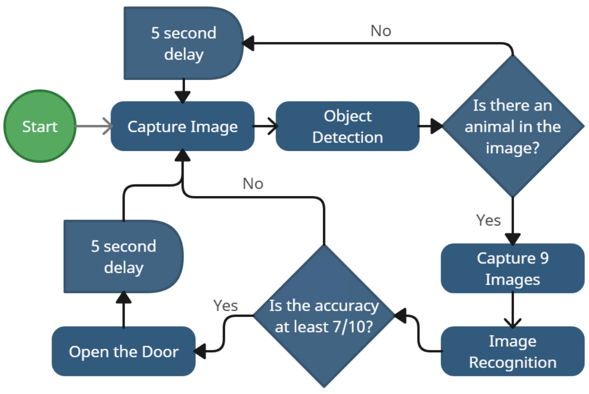


Figure 11. The core loop of the software.

2.2.1 Object Detection

The model selected to perform object detection was YOLOv5[7]. YOLOv5 is a further developed version of the YOLO models by Glenn Jocher. YOLOv5 was selected since it is easy to use and has fast inference speed. As an experimental path, all the training images were cropped with YOLOv5 to only include the pet in the image with a small buffer edge around them. This would in theory help the model by reducing unnecessary parts of the image. However, this did not yield any significant results, regardless of whether during inference the image was also cropped or not. Another option was to train the model without YOLOv5 cropped images, but instead crop the images for inference. This resulted in worse accuracy than without the cropping. As using YOLOv5 for training or inference did not succeed, it was only used for the initial check: whether there is an animal in the image. This knowledge helps analyse edge case images. For example, if the image has more than one animal in it, this check can separate these animals as separate images to be separately identified by the classifier. Another example would be images in which there are no animals. To prevent the classifier from accidentally seeing an animal in such images, YOLOv5 first checks this.

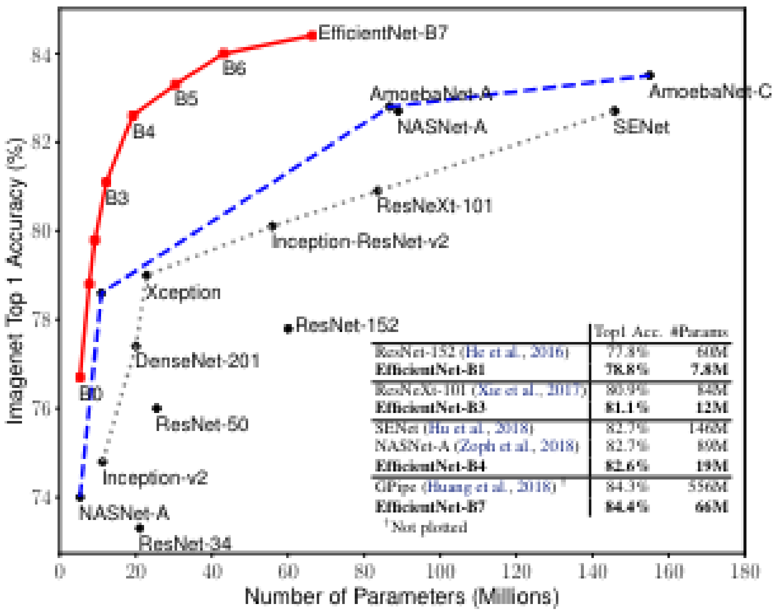
2.2.2 Image Classification

Since this software is supposed to run on a device with limited computational power, the model selected needs to balance accuracy and efficiency. For this, a pre-trained model with already efficient results was selected to reduce the time and effort required to train a well-performing model. TIMM[8] or PyTorch Image Models is a collection of models with pre-trained weights [23]. These models have varying computational requirements and results. To determine which model to select for the task at hand, a previous work[9] of the author was used which analysed a few of the Timm models. This analysis consisted of first selecting 15 timm models with words such as "efficient", "tiny" or "small" in their names as the aim was to select the most efficient model. All of the selected models were then improved upon by using transfer learning on the dataset described in the previous chapter. The script used for the transfer learning process was a modified and adapted version of the script by Mills [24]. These fine-tuned models were then tested on a previously unseen portion of the dataset for their F1 scores, FLOP counts and inference speed. All of these metrics combined resulted in a

[7] https://pytorch.org/hub/ultralytics_yolov5/ Visited 04.05.2024
[8] https://huggingface.co/docs/timm/index Visited 01.05.2024
[9] https://github.com/villemsusi/TIMM-model-analysis Visited 10.05.2024

performance ranking of which the best model was efficientnet_b0.ra_in1k[10]. This model was therefore selected as the basis of this work. The results of this analysis can be found in Appendix I.

Tan et al. [25] concluded in their research that to reach better accuracy, CNN-s can be scaled up. Their paper stated that this is often done in only one of the three dimensions: width, depth, or length. Scaling multiple dimensions independently of each other can be done but requires a lot of manual work. They however proposed a method to scale CNN-s up in every dimension with a constant ratio. This resulted in the birth of EfficientNet. As seen in figure 12 the latter versions of EfficientNet improve accuracy significantly while not increasing the number of parameters as much. However since this model aims to use as few resources as possible, the b0 version was chosen.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018)[†]	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

[†] Not plotted

Figure 12. Model size vs. ImageNet accuracy [25].

2.2.3 Training

During the development of this work, multiple approaches were taken for the model structure. At first, the model was advanced via transfer learning. For this, new layers were added to the existing model based on the inspiration received from Magid et al. [26]. Alternatively, the model was just fine-tuned with the same data without the new layers. Since the transfer learning did not show significant improvements over the fine-tuning and since the existing weights were trained on the ImageNet-1k images which is an extensive dataset and

[10] https://huggingface.co/timm/efficientnet_b0.ra_in1k Visited 01.05.2024

the model is therefore already quite accurate, it was decided to only fine-tune the existing model to reduce the time it would take for a user to train their model. Regarding the data structure, two approaches were taken.

Firstly, each class of the Oxford dataset was designated as a separate class for the model. This would force the model to classify between 38 classes, however the surrounding infrastructural software would only accept one of these classes as the correct one. The advantage of this idea is to reduce the class imbalances by evening out the number of images for each class in the dataset. This turned out to not be an effective solution since the resulting model predicted with low accuracy.

The second approach was to train the model only on the images of the correct specimen for a one-class classifier. This approach would eliminate the need for extra images and would therefore make training considerably faster. A possible disadvantage of this approach is the lack of data used for training and therefore the possibility that the resulting model wouldn't be effective at generalising. However, this approach was considered outside of the scope of this work due to the technical complexities involved.

The final structure found for the model was to train it to predict two classes: "Correct" and "Incorrect". The "Correct" class contains all of the images of the chosen specimen and the "Incorrect" class contains all of the augmented versions of the chosen images from the Oxford dataset. All of the images were resized to 288 by 288 pixels to further reduce the computational resources necessary for training the model. The maximum learning rate was set to 10^{-3} and the selected optimiser was AdamW. The model was trained for 3 epochs since experimenting showed this to have the best balance of time spent training and model accuracy.

The trained models were evaluated for the F1-score on the testing data. The model was at first trained on a Geforce RTX 2070 Super[11] GPU to reduce the training time spent while experimenting. In the production phase, the model is planned to be trained on the CPU of the Raspberry Pi which is considerably slower but keeps the core idea of an edge device.

[11] https://www.nvidia.com/en-us/geforce/news/gfecnt/geforce-rtx-20-series-super-gpus-out-now/ Visited 04.05.2024

2.2.4 Hardware Setup

This software was run on a Raspberry Pi 4 Model B[12]. Connected to it was a Raspberry Pi 4B/3B+ Night Vision Camera - FishEye[13] camera module, which has IR capabilities for nighttime use. The camera module was connected to the Pi's integrated camera module port via a camera module ribbon cable (see Fig. 13). The interaction with the Pi was handled over Ethernet via a custom browser-based user interface which is further discussed in the next chapter. The only necessary cable connection other than Ethernet was power - no additional peripherals were required after the initial networking setup. Since networking is not the focus of this work, the connection setup to the home network over Ethernet will not be discussed.

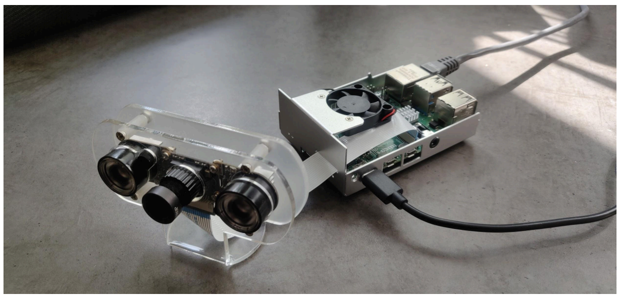


Figure 13. Raspberry Pi setup with camera module.

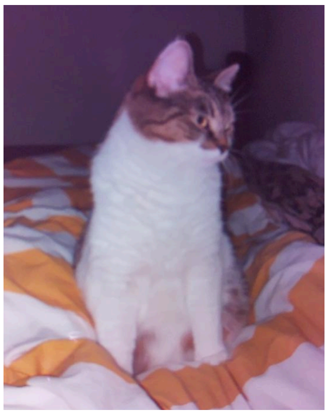


Figure 14. Image taken with the camera module.

2.2.5 User Interface

The user interface (UI) was built with Python 3.9[14] using the Streamlit[15] library. Streamlit is a purely Python-based library designed for creating web applications. The core concept is to be able to quickly prototype simple web apps without the hassle of front-end design while

[12] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ Visited 07.05.2024
[13] https://digipurk.ee/en/product/raspberry-pi-4b-3b-night-vision-camera/ Visited 07.05.2024
[14] https://www.python.org/downloads/release/python-390/ Visited 07.05.2024
[15] https://streamlit.io/ Visited 07.05.2024

focusing mainly on functionality and data visualisation. Since the aim of the UI in this work is not to be beautiful, but rather functional, this library fits well.
The UI was designed with a focus on simplicity and functionality. The core function of the web interface was the ability to upload images and train a new model with those images. Secondary priorities were the possibility to constantly stream the camera feed and to open the door at will. Finally, it was deemed important to have the functionality to access the capture logs to check captured images with their corresponding timestamps.

3 Results

This chapter showcases the resulting software from the perspective of the interface and the identification model. Additionally, it opens possible avenues for further development. The source code for this project is available at https://github.com/villemsusi/PetIdentification.

3.1 Resulting Model

As a final result, three models were trained, each for a different specimen. Two models were trained for a cat and one for a dog. Each model was trained with images of those animals. These models performed with varying outcomes. All of the models predicted correctly on the negative class every time. The least accurate model (Dog 1) predicted correctly on the positive class 14% of the time and had an F1-Score of 0.25. The cat models Cat 1 and Cat 2 predicted correctly on the positive class 50% and 73% of the time and had F1 scores of 0.67 and 0.84 respectively (see fig. 15).

Model	**True Positive Rate**	**F1-Score**
Dog 1	0.14	0.25
Cat 1	0.5	0.67
Cat 2	0.73	0.84

Figure 15. True Positive Rates and F1-Scores of the resulting models.

These results correspond with the amount and variety of positive class training images given for each model. As the positive class, the Dog 1 dataset had 7 training images and 4 validation images with relatively similar backgrounds and poses. The Cat 1 dataset contained 8 training images and 4 validation images with more variation in the scenery and poses. The Cat 2 dataset contained 9 training images and 5 validation images while having the most variation. The amount of variation was estimated subjectively based on the pose of the animal, the amount of the animal's body in the image and the environment around the animal. This discrepancy in results points to the instability of the model and the level of reliance it has on the given training images vis-a-vis performance. Acknowledging this, the resulting model is far from being reliable enough to use offhandedly but can yield promising results when calibrated properly.

3.2 User Interface

To interact with the software, this solution uses a graphical web interface. This is hosted and accessible on the Raspberry Pi but is also accessible by another computer on the same network. The web application is split into three tabs: Train, Monitor and Log. These tabs were made with functionality and simplicity in mind.

The train tab is used to initially train the model with images supplied by the user through the same interface (see Fig. 16).

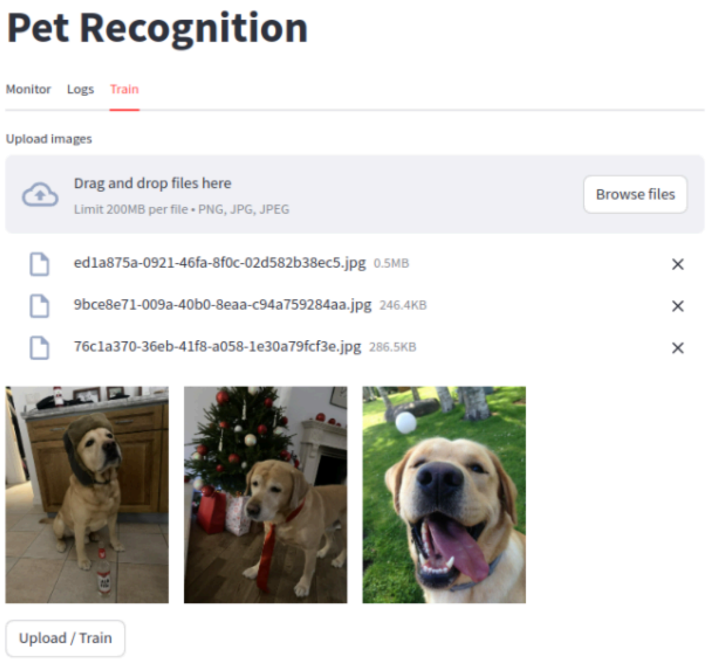

Pet Recognition
Monitor
Logs
Train
Upload images
Drag and drop files here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
ed1a875a-0921-46fa-8f0c-02d582b38ec5.jpg 0.5MB
9bce8e71-009a-40b0-8eaa-c94a759284aa.jpg 246.4KB
76c1a370-36eb-41f8-a058-1e30a79fcf3e.jpg 286.5KB
Upload / Train

Figure 16. Train tab of the web interface.

The log tab lets the user select how many of the latest snaps the user wishes to see and showcases these images with their corresponding capture times (see Fig. 17).

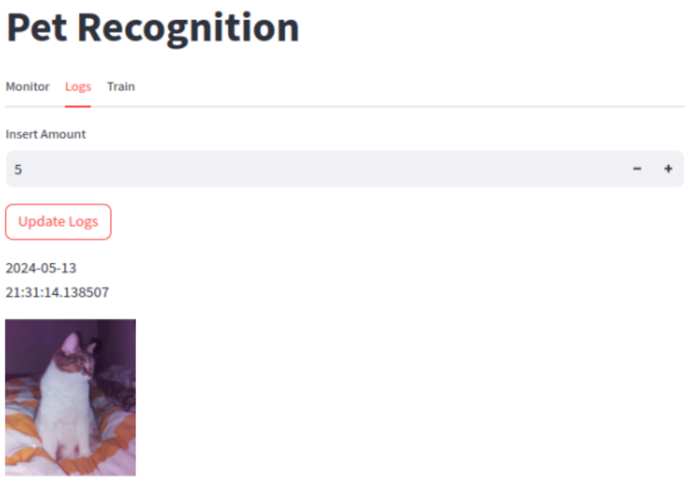


Figure 17. Log tab of the web interface.

The monitor tab functions as a tool to monitor the live feed streaming from the camera (see Fig. 18). The live feed streams at approx. 5-10 frames per second with a resolution of 720 x 720. The camera snaps images every 5-10 seconds based on the current computational power available. It’s also possible to snap images manually from the live feed view.

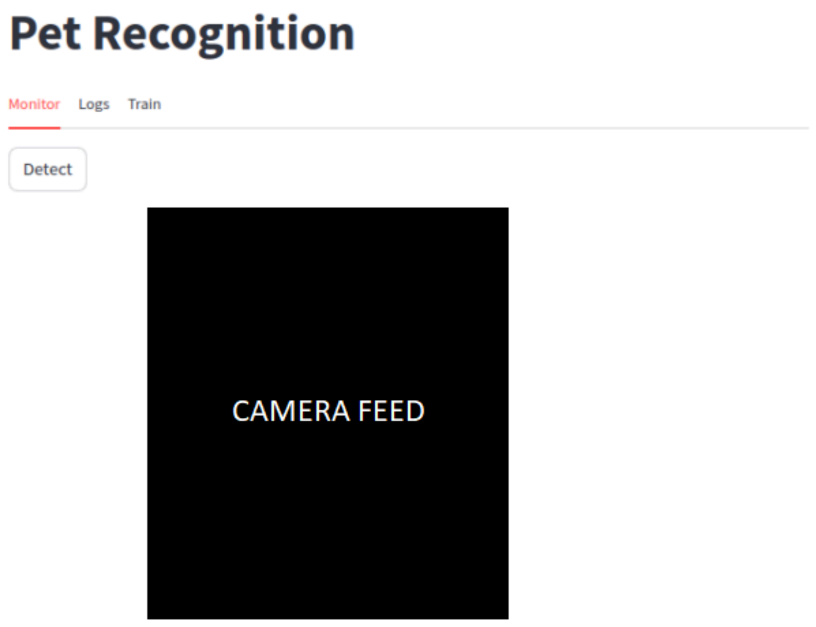


Figure 18. Monitor tab of the web interface.

3.3 Future Work

The following things could be improved upon regarding this work.

Firstly, the camera currently selected for the hardware setup has a fisheye lens which is very wide-angled. Therefore the images it captures have strong distortion (mostly at the edges) and this could cause images to be unrecognisable by the model since the training images won’t necessarily be distorted to such a degree. To combat this, image correction could be applied to each of the inference images, distorting the images back towards the normal look. Another option would be to swap out the camera for a wide-angle lens which would decrease the area from which the pet can be seen but would result in less distortion.

Secondly, the camera currently used has two infrared emitters which allow it to be used during nighttime. For this reason, the camera does not have an infrared lens attached to it which causes the images it captures to have strong blue and red hues. This however again affects the model's capability to correctly identify. To combat this, currently, the infrared emitters are covered to reduce the amount of extra infrared light in the images but remove the nighttime capabilities. This is done because the camera does not have the option to select between using the emitters or not. As a better solution, the infrared emitters could be modified to be a separate entity controlled by the Raspberry Pi and therefore be togglable to work only at night.

Thirdly, the device currently captures an image for analysis every couple of seconds which keeps the device constantly computing. This could over a period of time wear out the device. A solution for this could be to add a motion or proximity sensor to the Raspberry Pi GPIO pins to trigger the initial detection.

Most importantly, as previously mentioned, the current model is objectively unreliable. This could be further improved upon by identifying the specific details which make a set of training images more useful for the training process. This would then help guide the user with written instructions to select better images. Another option would be to further experiment with different layers for transfer learning.

Finally, to complement the existing software, an integrated door hardware solution could be added to the system to have a functional and purposeful end product.

Conclusion

In this thesis, a new pet identification system running on a Raspberry Pi was designed and developed to operate a hypothetical smart pet door. The system consists of three components. Firstly, the hardware consists of a controller (Raspberry Pi) and a camera. The controller could also hypothetically be connected to a pet door. Secondly, the computer vision models are a pre-trained YOLOv5 to perform object detection of pets and a fine-tuned EfficientNet to perform pet identification. The EfficientNet model is fine-tuned using a dataset consisting of internet-sourced images as the negative class and user-provided images as the positive class. Finally, a web interface for the user to interact with the hardware, fine-tune the model on new data and monitor the camera feed in real-time. The goal of this system was to reach an F1-score of 0.8 on the identification task.

To perform pet detection in images, the author set up the YOLOv5 model. To perform pet identification the author fine-tuned the EfficientNet model on three datasets, which resulted in F1 scores of 0.84, 0.67 and 0.25. This was done using a minimal amount of training images. The results gained show that with sufficiently good data, the identification works well, but with bad data, the model performs exceptionally badly. To add reliability, the core loop of the system checks multiple images to verify the identity of the targeted animal.

The author is confident that this model structure can be good enough, as shown by the results. The model should be further improved by identifying which characteristics constitute good data, reducing the distortion caused by the fisheye lens, adding nighttime availability by controlling the infrared emitters, adding a proximity sensor to the Raspberry Pi and reducing the limitations caused by a small dataset.

References

[1] Hosna, A., Merry, E., Gyalmo, J. et al. (2022). Transfer learning: a friendly introduction. Journal of Big Data 9(1), pp 1-19. https://doi.org/10.1186/s40537-022-00652-w

[2] Azizi, E., Zaman, L. (2023). Deep Learning Pet Identification Using Face And Body. Information 2023, 14, 278. https://doi.org/10.3390/info14050278

[3] Elngar, A. A., Arafa, M., Fathy, A., Moustafa, B., Mahmoud, O., Shaban, M., Fawzy, N. (2021). Image Classification Based On CNN: A Survey. Journal of Cybersecurity and Information Management, pp. 18-50, 2021, doi: 10.54216/jcim.060102. (16.04.2024)

[4] Agarap, A. F. (2019). Deep Learning using Rectified Linear Units (ReLU). arXiv, Article 1803.08375. https://doi.org/10.48550/arXiv.1803.08375

[5] University of Washington (2022). Convolutional Neural Networks. CSE416 Textbook, chapter 18. https://courses.cs.washington.edu/courses/cse416/22su/lectures/10/lecture_10.pdf (12.03.2024)

[6] Rashid, M., Gu, X., Lee, Y. J. (2017). Interspecies Knowledge Transfer for Facial Keypoint Detection. arXiv, Article 1704.04023. https://doi.org/10.48550/arXiv.1704.04023 (16.04.2024)

[7] Wu, Y., Ji, Q. (2015). Discriminative Deep Face Shape Model for Facial Point Detection. Int J Comput Vis 113, 37–53. https://doi.org/10.1007/s11263-014-0775-8

[8]Schroff, F., Kalenichenko, D., Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. arXiv, Article 1503.03832. https://doi.org/10.1109/CVPR.2015.7298682

[9] Xiao, Y., Tian, Z., Yu, J., Zhang, Y., Lui, S., Du, S., Lan, X. (2020). A review of object detection based on deep learning. Multimedia Tools And Applications, 79, pp 23729-23791, 2020. https://link.springer.com/content/pdf/10.1007/s11042-020-08976-6.pdf

[10] Parab, C. U., Mwitta, C., Hayes, M., Schmidt, J. M., Riley, D., Fue, K., Bhandarkar, S., Rains, G. C. (2022). Comparison of Single-Shot and Two-Shot Deep Neural Network Models for Whitefly Detection in IoT Web Application. AgriEngineering 4, no. 2, pp 507-522. https://doi.org/10.3390/agriengineering4020034

[11] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. arXiv, Article 1506.02640. https://doi.org/10.48550/arXiv.1506.02640

[12] Zhao, C.W., Jegatheesan, J. and Loon, S.C. (2015). Exploring iot application using raspberry pi. International Journal of Computer Networks and Applications, 2(1), pp.27-34.

[13] Fanariotis, A., Orphanoudakis, T., Kotrotsios, K., Fotopoulos, V., Keramidas, G., Karkazis, P. (2023). Power Efficient Machine Learning Models Deployment on Edge IoT Devices. Sensors 2023, 23, 1595. https://doi.org/10.3390/s23031595

[14] Johnston, S. J., Cox, S. J. (2017). The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams. Electronics 2017, 6, 51. https://doi.org/10.3390/electronics6030051

[15] Lan, Y. (2022). Pet Finding: Computer Vision Approaches for Pet Face Recognition and Verification. Master's thesis. University of California. Applied Statistics. https://escholarship.org/content/qt6nf7b2vq/qt6nf7b2vq_noSplash_4fb9b2292486a1fa5f2e89deaf02dc11.pdf?t=r8xrld (28.11.2023)

[16] Klein, A. (2019). Pet Cat Face Verification and Identification. Stanford University. https://cs230.stanford.edu/projects_fall_2019/reports/26251543.pdf (28.11.2023)

[17] Bayer, M., Kaufhold, M.-A., Reuter, C. (2022). A Survey on Data Augmentation for Text Classification. ACM Computing Surveys. Volume 55, Issue 7, Article No.: 146, pp 1–39. https://dl.acm.org/doi/pdf/10.1145/3544558 (28.11.2023)

[18] Shorten, C., Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. Journal of Big Data 6(1), pp 1-48. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0 (29.11.2023)

[19] FigHo, D., Liang, E. (2019). Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules. arXiv, Article 1905.05393. https://doi.org/10.48550/arXiv.1905.05393

[20] Ren, W., Shengen, Y., Yi, S., Qingqing, D., Gang, S. (2015). Deep image: scaling up image recognition. CoRR, abs/1501.02876.

[21] Zhong, Z., Zheng, L., Kang, G., Li, S., Yi, Y. (2017). Random Erasing Data Augmentation. arXiv, Article 1708.04896. https://doi.org/10.48550/arXiv.1708.04896 (29.11.2023)

[22] Zhun, Z., Liang, Z., Guoliang, K., Shaozi, L., Yi, Y. (2017). Random erasing data augmentation. arXiv, Article 1708.04896. https://doi.org/10.48550/arXiv.1708.04896

[23] Wightman, R., Touvron, H., Jégou, H. (2021). ResNet strikes back: An improved training procedure in timm. arXiv, Article 2110.00476. https://doi.org/10.48550/arXiv.2110.00476

[24] Mills, C. (2023). Fine-Tuning Image Classifiers with PyTorch and the timm library for Beginners. https://christianjmills.com/posts/pytorch-train-image-classifier-timm-hf-tutorial/ (24.01.2024)

[25] Tan, M., Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Proceedings of the 36th International Conference on Machine Learning, PMLR 97:6105-6114.

[26] Magid, S. A., Petrini, F., Dezfouli, B. (2019). Image Classification on IoT Edge Devices: Profiling and Modeling. arXiv, Article 1902.11119. https://doi.org/10.48550/arXiv.1902.11119

Appendix

I. TIMM model analysis results

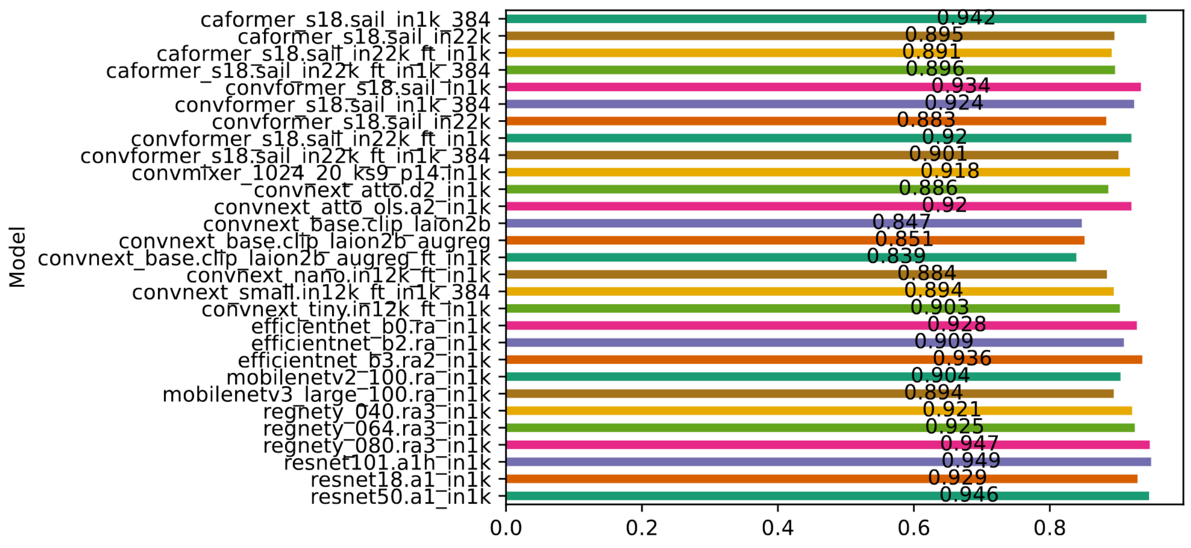


Figure 19. F1-scores of selected timm models.

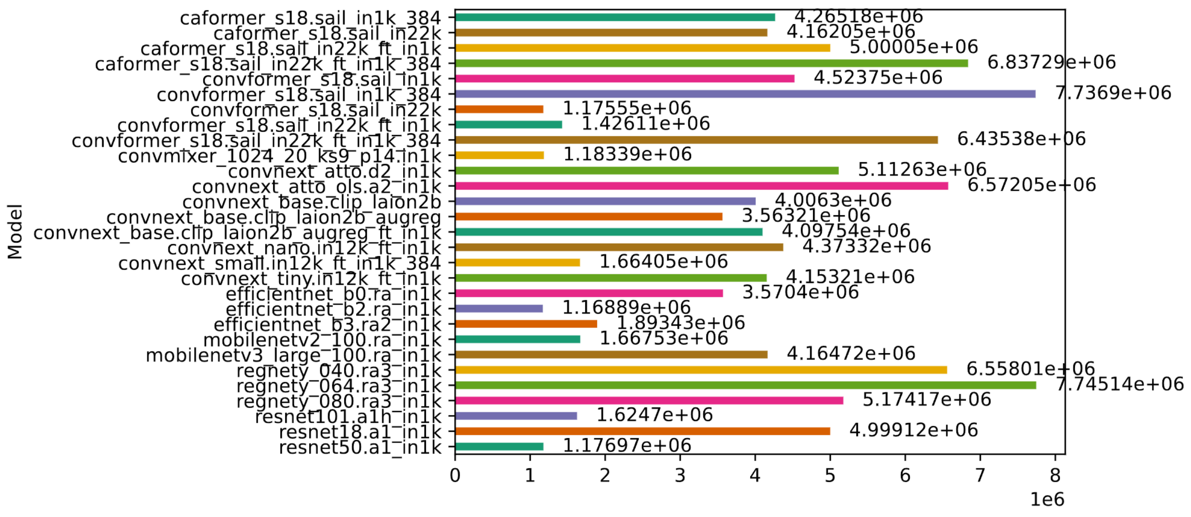


Figure 20. FLOPs of selected timm models.

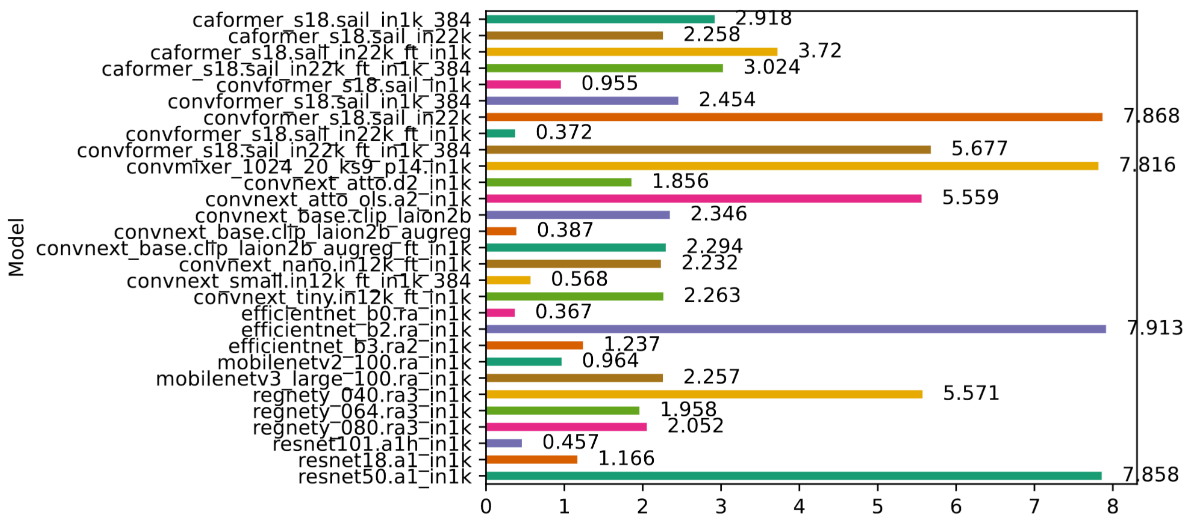


Figure 21. Time consumption for single image inference for Timm models.

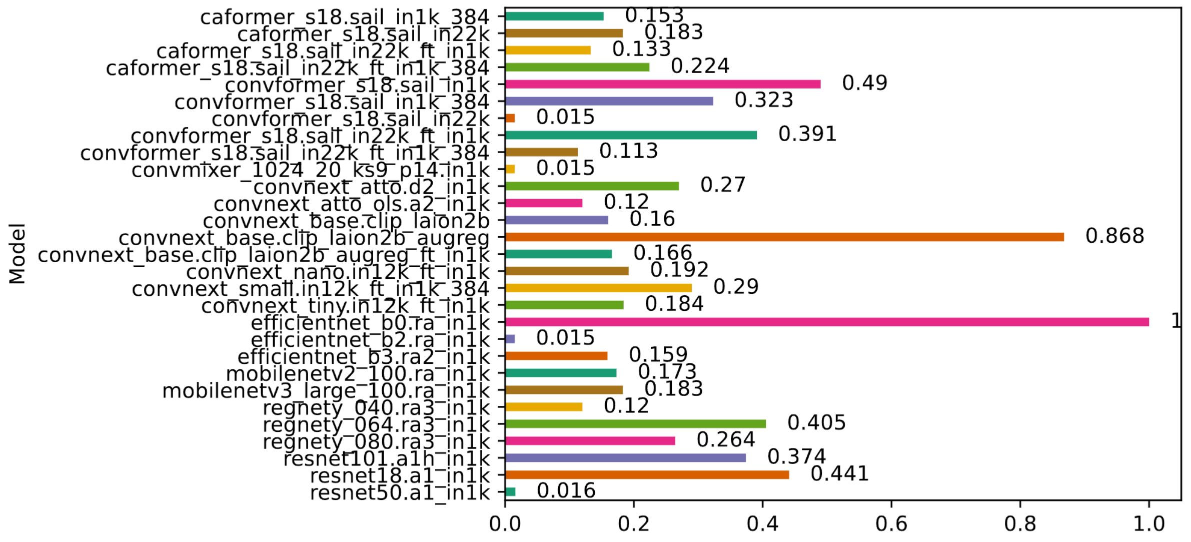


Figure 22. Normalised (F1 * FLOPs/time) for timm models.

II. Licence

Non-exclusive licence to reproduce the thesis and make the thesis public