Animal Observation Using Motion and Object Detection

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Valkeakoski, Electrical and Automation Engineering

Fall 2020

Hung Le

Khang Truong

Bachelor of Electrical and Automation Engineering
Valkeakoski

Author Hung Le, Khang Truong **Year** 2020

Subject Animal Observation Using Motion and Object Detection

Supervisor(s) Raine Lehto

ABSTRACT

This thesis project was commissioned by Tentrio Oy. In the project, hours-long videos that were taken by a fixed camera, often overlooking a room, were processed. The goal was to make a program that would take these video files and based on certain criteria, would output edited videos that were essentially relevant sections of the input videos. More specifically, the primary goal of this project was to make a program that could detect the presence of motion in video frames. Additionally, as a secondary objective, the program was to also look for the presence of cats or dogs. Any segment of the video featuring these elements was to be combined into a single output video. The timestamps of wherein the relevant video segment took place in the video were also to be recorded.

To achieve this, Python 3.6 was used as the main programming language along with several libraries. Motion detection was implemented using OpenCV and FFmpeg. On the other hand, the part of the program that handled animal detection was done using TensorFlow, an open source machine learning library. The program needed to be set up as an online service running from a server that would process any video file uploaded by an authenticated user. The server would also allow the user to download the output videos and timestamps. The hardware in which the server ran on was a remote machine provided by the commissioning party. The server was written in Python with Django as the web framework.

In the end, a motion detection program running on a remote server was created. On the other hand, object detection was not implemented due to time constraints. Properly integrating object detection into the program would likely take too much time. Moreover, the commissioning party was satisfied with the results which was why object detection was left out in the final product.

Keywords Animal observation, machine learning, object detection, video editing

Pages 35 pages including appendices 1 page

CONTENTS

1 INTRODUCTION

With the rise of CCTV and video surveillance, more and more people decide to equip their houses with a home security camera system. However, due to the fact that the system is mostly required when there is nobody home, the majority of the footage is of empty rooms. Therefore, a motion detector could be used in order to sort out the important parts of the video. This detector would be managed by OpenCV, an open source library mainly aimed at computer vision and would be capable of operating in a Linux environment, where the bulk of the project would be written and operated on according to the wish of the third-party company. Python is a robust and flexible programming language, with many options supporting image and video processing and manipulating. Linux is an open source operating system, allowing many customized features tailored to each user's needs.

Furthermore, if the house owner has pets, more attention is demanded to keep track of them. Thus, an object detector might be applied to solve this issue. The object detection would be handled by TensorFlow through the use of machine learning. TensorFlow is a free and open source software library with many applications such as language detection, voice recognition and, relating to this project, image processing.

Afterwards, since just knowing when things happen still requires one to search for them in the videos, some kind of video processing program was needed to trim out the unnecessary parts. Hence, FFmpeg was used to manipulate the files in this project to get the desired results. Ffmpeg is a free and open source software project that includes tools which can be used for transcoding, streaming and playing multimedia files.

Finally, the goal of the project was to have a website where unprocessed videos could be uploaded, and be able to download the finished videos after they had been processed by the program. For this task, the company provided access to their server, while Django was used as the web framework. Django is a free and open source web framework based on Python.

This project was conducted by two people, Hung Le and Khang Truong. Mr Le was in charge of object detection and web server section. Mr Truong was responsible for the motion detection and video trimming segment. Both managed the data annotation for object detection.

2 MOTION DETECTION

The first part of this project deals with motion detection. In particular, an algorithm was used to pick out any movements during the length of the video. Most importantly, the main point of this was to trim down the video to the relevant parts. OpenCV, a free and open source computer vision and machine learning software library, on Python 3.6 was used to read and process the video in order to find the movements. (OpenCV team, 2020)

2.1 Introduction

Motion detection is the use of one or multiple technologies and software algorithms to detect moving objects within a specific area. There are various ways to detect motion, like using sensors to measure changes in light, infrared, sound, ultrasound, vibration, magnetism or even radio and microwave. In the case of motion detection in videos, frames differencing or background subtraction is used there instead.

Motion detection has many applications in everyday life, such as automatic doors, lightings, sinks, flushers, dryers, to something like intruder alarms. In the field of computer vision, such things as traffic controlling, intelligent video surveillance and even human behavior identification are possible. (Motion Detection, 2020)

2.2 Operation

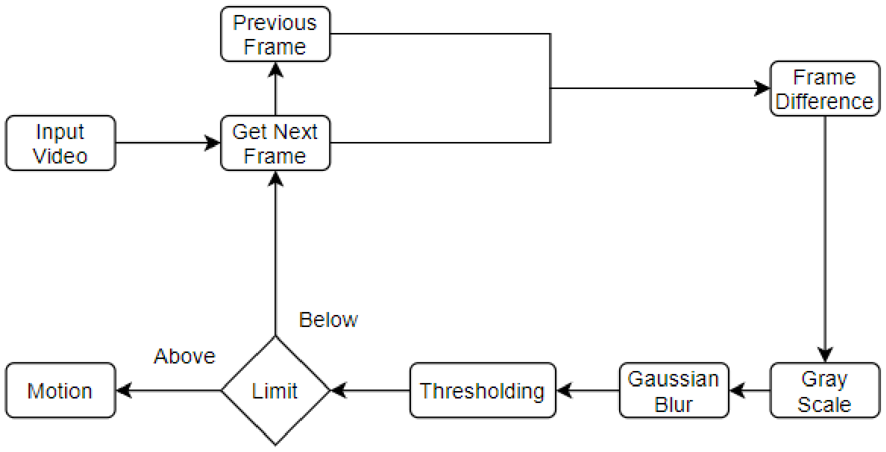


Figure 1. Flowchart of how motion detection works

A basic frame by frame comparison was used in order to detect motions. First the video was read one frame at a time. Then, the absolute difference between two adjacent frames, if there was any, was found. Next, the frame difference image got converted into grayscale, then Gaussian Blur for easier processing. Afterwards, the image was transformed into purely black and white, any pixel that had a value higher than a predetermined threshold was assigned white while the rest as black, in order for the contrast to stand out more.

The final comparison picture was essentially the way to determine whether there were motions or not. If the number of white pixels was higher than a limit, it means the second frame was different enough from the first frame to say that there was motion happening.

2.2.1 Frame skipping

The videos used for this task often had high file sizes, usually many hours long, and processing them tended to take several times the length of the videos. In order to cut down the procedure time, a frame skipping technique was employed. Basically, the program would only check and work on the video frames after an interval of predetermined number of frames.

2.2.2 Delay timer

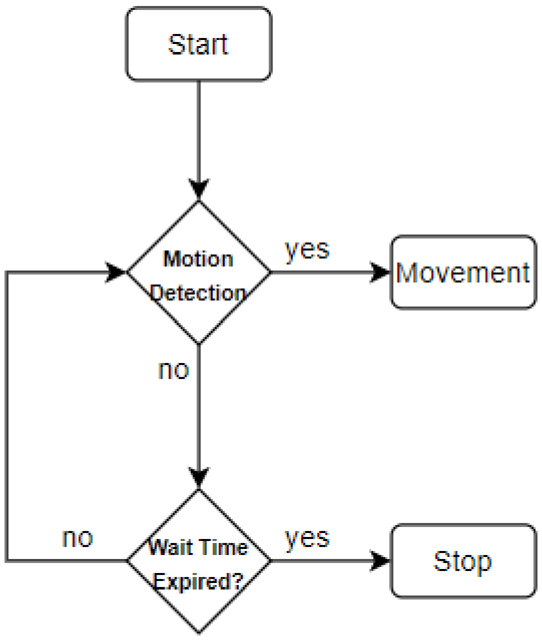
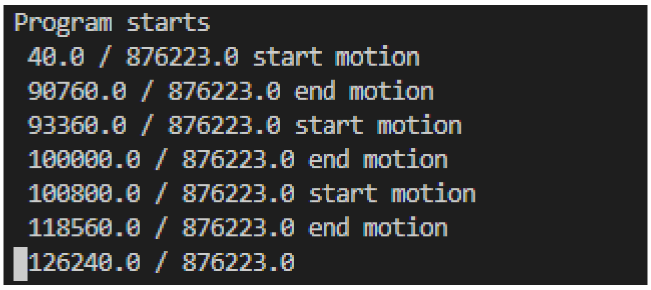


Figure 2. Time delay relay

Motion detection worked using a threshold, meaning if the number of pixels in the difference frame was higher than the threshold then there was motion, and no motion if lower. Usually, the pixel numbers would jump above and below the threshold a lot and make the recordings and videos look choppy and disjointed. Therefore, in order to smooth them out, A kind of time delay was used.

As demonstrated in Figure 2, while checking for motion, whenever the detector could not find any movement, instead of immediately concluding that it did not find anything, the program would keep observing for a bit longer. If the detector discovered any motion during this time, it would reset the timer and start looking again.

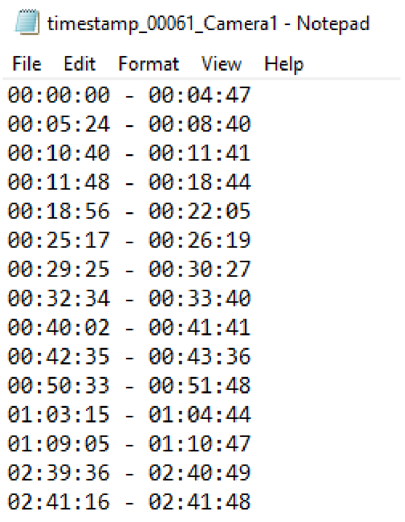
However, if the detector still had not noticed any movement after the timer had finished, then the program could now declare that the current motion had ended and stop noting down this section. Then the program would wait until the detector picked up the next sign of movement again.



```
Program starts
 40.0 / 876223.0 start motion
 90760.0 / 876223.0 end motion
 93360.0 / 876223.0 start motion
 100000.0 / 876223.0 end motion
 100800.0 / 876223.0 start motion
 118560.0 / 876223.0 end motion
 126240.0 / 876223.0
```

Figure 3. Running status of motion detection

The program ran detection and printed out the frame position of the start and end of motion on the terminal for easy tracking. The timestamps were then extracted into a log file.



```
00:00:00 - 00:04:47
00:05:24 - 00:08:40
00:10:40 - 00:11:41
00:11:48 - 00:18:44
00:18:56 - 00:22:05
00:25:17 - 00:26:19
00:29:25 - 00:30:27
00:32:34 - 00:33:40
00:40:02 - 00:41:41
00:42:35 - 00:43:36
00:50:33 - 00:51:48
01:03:15 - 01:04:44
01:09:05 - 01:10:47
02:39:36 - 02:40:49
02:41:16 - 02:41:48
```

Figure 4. A timestamp log

2.2.3 Error

For unknown reasons, the built-in timestamp tracker of OpenCV did not work correctly. The videos given by the third-party company had two types, 30 fps and 50 fps. The tracker worked accurately for the videos with 30 fps, but did not do the same with the 50 fps ones. The timestamps were two times slower than the true time of the videos in the latter situation.

A temporary solution was devised. If the video has 30 fps, then the timestamp log will record it as normal. If the video has 50 fps, the timestamp will be multiplied by two and then be recorded to the log.

2.3 **Frame differencing**

Motion was detected by using frame differencing, alongside with some filters. Frame differencing is a technique where the computer compares the difference between two video frames. If there are differences, something has changed. Most of the time, these are just “noise” from the environment like shadows, lighting, or a part of the camera devices themselves like auto focus, and brightness correction. Therefore, blur and threshold should be used in conjunction in order to filter out the unnecessary parts.

2.3.1 Frame difference

Comparing frames is a common method of detecting motion. Two frames are put under scrutiny in order to check for the absolute difference between them. Changes mean something happens, filters and thresholds are used to sort out the noise.

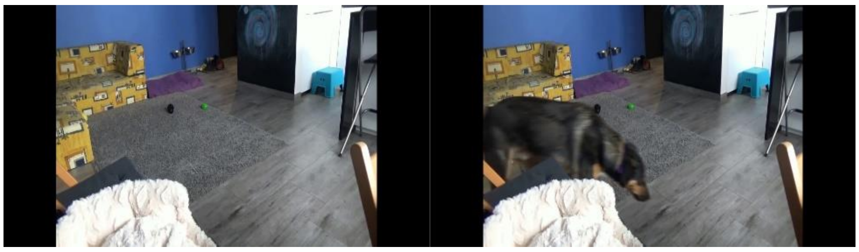


Figure 5. Empty room (left) and a room with an animal (right)

As seen in Figure 5, since the camera is fixed, the background is the same in both pictures; therefore, any difference while comparing the frames means that something is moving.

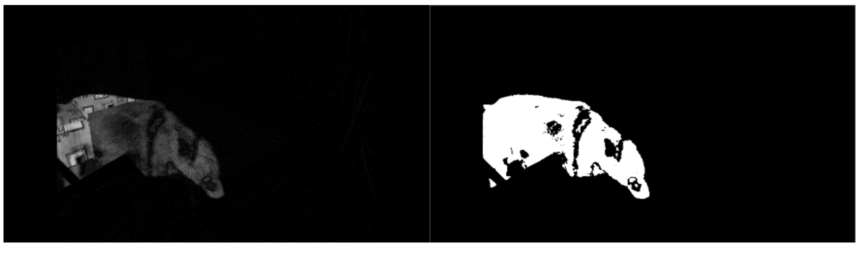


Figure 6. Frame difference (left) and Frame after filters(right)

2.3.2 Filters

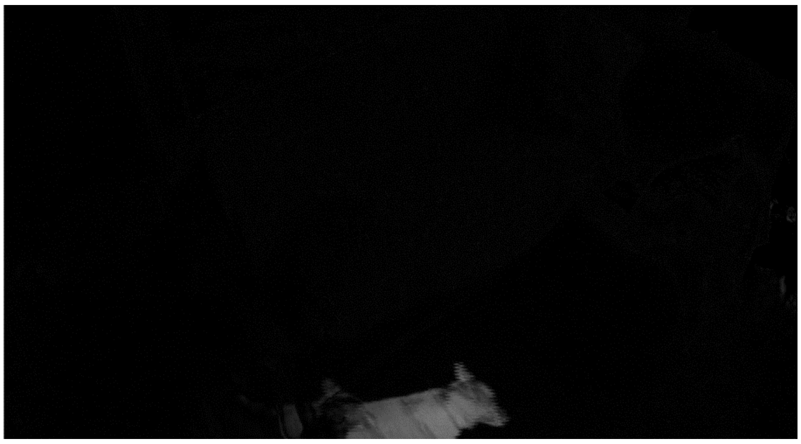


Figure 7. Grayscale of difference image

Normally, comparison images taken by using the frame differencing method often have only the sections with movements visible, since the camera is fixed and thus the background is the same between the frames.

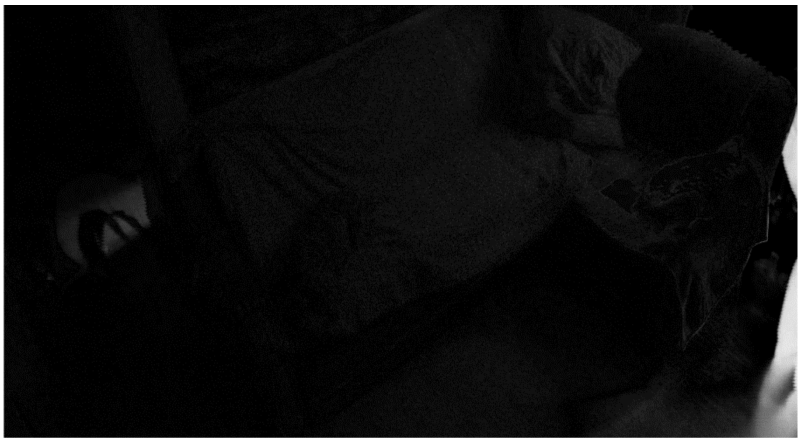


Figure 8. Grayscale of difference image with noise

However, occasionally the camera can pick up minute differences created by something like lighting or focus shift, and thus create a lot of extra unnecessary details.

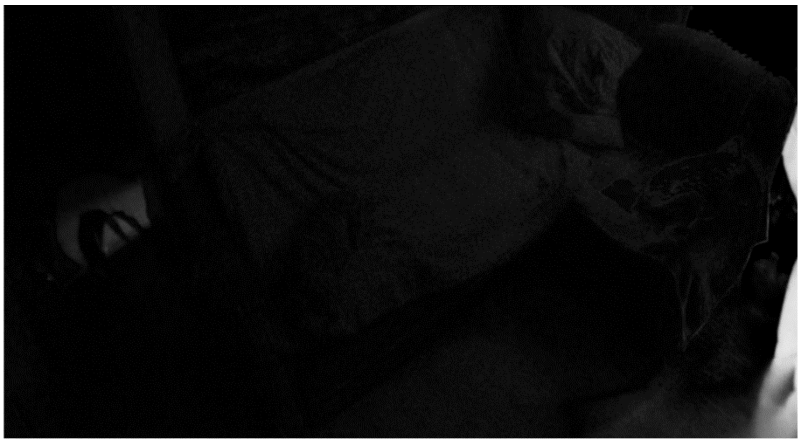


Figure 9. Image after using Gaussian smoothing

This can be dealt with by using Gaussian smoothing. It is an image processing technique via the Gaussian function, it can smooth out the image and reduce noise and detail.

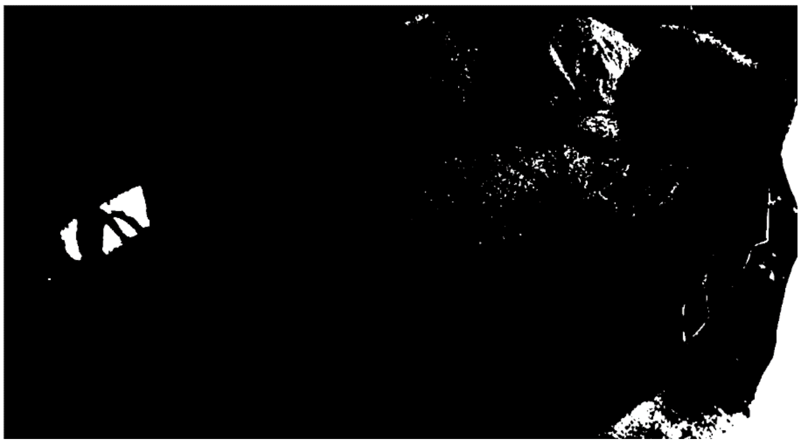


Figure 10. Image after filtering through thresholding

After that, further cleaning can be done through thresholding. It is another image processing technique where each pixel currently on the grayscale is replaced with completely black and white pixels according to a fixed constant.

3 OBJECT DETECTION

The other half of this project relates to machine learning. More specifically, machine learning was used to teach the computer to recognize and identify instances of cats and dogs whenever and wherever they are in the video. This part of the program would work in conjunction with the motion detection aspect of the program as another metric to consider during video editing. TensorFlow 1.14 on Python 3.6 was used as the training framework for the machine learning model which then will be imported by

OpenCV's DNN library where the model would perform object detection on videos.

3.1 **Theory**

3.1.1 Introduction

Machine learning is a field of study in which a computer is trained using existing data to make predictions or calculations. The aim of the training process is essentially creating and modifying mathematical algorithms until the computer can do these tasks satisfactorily. This collective of algorithms is called a "model". With this model, the computer can then perform these tasks on its own without any additional input from human operators.

Machine learning has a wide variety of applications such as recognizing e.g speech or text patterns, machine translation, predicting economic trends, object classification and detection.

3.1.2 Deep learning

For this project, deep learning was utilized to teach the machine learning model how to recognize the presence of dogs and cats in videos. Deep learning is a subset of machine learning, which uses (amongst others) "deep neural networks" (DNN), a type of model architecture.

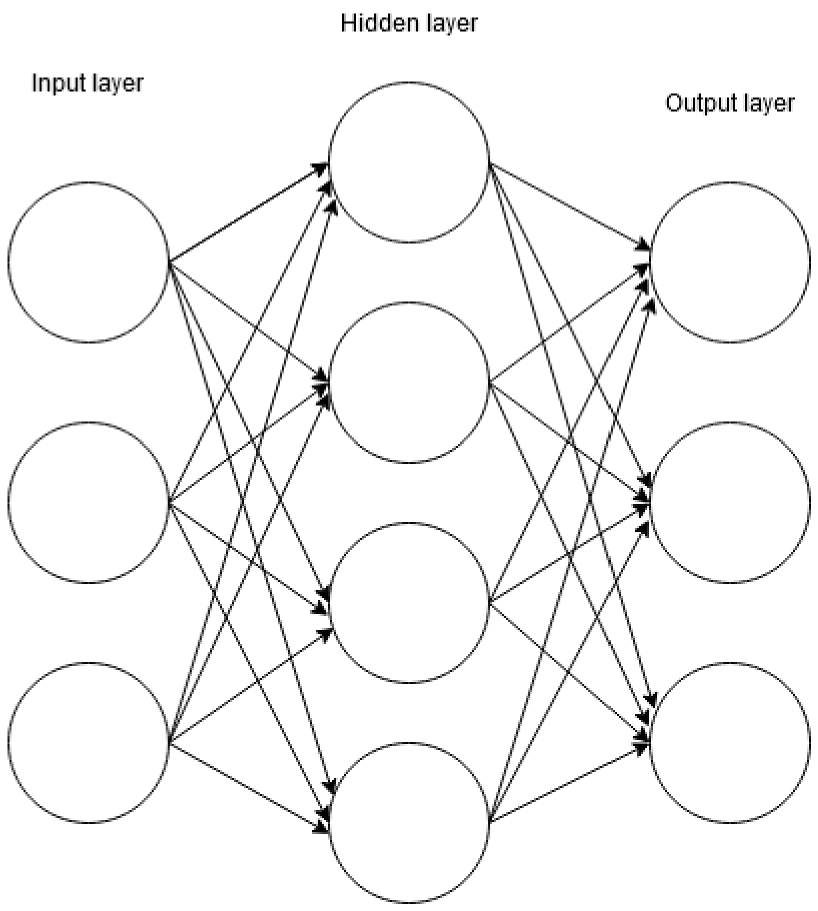


Figure 11. Typical topology of a deep neural network (Artificial neural network, 2020)

Generally, DNNs consist of many layers. They are, in order of appearance, the “input layer”, the “hidden layer” and the “output layer”. Each of these contains many “nodes” or “neurons”. These individual nodes are simple numbers or variables. All the nodes of a layer are connected to all the nodes of the previous and next layer, forming a large interconnected network. The structure of these models somewhat mimics the structure of the human brain hence the term “neural network”.

DNNs begin with the input layer where the models would receive inputs. The inputs are then passed through to the hidden layer where they are transformed as they go. Though the hidden layer may consist of a single layer (as implied by the above diagram), the hidden layer actually typically consists of hundreds, if not thousands of layers of neurons. Truly advanced DNNs may contain thousands of layers with millions of nodes. Each of the nodes contain variables or their own “weights” and “biases”. Using the resulting outputs in the “output layer”, the model can then make a prediction.

From a purely mathematical standpoint, the model is a single massive equation where every node is a variable with its own weight and bias.

Fundamentally, by iteratively adjusting these individual values, the model can learn to make accurate predictions.

3.1.3 Faster-RCNN

In this project, a model that could perform object detection on videos was needed, meaning it would scan each frame for objects of interest and determine their presence in the frame with decent accuracy. For the project, pre-trained models were chosen because they were already quite good at detecting whatever object they were trained on and objects in general. Moreover, making a model from scratch is not only difficult but quite time-consuming. There are a number of publicly available machine learning models commonly used for object detection that are pre-trained using public datasets (such as COCO, and Open Images).

The chosen model was of type "Faster RCNN", a type of DNN. RCNN stands for "region convolutional neural network". Faster RCNN is a tried and proven object detection model type, the third iteration of the RCNN type architecture. Faster RCNN was picked for a number of reasons:

- Generally, when using object detection models, one has to strike a balance between speed and accuracy. Faster RCNN performs excellently for current needs in both categories.
- The data that is used to train is marked with bounding boxes and the data that is needed to continue training should be the same. This makes it easier to adapt other datasets into a suitable format. For example, The Mask RCNN model type is indeed more advanced but requires "masks" that have to be more clearly defined and therefore more difficult to prepare data for.
- The team already had personal experience using this particular model and were satisfied with the result.

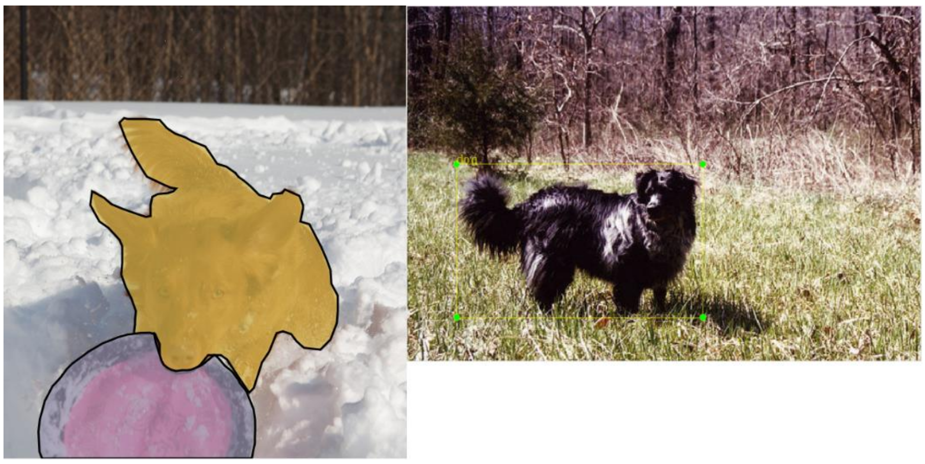


Figure 12. Images marked with masks (left) and bounding boxes (right) (Everingham, Gool, Williams, Winn, & Zisserman, 2012)

3.1.4 Model operation

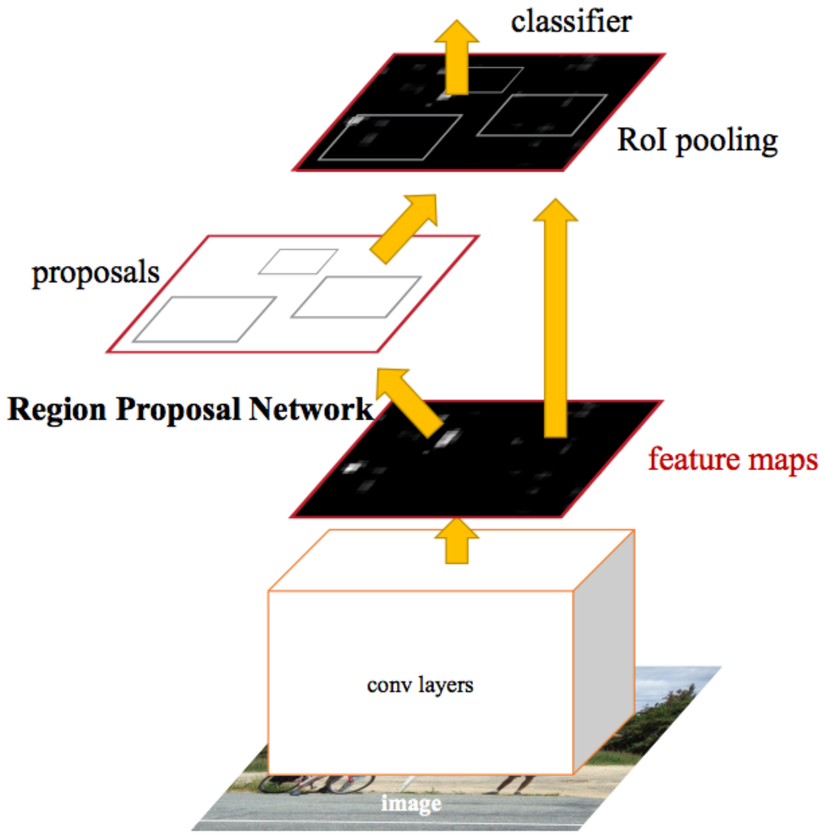


Figure 13. General outline of Faster RCNN architecture (Ren, He, Girshick, & Sun, 2016)

The general operations of the model can be outlined based on the following reports: “Speed/accuracy trade-offs for modern convolutional object detectors” which contains a general summary of different object detector architectures and “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks” written by the creators of Faster-RCNN. (Huang, et al., 2017) (Ren, He, Girshick, & Sun, 2016)

The inputs for these models are images which can be represented by multidimensional arrays (tensors). Each pixel in an image has a R, G, B value corresponding to the 3 color channels. For example, an image with the height of 800 and the width of 600 would be represented by the tensor [800x600x3]. The desired output is that the objects of interest are accurately classified as well as localized by bounding boxes coordinates.

The input begins by passing through layers of convolutional networks to generate feature maps. These layers do this by using “kernels” or “filters” to summarize the images without losing the general topological details of the image. The result has a much smaller dimension compared to the original, reducing the computational burden for later calculations.

The feature map is passed to the region proposal network. In the network, the image is processed by a feature extractor, which is essentially an internal pre-trained neural network. The choice of the feature extractor can really set models' performance apart from each other. Here the feature extractor uses what are called "anchors" of different scales and aspect ratios. These anchors will then "slide" over the feature map. Looking through the anchors, the model chooses anchors that are likely to contain the object and outputs region proposals as a set of bounding boxes, each with a score of how sure the model thinks there is an object inside. Note that the model still hasn't classified what the object inside the bounding box is, just that something is there.

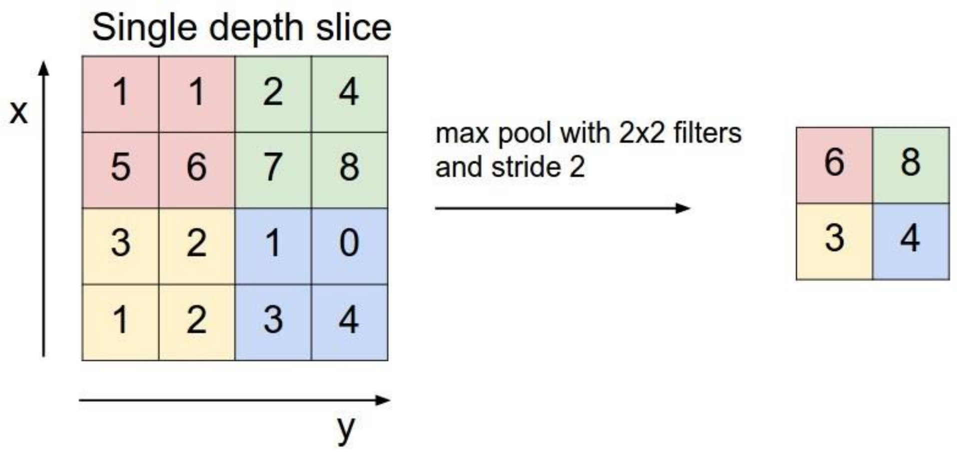


Figure 14. Max pooling example (Stanford University, 2020)

In the earlier iteration of the architecture, each region proposal would have to be cropped out of the feature map and pass them through to the classifier. It can be a very computationally expensive operation as an image can have many proposals. The model mitigates this with "region of interest pooling". Using the region proposals, crops are made from the feature map from the previous step. These crops are all scaled to a predefined size using "max pooling". They are then passed as feature maps to the classifier where their object class is determined and their bounding boxes adjusted to fit the object in the frame better.

3.2 **Methodology**

The model was pre-trained using the COCO (Common Object in COntext) dataset that covers 80 classes of objects such as person, airplane, computer, apple, etc. It contains about 123,000 images with about 886.000 instances of those objects in the images. (COCO Consortium, 2020) Most notably, those classes include cats and dogs. This means the pre-trained model was already trained to detect these animals. However, empirical data had shown that even though the model was decent at what it does, it could be improved by continuing to train the model further, aiming to specialize in detecting cats and dogs, which could result in better performance, improving processing speed and accuracy.

TensorFlow Object Detection API (TF OD API) was used to train the model. It is an open source framework built on top of TensorFlow, a machine learning library for Python. It is an open source repository containing various tools to ease the training and deployment of object detection models. The tools are written in Python. The TensorFlow version used was 1.14 running on Python 3.6. The pre-trained model is also taken from TensorFlow's object detection model zoo, specifically: "Faster RCNN Inception V2 COCO" (Google LLC, 2020). The name means that the model is using the Faster RCNN architecture with Inception V2 as its feature extractor, and the model was trained using the COCO dataset.

3.2.1 Dataset

In order to train the model, a dataset was needed. The building blocks of the dataset were images with bounding box annotations. In machine learning, these bounding boxes in training examples are often referred to as "ground-truths". A training example could be created with "LabelImg", an open source image annotation tool (Tzutalin, 2020).

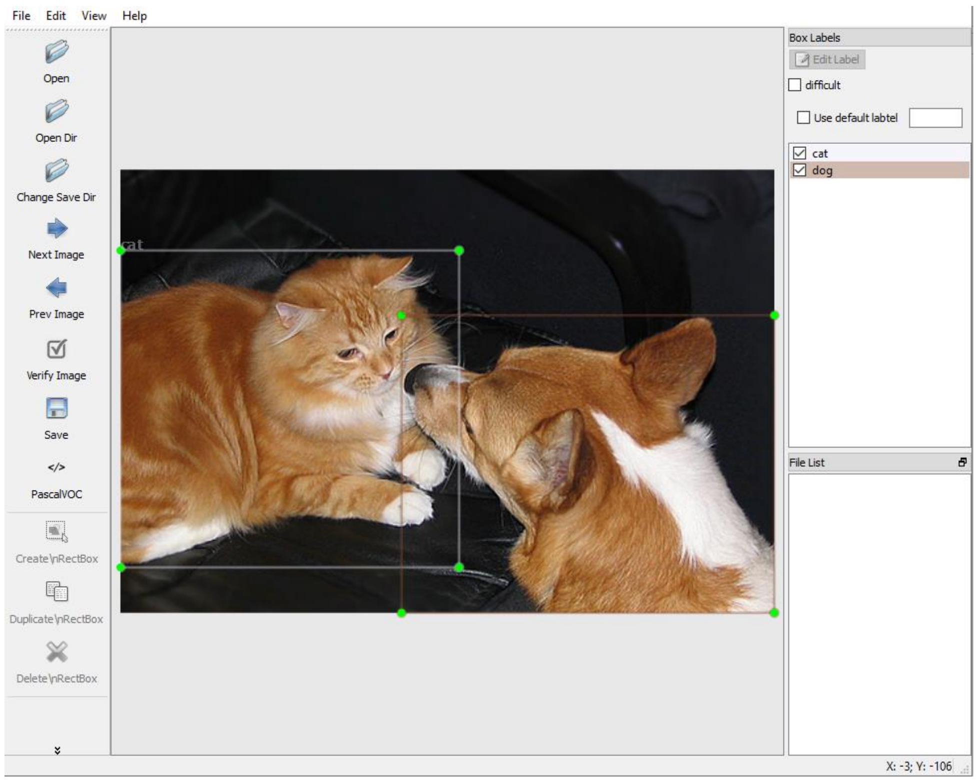


Figure 15. LabelImg example

The images were loaded into the program where the user could draw bounding boxes which indicated the object's location and assigned a class to each of those boxes. Having done so, the changes could be saved. The program outputted these annotations as xml files in PASCAL VOC format. Most notably, the xml file contained the coordinates (upper left and lower right points) of the bounding boxes. Refer to Appendix 1 as an example XML file.

There are many freely available datasets that can be used. The datasets that were chosen are:

- PASCAL VOC 2012 Challenge (Everingham, Gool, Williams, Winn, & Zisserman, 2012): A public dataset for the 2012 PASCAL VOC competition which contains 17,125 images with annotation, both bounding boxes and segmentation masks. The PASCAL VOC's dataset contains classes (a total of 20) and examples that were irrelevant and thus were removed. After filtering out the unnecessary data, there were 2,434 images left for usage.
- Stanford Dogs Dataset (Stanford University, 2020): A public dataset which contains 20,580 images of 120 breeds of dogs. Each breed had an its own label. Stanford's dataset already came with the bounding box annotations though some adjustments were still needed to be made. There was no need for the classification of individual dog breeds and so the label names were all changed to simply "dog". The "filename" xml element referred to something else rather than the name of the image. This problem was rectified.
- Kaggle Dogs Vs. Cats (Kaggle, 2020): A public dataset provided for an image classification contest. The dataset consisted of 25,000 images in total with 12,500 images of cats and 12,500 images of dogs. Because Kaggle's dataset was originally used for image classification, it did not have annotations. Therefore, the bounding box annotations had to be made image by image with LabelImg. During this process, some images that were thought to be unsuitable (too pixelated, corrupted, etc.) were removed. A good part of the dog images also went unused as it was considered that there were sufficient dog related data from Stanford's dataset. In the end, 12,426 cat images and 2,501 dog images were annotated, a total of 14,927 examples.

All in all, the dataset contained about 37,940 images of cats and dogs. In machine learning, data is often split into a train set and a validation set. The train set is used to teach the model how to do its task properly. The model would try to make predictions on an example, which are then compared with the actual bounding boxes coordinates. The model's parameters are then adjusted accordingly. On the other hand, the validation data is used to evaluate the model's ability against unfamiliar data. This set should not have any overlap with the training set and should not be used in training as it will create biases. As a general rule of thumb, the train/validation set ratio should be 80/20. However, depending on a variety of factors (such as model architecture, data variations, etc), the ideal ratio could be 70/30 or 95/5. Often the ratio which works best for the problem at hand can only be found by experimenting. During training, the ratio of about 80/20(for the train and the validation set respectively) was observed to be what works best for the model. With the same ratio applied to all the individual datasets, there were 30.500 training examples and 7.440 evaluation examples.

Additionally, TF OD API required the dataset to be in TFRecord (TensorFlow record) format. In this format, the training examples were serialized into large "record" files (100-200 MB) which marginally improved processing efficiency. The conversion was done with a Python script.

3.2.2 Configuring training process

With the dataset ready, configuration of the training job was needed. Notably, configuration could have major impact on the model's performance. The training process takes a "config" file to set parameters for the training process. There were many parameters that can be set such as numbers of classes, paths to external files, set anchor boxes' size and aspect ratios, etc. The most important parameters (also known as hyperparameters) are as follow:

- Number of training steps: determined the number of training steps the model will go through. During each step the model would process as many training examples as the "batch size" parameter. The model was updated at the end of each step. It was important to tune this parameter so that the model did not underfit or overfit (concepts that will be explained later).
- Batch size: the number of examples the model would train during each step. The larger the batch size, the more data was being fed into the model before update. In general, a larger batch size led to better model performance but the training process would be slower and memory consumption would increase. It was generally recommended that the batch size should be as high as the hardware can allow.
- Learning rate: determined how much the model should adjust itself after each training pass. A learning rate that was too high means adjustments may overshoot. If it was set too low, the model would have taken too long to reach the target accuracy. The learning rate could be set to change under certain conditions.

Although there are general rules of thumb as to how a user should set these parameters, fine-tuning the configuration for a specific machine learning problem could be a major obstacle during the training process since there can be many factors affecting the outcome. Arriving at the optimal training parameters was often a matter of trial and error.

3.2.3 Training and evaluation process

Having configured the training parameters and preparing a dataset, the training can begin. Using "model_main.py" in the TF OD API repository, the process can be initiated from the command line where training progress and status can be monitored (Google LLC, 2020). Periodically the program would output "ckpt" (checkpoint) files which contain the values of the model's parameters at that point. If for any reason, the training process was interrupted, it could be continued from that checkpoint file. The program also continuously outputs "events" files which can be loaded using TensorBoard, TensorFlow's visualization toolkit (Google LLC, 2020). Using TensorBoard, the model's training progress could be viewed in real-time.

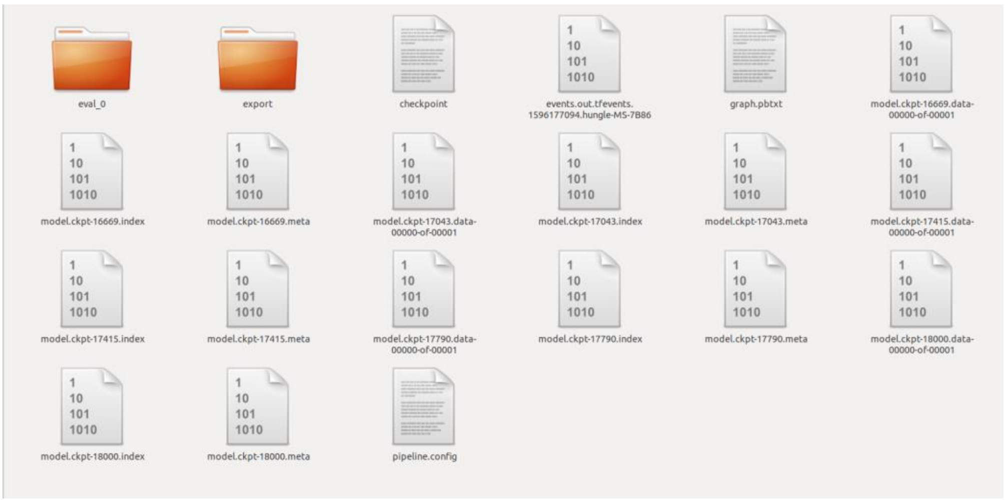


Figure 16. Example of training output

TensorBoard provides various measurements and visualizations. Various things could be viewed, such as the model's structure, its weight values and their distribution, prediction output examples and most importantly, the evaluation metrics for the model. These metrics are also displayed on the command line every time it saves a checkpoint file which is rather inconvenient. TensorBoard can plot these data points into graphs, allowing the user to view these metrics in a more visually-intuitive manner. These metrics are:

- Mean average precision (mAP)
- Mean average recall (mAR)
- Loss

AP and AR are defined by Pascal VOC. To understand what these metrics mean, it is necessary to understand how the model's prediction is being evaluated. First is the concept of Intersection over Union (IoU). IoU is the area in which the model's bounding box prediction of the object overlaps with the ground-truth in the training example divided by the combined area of both the boxes. The higher the IoU, the more accurately the model's prediction of the location of the object was.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Whenever the model makes a prediction, it places bounding boxes where it thinks the object of interest is. Attached to each of them is a class and a "confidence score" (a value between 0.0 and 1.0) which is how sure the model is about that prediction. Any box with a score lower than a certain threshold is filtered out. For any box that remains, using the ground-truth as a comparison, if its predicted class is correct and its IoU value is over a threshold, it is considered a "true positive". If the prediction fails to meet any of the 2 criteria, it is considered a "false positive". If the model fails to identify the presence of an object where it should have, it is considered a "false negative".

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

Precision is defined as the number of true positives divided by the sum of true positives and false positives. In other words, the higher the precision, the more likely the model's prediction is correct. On the other hand, recall is defined as the number of true positives divided by the sum of true positives and false negatives.

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

In this case, recall measures whether the model has detected all the relevant objects or not.

There is an inverse relation between precision and recall. At each confidence score threshold value there is a corresponding pair of precision and recall value which can be arranged as a measure called the precision/recall curve. Although assessing the model's performance using this curve is possible, it is not intuitive for comparing different models or monitoring performance over time. A numerical value would be better for these tasks which is what AP is. It is the averaged value of precision across all recall levels. More specifically, AP is defined as the area under an interpolated precision/recall curve. Similarly, AR is also used for model performance assessment. In short, AR is calculated by averaging recall values at IoU threshold from 0.5 to 1. The higher the AP and AR value, the more accurate the model. In addition, there are also the metrics mAP which is defined as the AP value averaged over all object classes. Same as mAP, mAR is defined as the mean of AR over all object classes. mAR and mAP are 2 of the metrics that TF OD API outputs.

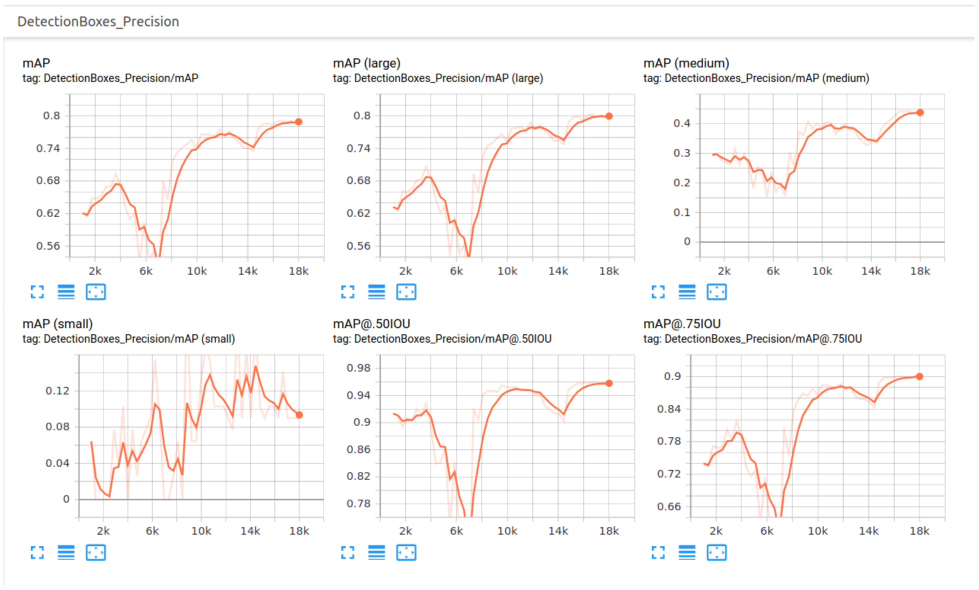
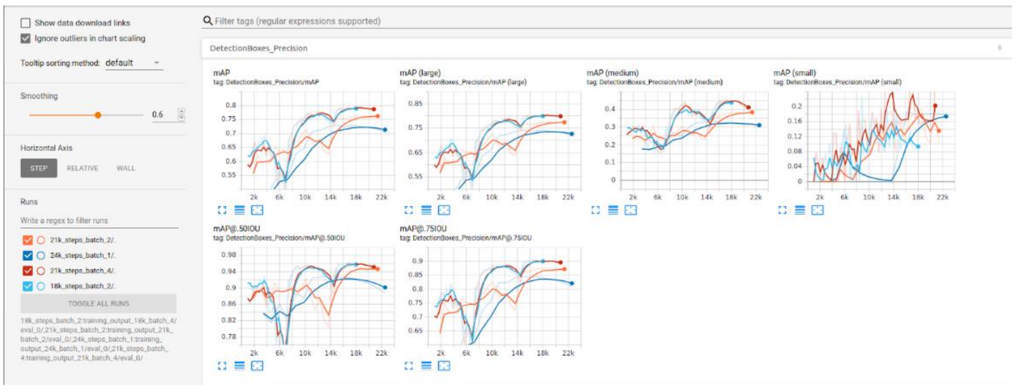


Figure 17. Average precision(mAP) on TensorBoard

TF OD API uses COCO object detection evaluation metrics as the evaluation metrics which is a variation of PASCAL VOC's own metrics. (COCO Consortium, 2020) They are:

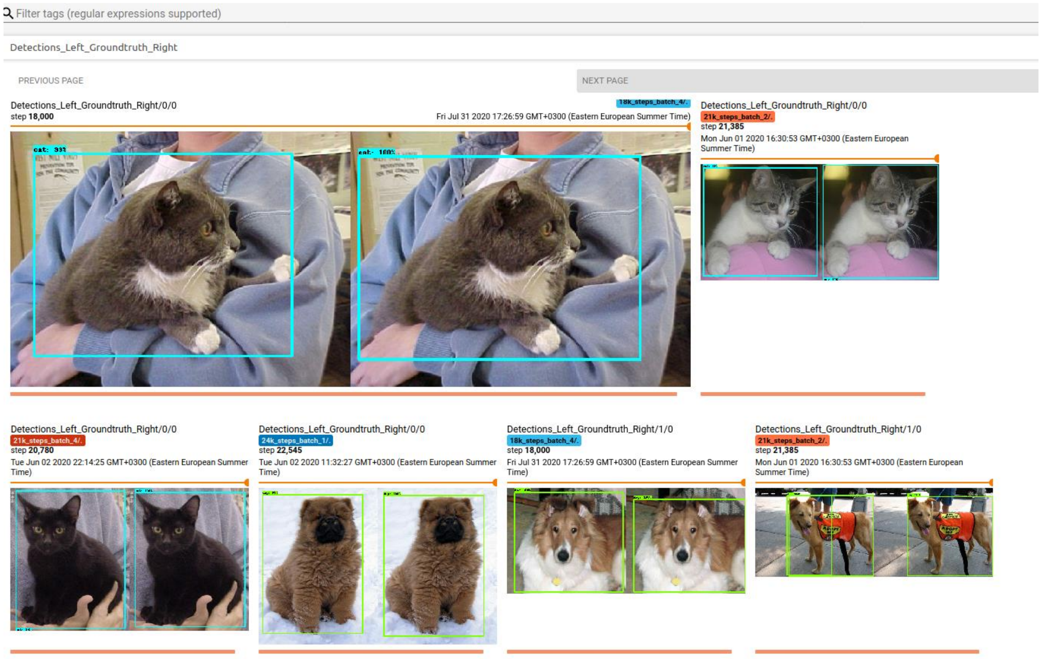
- mAP at IoU = .5:.05:.95: metric calculated by averaging 10 mAP values. Each value is taken from the IoU thresholds between 0.5 to 0.95 with a 0.05 increment between each threshold value i.e 0.5, 0.55, 0.6...,0.95. This is the metric that COCO considers the most important. The reasoning is that the way the metric is calculated would reward models with more accurate object localization more and penalize ones that don't more.
- mAP at IoU = .5: mAP at IoU threshold of 0.5. This is the metric used by PASCAL VOC.
- mAP at IoU = .75: mAP at IoU threshold of 0.75. This is the “strict metric”.
- mAR at Max = 1: mAR given over 1 detection per image, averaged over classes and IoUs.
- mAR at Max = 10: mAR given over 10 detections per image.
- mAR at Max = 100: mAR given over 100 detections per image.
- mAP and mAR small: mAP and mAR for small objects (area < 32^2 pixels).
- mAP and mAR medium: mAP and mAR for medium objects (32^2 < area < 96^2 pixels).
- mAP and mAR large: mAP and mAR for large objects (area > 96^2 pixels).

Loss, TF OD API's third evaluation metric, measures how much the model's predictions deviated from the ground-truths. More specifically, things like how much the model's prediction bounding boxes overlap with the ground-truths, how often the model assigns the wrong class to its prediction, etc. all contribute to loss. The lower the loss, the fewer mistakes are made. TF OD API outputs multiple kinds of loss which are loss values related to region proposals, loss values related to classification, and loss in general. In addition, TensorBoard allows the user to compare these evaluation metrics between different models which is quite a useful feature.


Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
Runs
21k_steps_batch_2/.
24k_steps_batch_1/.
21k_steps_batch_4/.
18k_steps_batch_2/.
18k_steps_batch_2:training_output_18k_batch_4/eval_0/,21k_steps_batch_2:training_output_21k_batch_2/eval_0/,24k_steps_batch_1:training_output_24k_batch_1/eval_0/,21k_steps_batch_4:training_output_21k_batch_4/eval_0/

DetectionBoxes_Precision
mAP — tag: DetectionBoxes_Precision/mAP
mAP (large) — tag: DetectionBoxes_Precision/mAP (large)
mAP (medium) — tag: DetectionBoxes_Precision/mAP (medium)
mAP (small) — tag: DetectionBoxes_Precision/mAP (small)
mAP@.50IOU — tag: DetectionBoxes_Precision/mAP@.50IOU
mAP@.75IOU — tag: DetectionBoxes_Precision/mAP@.75IOU

Figure 18. TensorBoard loading multiple event files at the same time for comparisons


Detections_Left_Groundtruth_Right

Detections_Left_Groundtruth_Right/0/0 — 18k_steps_batch_4/. — step 18,000 — Fri Jul 31 2020 17:26:59 GMT+0300 (Eastern European Summer Time)
Detections_Left_Groundtruth_Right/0/0 — 21k_steps_batch_2/. — step 21,385 — Mon Jun 01 2020 16:30:53 GMT+0300 (Eastern European Summer Time)
Detections_Left_Groundtruth_Right/0/0 — 21k_steps_batch_4/. — step 20,780 — Tue Jun 02 2020 22:14:25 GMT+0300 (Eastern European Summer Time)
Detections_Left_Groundtruth_Right/0/0 — 24k_steps_batch_1/. — step 22,545 — Tue Jun 02 2020 11:32:27 GMT+0300 (Eastern European Summer Time)
Detections_Left_Groundtruth_Right/1/0 — 18k_steps_batch_4/. — step 18,000 — Fri Jul 31 2020 17:26:59 GMT+0300 (Eastern European Summer Time)
Detections_Left_Groundtruth_Right/1/0 — 21k_steps_batch_2/. — step 21,385 — Mon Jun 01 2020 16:30:53 GMT+0300 (Eastern European Summer Time)

Figure 19. TensorBoard displaying examples of model predictions

With these metrics, a good idea of how the model was performing could be had. However, in machine learning, one should be aware of overfitting and underfitting. Overfitting occurs when the model learns the data too well, often occurring when it is trained longer than necessary. The model misinterprets the random noises and variations in the data as learnable patterns where no such pattern exists. The model may get very good at its task on the training data but would perform worse on the testing data. Underfitting is the opposite problem. Models suffering from underfitting have poor performance due because it has not learned the underlying patterns and features of the objects of interest. Underfitting is generally rectified with more training. The end goal for the training is achieving an optimal state where the model is trained just well enough, neither underfitted nor overfitted, performing well on unfamiliar data.

The training process would repeat many times as needed. After several training iterations, each with some differences in parameter, the resulting models' evaluation data could be examined and observed what changes worked best and what didn't. It was a process of continuous refinement. When a model was found to work well, because the plan was to use it in OpenCV, the model was need to be exported into a usable format. One of TF OD API tools would facilitate to do just that. "export_inference_graph.py" would take in checkpoint files and a training configuration file and outputs a "frozen inference graph" (.pb extension) which would contain the model's architecture and variables, ready to be deployed to other platforms and programs (Google LLC, 2020).

3.2.4 Process summary

Having established the necessary machine learning concepts, the general workflow could be summarized. The task was to detect cats and dogs whenever they are in the image frame. First, a pre-trained object detection model was chosen that was appropriate. In this case, it was Faster RCNN pre-trained on the COCO dataset. The model can be trained using one's own dataset. A dataset consists of a training set where the model would learn the features and patterns of the object classes in question (cats and dogs in this case) and a validation set where the model's performance would be evaluated by performing object detection on the examples. The training process itself was done using TensorFlow Object Detection API and was both time and computationally intensive. Completing a training run of a model often took a whole day to complete and would take up most of the computer processing power. TensorBoard was used to monitor the model's training progress as well as evaluation metrics. Based on these metrics, the training parameters was fine-tuned to try and improve the model's effectiveness at the task at hand. Having chosen the model that achieved the best results, the model was then imported into OpenCV after a format conversion so that it could be used in the program.

3.2.5 Results

After multiple training iterations, a model that performed well was chosen. The general axiom of hyperparameters having major influence on training results held true. When the general configuration of the training process was figured out, it was a matter of tweaking the batch size, number of training steps and learning rate to achieve the best results. Here are some examples of the object detection model in action. The number is the confident score from the model.

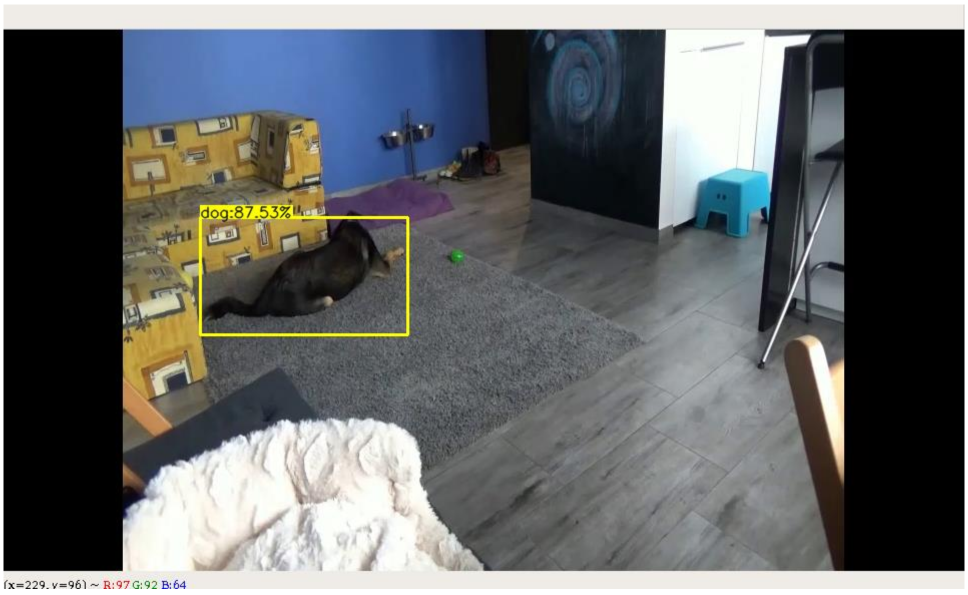


Figure 20. Example of the model functioning correctly

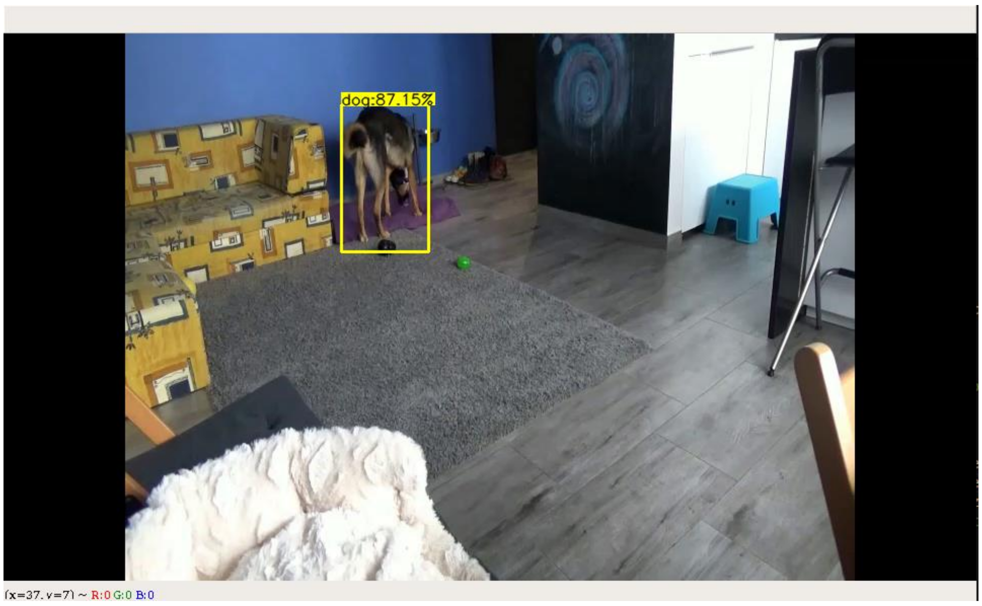


Figure 21. Example of the model functioning correctly

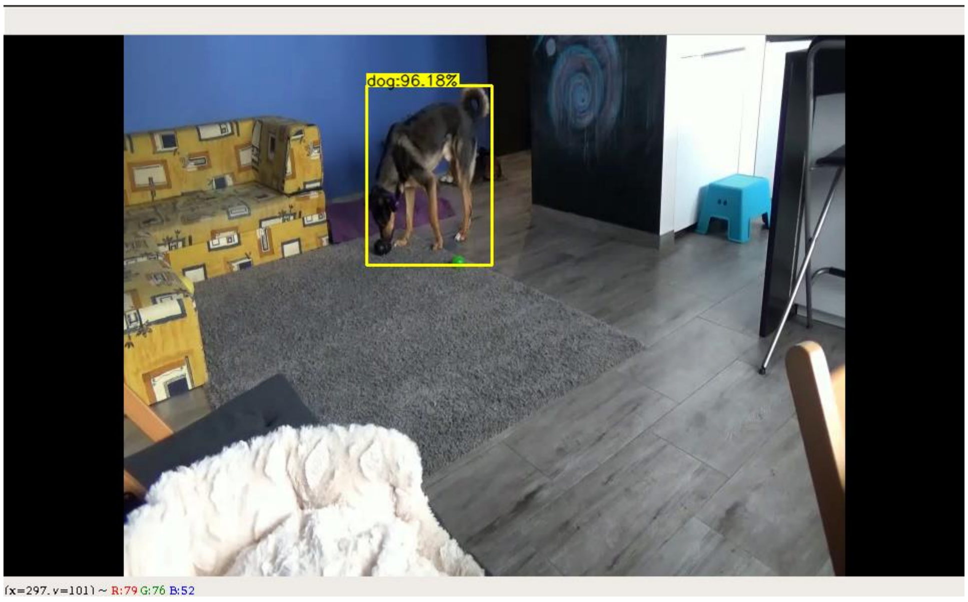


Figure 22. Example of the model functioning correctly

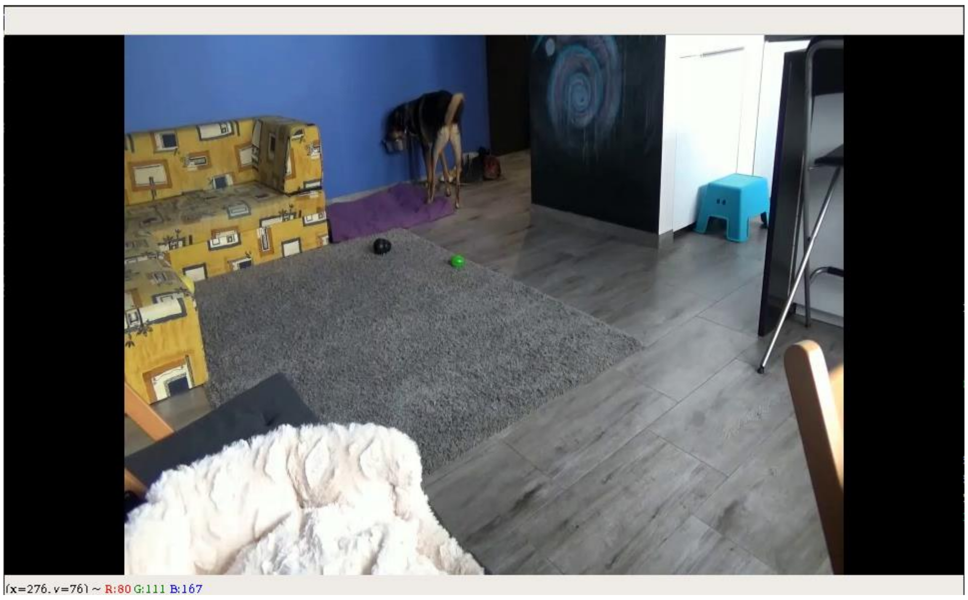


Figure 23. Example of the model failing to detect the object

The model performed well in general. It was able to detect dogs and cats from the multiple angles. However, the model was not perfect. It sometimes had problem detecting animals from behind. The reason was that the dataset was somewhat lacking in this regard, something that could be rectified in later versions.

Even with this issue, the more important problem was how object detection should be integrated into the existing program. Running object detection was quite resource intensive, requiring a somewhat decent hardware to run smoothly. Even with good hardware, the process was not fast. The time it took for the model to process 1 image frame was about

1,5 seconds. Given that there could be at least 24 frames for every second in a video, processing time could be high if it was left running as is.

Cursory steps were taken to alleviate these problems and some preliminary testing was done. Unfortunately, there was not enough time in the end, with the other parts of the project taking longer to complete than expected. Even though the research and the training for the object detection model had been done, it had to be left out in the end.

4 VIDEO TRIMMING

The original goal of this project was to make a program that can take out relevant sections of a video and stitch them together into a new file. The detection algorithms were used in order to pinpoint the location of the pertinent parts. Those segments were then extracted and merged into a video containing only what were needed.

4.1 FFmpeg

The video cutting and merging are handled by FFmpeg, a free and open-source software used for manipulating video, audio and other multimedia files. FFmpeg stands for Fast Forward Motion Picture Expert Group. Many common software and websites use FFmpeg in order to deal with audio-visual files, such as VLC, Google Chrome, YouTube and many more. (FFmpeg team, 2020)

4.2 Trimming

Because FFmpeg didn't have a built-in trimming function, some extra steps were taken to achieve the desired results.

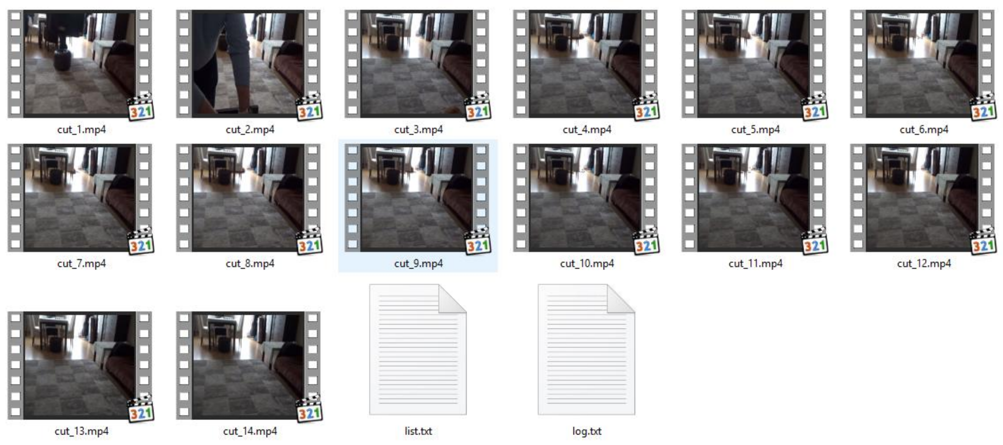
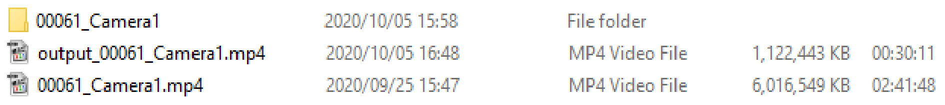


Figure 24. Individual sections of motion cut from the original video

First, the video was cut into smaller parts according to the timestamps taken from the detection section. Then those sections were concatenated into the final video. Thus, creating the effect that the video got trimmed of the unnecessary parts.



00061_Camera1	2020/10/05 15:58	File folder		
output_00061_Camera1.mp4	2020/10/05 16:48	MP4 Video File	1,122,443 KB	00:30:11
00061_Camera1.mp4	2020/09/25 15:47	MP4 Video File	6,016,549 KB	02:41:48

Figure 25. Output folder

As seen in Figure 25, the output folder had a temporary folder, containing the cut sections and other logs, which would be deleted afterwards. The trimmed and original video were put side by side in order to show the difference of before and after trimming.

5 EXTRA FEATURES

The commissioning party requested several additional features for their own reasons.

5.1 First part skipping

The commissioning party required that the first 90 minutes of the videos are not to be processed. Therefore, before processing the videos, a new video was created without the first 90 minutes, then performing detection on the shorter video.

5.2 Period before and after motion

The commissioning party required that when detecting motions, there should be a 30-second buffer zone before and after each motion section in order to observe motion better. In order to achieve this, when writing down the timestamps for the beginning and ending of motion, the numbers were pushed forward 30 seconds for the former and 30 seconds backwards for the latter.

6 SERVER

With the main program figured out, it needed to be set up to work on a web server. A system for users to upload videos to the server was also needed. The main program had to automatically and periodically run on a predetermined folder on the server, which would process any video files in it using motion detection and would output the result into a separate folder. It was also decided that with appropriate credentials, the server can be accessed to view the program's logs as a web page as one way to determine its status as the video processing could take a long time. Hence the need for a frontend and backend for the web server.

In order to create the web server, Django was chosen, a popular Python web framework (Django Software Foundation, 2020). This helped simplify the programming process for the server a bit. Django still provided useful features out of the box for the server such as a built-in users and passwords system with related administrator tools. The frontend of the server was a simple web page that allowed users to log in and view log files and see what the server had been doing. On the backend side of things, the server would look for new video files and process them at set intervals. However, the web server cannot serve static files (i.e. CSS files) with just Django as the framework. A full-stack solution like nginx + Gunicorn + Django would be needed to do so. In the end, only Gunicorn was used as the HTTP server and Django as the framework. The problem was too trivial to add unnecessary complexity to a simple server.

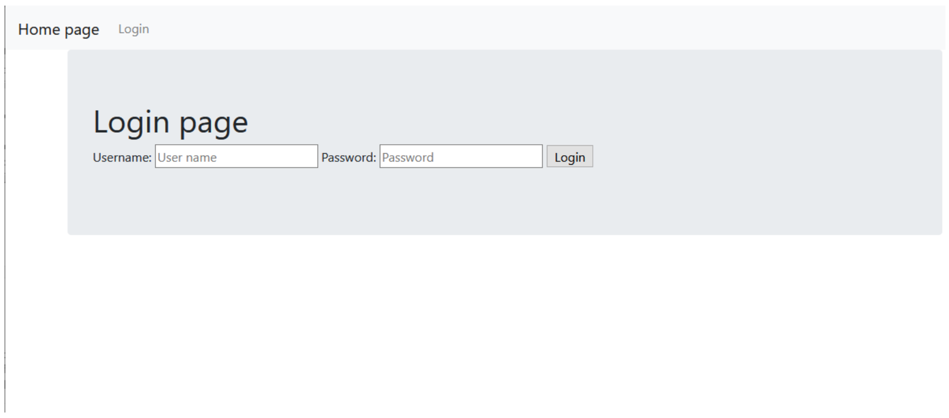
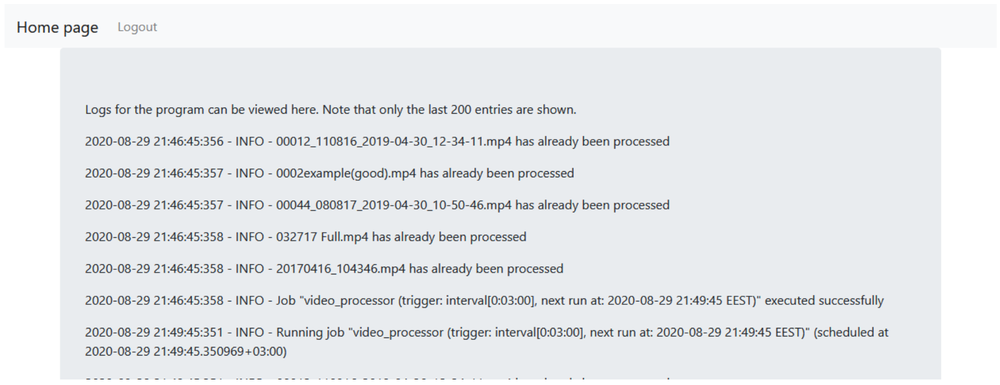


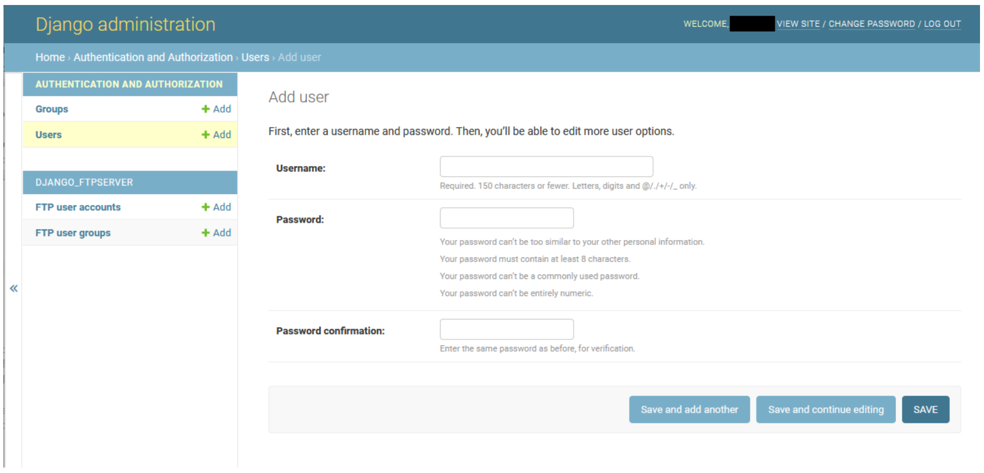
Figure 26. Web server's log in page



Logs for the program can be viewed here. Note that only the last 200 entries are shown.

```
2020-08-29 21:46:45:356 - INFO - 00012_110816_2019-04-30_12-34-11.mp4 has already been processed

2020-08-29 21:46:45:357 - INFO - 0002example(good).mp4 has already been processed

2020-08-29 21:46:45:357 - INFO - 00044_080817_2019-04-30_10-50-46.mp4 has already been processed

2020-08-29 21:46:45:358 - INFO - 032717 Full.mp4 has already been processed

2020-08-29 21:46:45:358 - INFO - 20170416_104346.mp4 has already been processed

2020-08-29 21:46:45:358 - INFO - Job "video_processor (trigger: interval[0:03:00], next run at: 2020-08-29 21:49:45 EEST)" executed successfully

2020-08-29 21:49:45:351 - INFO - Running job "video_processor (trigger: interval[0:03:00], next run at: 2020-08-29 21:49:45 EEST)" (scheduled at
2020-08-29 21:49:45.350969+03:00)
```

Figure 27. User's page


Django administration
WELCOME [redacted] VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Authentication and Authorization › Users › Add user
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
DJANGO_FTPSERVER
FTP user accounts + Add
FTP user groups + Add
«
Save and add another
Save and continue editing
SAVE

Add user

First, enter a username and password. Then, you'll be able to edit more user options.

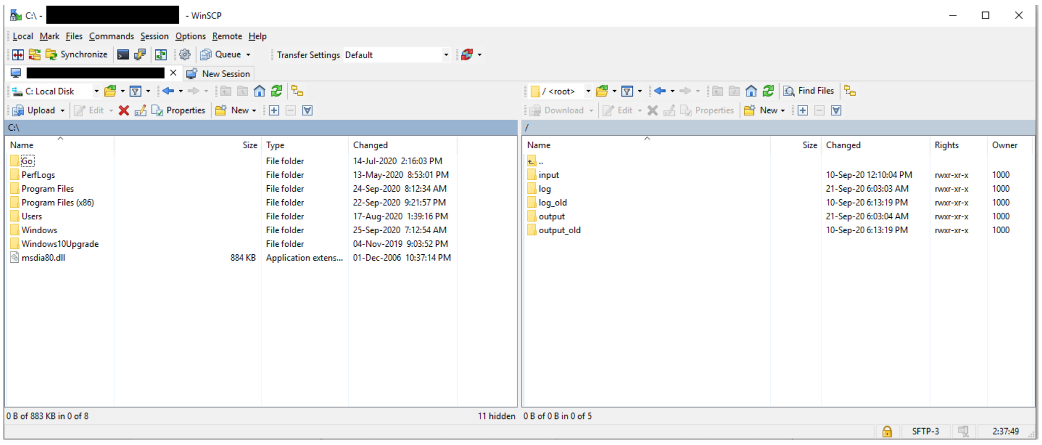
Username:
Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:
Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.

Password confirmation:
Enter the same password as before, for verification.

Figure 28. Django's administrative tools

To facilitate file upload and download for the server, SFTP was used as the file transfer protocol. SFTP stands for SSH File Transfer Protocol. It would allow encrypted (and therefore secure) file transfers between machines. A SFTP server was implemented using rspivak's stub SFTP server code as the base which was using the Python library "paramiko" and would work alongside the web server (Pointer & Forcier, 2020; rspivak, 2017). With the right credentials, users could log in using a SFTP client to access the SFTP server. On the server, there were three folders: "input" which would be where the user should upload video files for the main program to process, "output" which contained the processed video and "log" which contained the video timestamps of events of interest. Early on, a server using FTP, which is SFTP's preceding file transfer protocol, was briefly used but eventually discarded due to security concerns.



C:\

Name	Size	Type	Changed
Go		File folder	14-Jul-2020 2:16:03 PM
PerfLogs		File folder	13-May-2020 8:53:01 PM
Program Files		File folder	24-Sep-2020 8:12:34 AM
Program Files (x86)		File folder	22-Sep-2020 9:21:57 PM
Users		File folder	17-Aug-2020 1:39:16 PM
Windows		File folder	25-Sep-2020 7:12:54 AM
Windows10Upgrade		File folder	04-Nov-2019 9:03:52 PM
msdia80.dll	884 KB	Application extens...	01-Dec-2006 10:37:14 PM

/

Name	Size	Changed	Rights	Owner
..				
input		10-Sep-20 12:10:04 PM	rwxr-xr-x	1000
log		21-Sep-20 6:03:03 AM	rwxr-xr-x	1000
log_old		10-Sep-20 6:13:19 PM	rwxr-xr-x	1000
output		21-Sep-20 6:03:04 AM	rwxr-xr-x	1000
output_old		10-Sep-20 6:13:19 PM	rwxr-xr-x	1000

0 B of 883 KB in 0 of 8   11 hidden   0 B of 0 B in 0 of 5

SFTP-3   2:37:49

Figure 29. SFTP server accessed from WinSCP, a free SFTP client

Through this, all the individual pieces that made up the project had been put together. The process would begin with the user uploading a video file through the SFTP server. The web server would then process the video with the motion detection component, a task that it would do regularly.

The user could view the progress on the web page and could download the output videos through the same SFTP server.

7 CONCLUSION

The goal of this project was to make a web server where the commissioning party could upload videos of home security and be able to retrieve newly generated videos with just the parts of the original videos containing motions.

As for the web server, a simple HTTP web page was made where users can login to view the program logs. The main program periodically ran in the background, looking for any new videos in the SFTP server's directory. An SFTP server was also made and it used the same user credentials as the web server.

The program has the ability to read through all the files in the input folder, pick out the media files and then perform motion detection on them. Afterwards new videos were produced and was put into the output folder. Furthermore, logs were put into the log folder, including timestamps for the videos, a list composed of the names of the videos which had been processed, and a log file containing the information on the events of the program.

Research on object detection was done and a working prototype was built. However due to the long duration of processing time, extensive testing had not been done. Therefore, it was decided that it was left out of the main program. If the program can be continued to be developed, integrating object detection would be one way the program could be expanded upon.

After working through this project, a better understanding of the inner workings of multiple pieces of software like OpenCV, TensorFlow, Django was gained, as well as more experience working with web development and video processing.

References

Artificial neural network. (2020). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Artificial_neural_network

COCO Consortium. (2020). *Detection Evaluation*. Retrieved from COCO - Common Objects in Context: https://cocodataset.org/#detection-eval

Django Software Foundation. (2020). *Django*. Retrieved from Github: https://github.com/django/django

Everingham, M., Gool, L. v., Williams, C., Winn, J., & Zisserman, A. (2012). *Visual Object Classes Challenge 2012 (VOC2012)* . Retrieved from http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html

FFmpeg team. (2020). *FFmpeg*. Retrieved from Github: https://github.com/FFmpeg/FFmpeg

Google LLC. (2020). *TensorBoard*. Retrieved from Github: https://github.com/tensorflow/tensorboard

Google LLC. (2020). *TensorFlow Object Detection API*. Retrieved from Github: https://github.com/tensorflow/models/tree/master/research/object_detectio n

Heinisuo, O.-P. (2020, October). *skvark/opencv-python: Automated CI toolchain to produce precompiled opencv-python, opencv-python-headless, opencv-contrib-python and opencv-contrib-python-headless packages.* Retrieved from Github: https://github.com/skvark/opencv-python

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., . . . Research, G. (2017, April 25). *Speed/accuracy trade-offs for modern convolutional object detectors*. Retrieved from https://arxiv.org/pdf/1611.10012.pdf

Kaggle. (2020). *Dogs Vs. Cats*. Retrieved from https://www.kaggle.com/c/dogs-vs-cats/data

Motion Detection. (2020). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Motion_detection

OpenCV team. (2020). *opencv/opencv: Open Source Computer Vision Library*. Retrieved from Github: https://github.com/opencv/opencv

Pointer, R., & Forcier, J. (2020). *Paramiko*. Retrieved from Github: https://github.com/paramiko/paramiko

Ren, S., He, K., Girshick, R., & Sun, J. (2016, January 6). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* Retrieved from https://arxiv.org/pdf/1506.01497.pdf

rspivak. (2017). *sftpserver*. Retrieved from Github: https://github.com/rspivak/sftpserver

Stanford University. (2020). Retrieved from CS231n Convolutional Neural Networks for Visual Recognition: https://cs231n.github.io/convolutional-networks/

Stanford University. (2020). *Stanford Dogs dataset for Fine-Grained Visual Categorization*. Retrieved from http://vision.stanford.edu/aditya86/ImageNetDogs/

Tzutalin. (2020). *LabelImg*. Retrieved from Github: https://github.com/tzutalin/labelImg

Appendix 1

Example XML File

```
<annotation verified="yes">
<folder>test</folder>
<filename>0_c.jpg</filename>
<path>F:\kagglecatsanddogs_3367a\PetImages\test\0_c.jpg</path>
<source>
		<database>Unknown</database>
</source>
<size>
		<width>500</width>
		<height>375</height>
		<depth>3</depth>
</size>
<segmented>0</segmented>
<object>
		<name>cat</name>
		<pose>Unspecified</pose>
		<truncated>0</truncated>
		<difficult>0</difficult>
		<bndbox>
				<xmin>119</xmin>
				<ymin>18</ymin>
				<xmax>358</xmax>
				<ymax>241</ymax>
		</bndbox>
</object>
</annotation>
```