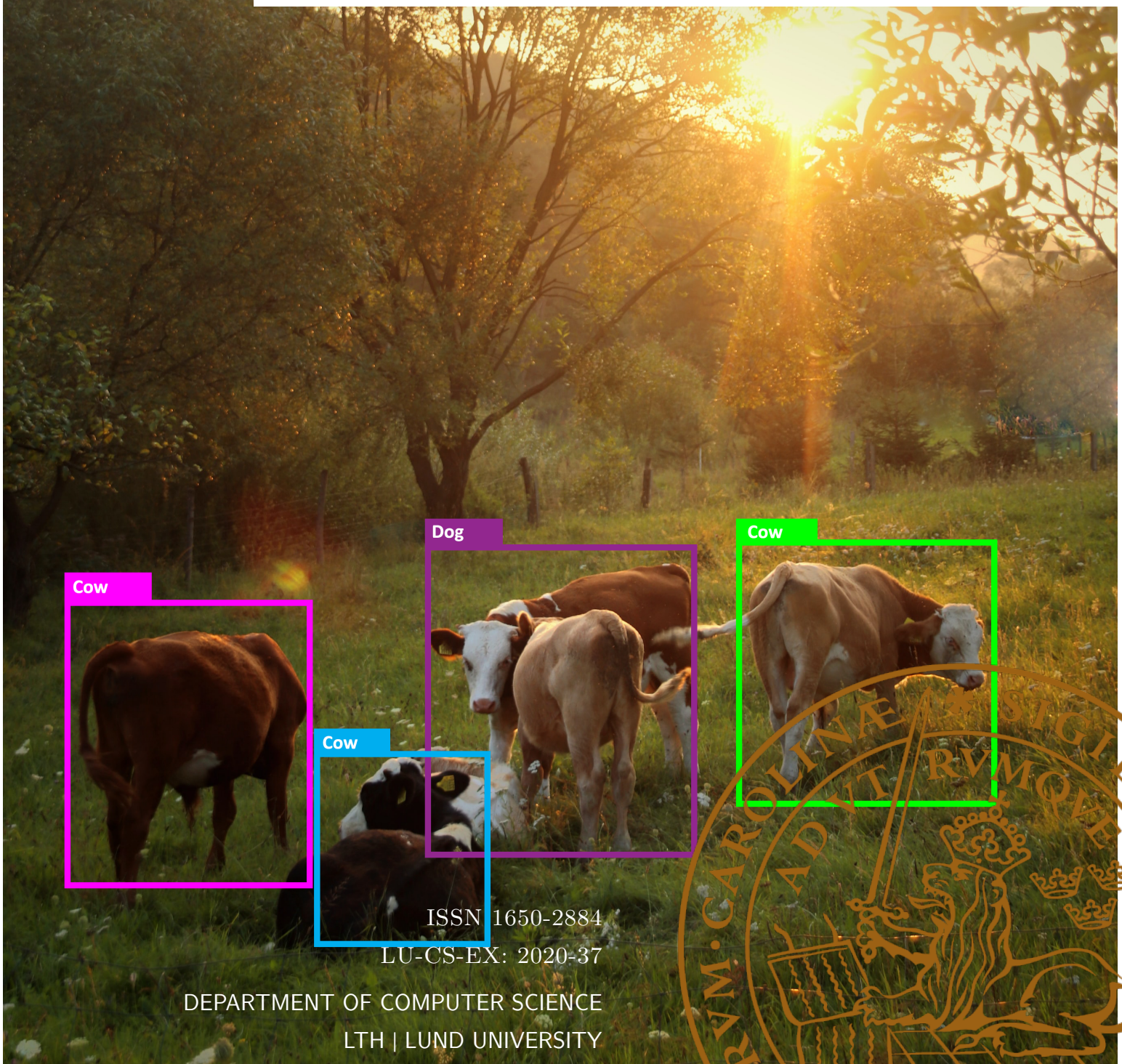
MASTER'S THESIS 2020

Motion Detection Alarm Verification Using Deep Learning in Surveillance Systems

Erik Rosengren, Jonathan Strandberg



ISSN 1650-2884
LU-CS-EX: 2020-37

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-37

Motion Detection Alarm Verification Using Deep Learning in Surveillance Systems

Erik Rosengren, Jonathan Strandberg

Motion Detection Alarm Verification Using Deep Learning in Surveillance Systems

Erik Rosengren
dat15ero@student.lu.se

Jonathan Strandberg
dat15jst@student.lu.se

July 1, 2020

Master's thesis work carried out at Axis Communications AB.

Supervisors: Jon Lindeheim, jon.lindeheim@axis.com
Jörn Janneck, jorn.janneck@cs.lth.se

Examiner: Mathias Haage, mathias.haage@cs.lth.se

Abstract

AXIS Companion is a cloud video management software which features configurable push notifications when something triggers a motion alarm. Motion alarms are however prone to false alarms which can be annoying for the end user. This thesis proposes using object detection based on deep convolutional neural networks to filter the alarms in order to lower the number of false alarms the end user receives. The object detection is run on a recording unit present in Axis surveillance systems.

Several networks such as MobileNetV1 through MobileNetV3, YOLOv3 and InceptionNetV2 were compared against each other on manually annotated video with different lighting conditions to see which network performed best. Different optimizations were also compared to find the most optimal combination of networks and optimizations.

The results were that MobileNetV3 was the most effective network, it produced around 0.3% false alarms and around 60% true alarms and did the best on night scenes.

Keywords: Object Detection, Deep Learning, Neural Networks, CCTV, Surveillance, Axis Communications, AXIS Companion, Alarm Verification, Motion Detection

Acknowledgements

We would like to thank Jörn Janneck and Jon Lindeheim for their great supervision, feedback and guidance during this thesis as well as their valuable help in acquiring the information that we needed in order to complete our work. We would also like to thank Marcus Wiedner for his help in setting up this thesis and acquiring the materials we needed to complete it. Lastly we would like to thank Emma Friberg for her proof-reading of this thesis.

Contents

Chapter 1
Introduction

As surveillance cameras become more advanced and gain better image quality than ever before, the question of how to improve the surveillance experience for the end user arises. With the ability to have several cameras recording high definition video at all times it becomes harder to continuously monitor all video feeds for suspicious activity. Not all small businesses can afford to have security guards monitor their surveillance cameras at all times, and might instead opt to rely on motion detection alarms to be able to react to a possible break-in. Motion detection alarms are highly effective in many scenarios but they are prone to false alarms which can be annoying to the end user. After all, who wants to be woken up in the middle of the night because a plastic bag blew past their camera?

1.1 Definitions and Terminology

Performance *is the amount of compute power needed for the object detection.*
Accuracy *refers to the classification accuracy of the object detection.*
False alarm *refers to when an object triggers an alarm when it should not.*
True alarm *refers to when an object triggers an alarm when it should.*
Missed alarm *refers to when an object does not trigger an alarm when it should.*

1.2 Axis Communications

Axis Communications is a video surveillance company based in Lund, Sweden that specializes in network cameras. In their product range are several different types of cameras ranging from simple and small indoor cameras to explosion-resistant thermal cameras for use in extreme environments. Axis also provides several software products to manage and configure surveillance systems, one of which is AXIS Companion.

1.2.1 AXIS Companion

AXIS Companion is a cloud video management system with a focus on smaller surveillance systems such as those one might find at a gas station, small store or in someone's home and supports up to 16 cameras. Most companion systems contain a so called AXIS Companion recorder which is a device with a large hard-drive that can handle recordings of several cameras at once. A maximum of 8 cameras can be connected to a recorder at a time. AXIS Companion supports cameras of very different capabilities and hardware and offers features such as remote live streaming and configuration which allows customers to view live streamed video and configure their cameras from anywhere in the world. AXIS Companion also includes an app for iOS and Android where users can view live streamed video from their cameras and receive push notifications when their cameras detect motion. Push notifications can be enabled during specific times for each camera, so for example only after office hours on weekdays.

A new recorder is under development with significantly better hardware than the previous one and it is supposed to replace the old recorder when it is released. The specific hardware components of the new recorder can be found in appendix B.1.

1.3 Problem Statement

The problem this thesis aims to solve is that of false alarms generated by motion detection. As mentioned earlier, false alarms can be very annoying for the end user and the intent is to lower the number of false alarms that the user experiences. This can be done in several ways, some of which are already implemented in Axis software such as swaying, small and short-lived objects being exempt from detection. This thesis will try to solve the problem by applying modern object detection using deep convolutional neural networks.

Some competing surveillance systems have already implemented object detection for this purpose, however, the main difference in existing solutions and the one proposed here is where the detection computations take place. Some competitors to Axis offer cloud solutions where all detections are sent to the cloud for processing and then the result is sent back to the software. Others do the computations directly on the camera, if the camera has hardware good enough to do so. This thesis will focus on doing so on the new recorder mentioned above since the solution then works with any type of camera.

1.3.1 Research Questions

With the problem defined it is possible to state which questions are to be answered in this thesis. These are the following in no particular order:

- **What is the best network for use in filtering of motion detection alarms both in terms of performance and accuracy.**
- **Which optimizations increase either the performance or the accuracy (or both)?**
- **Which customizable parameters generate the best results according to the metrics defined in Section 2.5?**

- **How many false and true alarms does the best network and optimizations produce on average?**
- **How are the networks affected by different lighting conditions and weather?**

1.4 Related Work

1.4.1 Neural Networks

Neural networks (sometimes called an Artificial Neural Networks) are a kind of computing system that can "learn" from examples. The basic structure of a Neural Network is several layers of nodes each with connections to **every** node in the next layer (except for the output layer which has no forward connections). Each connection has a weight and it is this weight that is updated during training to allow the neural network to "learn" what output to generate [5]. Training is done by taking inputs to which the output is known (training data) and feeding it through the network, at each step multiplying the value of the node with each connection and its corresponding weight to calculate the value of the next node. When this is done the resulting output nodes are compared to the known correct output in the training data and if these are not equal then the weights are altered through a method called back-propagation. This process is repeated until the network produces outputs that are near-equal to the correct answers from the training data [5]. Problems relating to over-fitting and regularization are not relevant to this thesis and will therefore not be introduced here.

1.4.2 Convolutional Neural Networks

Convolutional Neural Networks are a kind of neural network that instead of using fully connected layers use two dimensional kernel convolutions to connect layers. Instead of training weights for the entire layer the network only trains the kernels for the convolution layers which are also represented in two dimensions [5]. The kernel convolutions can be seen as a sliding filter across the image (a matrix) with the output being the sum of the products between each filter element and each image element. This way the kernel weights can be reused, less calculations need to be done compared to fully connected layers and spatial information is propagated to the next layer meaning the CNN can do its own feature extraction for use when classifying [5]. Repeating several convolutional layers results in down-sampling of the image (unless padding is used) while leading to the network extracting larger features. The output of several convolutional layers then represents the existence of a certain feature in the image. At the end of several convolution layers it is then possible to flatten the two dimensional layers into a one dimensional layer and then use fully connected layers to make a classifier for the features [5].

1.4.3 MobileNet

MobileNet developed by Google is an object detection network for use in lower power devices such as embedded systems or in smart phones while still providing almost real time object detection. MobileNet uses a scalable approach with a width multiplier $\alpha = (0, 1]$ that

controls the total width of the network, where a value $\alpha = 1$ denotes a standard width MobileNet. Decreasing the width of MobileNet scales down the number of hyper-parameters to be trained significantly while not losing too much of the original accuracy of the network [9]. This flexibility allows users of the network to scale the balance between performance accuracy depending on their particular needs.

1.4.4 Post Training Quantization

A big hurdle for neural networks is the memory and computational requirements. A recent approach to solve this has been to apply post training quantization [2]. Quantization in this context is a method of reducing the number of bits used to represent a number thereby reducing the space required but also reducing the possible set of values the number can represent. The act of applying post training quantization to a pre-trained network is applying quantization to the network weights and activation functions of a trained network. A typical way to represent a weight in a neural network is with a 32bit floating point value, with quantization this can be reduced to a 16bit floating point, or even an 8bit number. This can, in the most extreme cases, reduce the amount of space needed by a trained network 4 times and increase the speed of inference by a large amount. Reducing the number of bits of the weights allows for higher packing of data in SIMD registers. SIMD (Single Instruction Multiple Data) instructions are a common way of accelerating multimedia content on modern processors. Normal CPU registers typically only contain one value which instructions can operate on in sequence. SIMD registers in contrast can contain multiple data values which can be operated on in parallel and are therefore essential if neural networks should be run on a CPU. For example in the ARMv8 architecture where the SIMD extensions are referred to as NEON and the SIMD registers are 128bit wide, it is possible to pack $128/16 = 8$ 16bit numbers in contrast with only $128/32 = 4$ 32bit numbers [4]. Mathematical operations on values with fewer bits are in general faster and more energy efficient to perform on processors [6].

The compromise of quantization is accuracy, however measurements have shown that when comparing 32bit with 8bit neural networks, the extra precision gained by using 32bit floating point numbers does not significantly alter the total accuracy of the network and quantization can therefore be suited for lower powered devices [12].

1.4.5 False Alarm Filtering in the Cloud

In 2017 another thesis, with the same goal as this one, with the title *False Alarm Filtering within Camera Surveillance using an External Object Classification Service* was published. Lundholm et al. investigated the possibility of using the cloud to do alarm filtering of motion detection [11]. The filtering was done using AWS (Amazon Web Services) and the motion detection frames were sent to the cloud after being preprocessed. They found that their solution showed great promise in filtering out false alarms (filtering out 90% of the false alarms) while having a minimum impact on the amount of true alarms that passed the filtering. They also investigated which preprocessing techniques that could be used to minimize the amount to data that needed to be transferred to the cloud by trying such things as JPEG-compression and gray-scale images. The main difference between this thesis and the work done by Lundholm et al. is that this thesis focuses on running the object detection on premise instead of in the cloud and that this thesis compares several networks and backends.

1.5 Contributions

This thesis has had no particular division of labor and most design decisions have been made by both authors together but naturally some parts of the implementations were done by only one author. The following is the high level division of labor in no particular order. JS refers to Jonathan Strandberg and ER refers to Erik Rosengren.

Area	Contributor
Annotation application	JS
Annotate video	JS & ER
Recording (benchmark video)	JS & ER
Preprocessing	JS & ER
Detectors and networks (Open CV)	ER
Detectors and networks (Arm NN)	JS
Detectors and networks (Tengine)	JS
Benchmark application	JS
Metadata parsing	ER
RTSP Functions	ER
Postprocessing	ER
Error handling	JS
Graphs and results	JS & ER

Table 1.1: Contributions

Chapter 2
Approach

A common technique of doing object detection in live video is to take the entire video frame and feeding it through a deep neural network such as MobileNet. The output will then contain bounding boxes with labels for all detections. For most applications where simple classifications of images is needed this is sufficient but there are several problems to this approach which will be discussed in section 2.1. The approach in this thesis will be to send the frames through a pipeline of several steps in order to improve the accuracy and the performance over the naive approach mentioned above.

2.1 Challenges

There are several difficult sub-problems that will have to be overcome in this thesis in order to solve the problem statement mentioned in the introduction. Most of these problems are performance or accuracy related.

2.1.1 The Parked Car Problem

The parked car problem is a problem that occurs when sending an entire video frame into an object detector. Using the example of a parked car somewhere in the frame the object detector will detect the car even though it is not what triggered the alarm. This would obviously lead to unwanted notifications for the end user. A way to solve the "parked car"-problem would be using motion detection to classify only relevant parts of the current video frame. Using motion data to extract sub-frames from the original image has an additional benefit: the feed forward in the network would only be performed when something is moving in the frame. Feeding an image through a network is traditionally very computationally expensive and since a state-of-the-art graphics card to perform efficient convolutions is not available in the Axis recorder, this would save a significant amount of computing power.

Axis cameras already use motion detection to give bounding boxes for moving objects, the approach used in this thesis will be to use both in conjunction to eliminate false alarms given by using only motion detection and to effectively filter out stationary objects that would otherwise be classified as relevant by a deep convolution neural network. An example of the parked car problem can be seen in figure 2.1

Figure 2.1: An illustration of the parked car problem.

2.1.2 The Tree Problem

Another problem that occurs in some surveillance scenes occurs when objects are partially obstructed by other objects. For example: imagine a scene where cars drive on a road with adjacent trees. When the car is partially obstructed by the tree trunk most CNN based object detectors will either detect two cars or no car. This can have a large impact on the results and will have to be considered when evaluating object detection in these kinds of scenarios.

2.1.3 The Wrong Class Alarm Problem

When evaluating how well the solution performs there are several key things to measure, such things are described in more detailed in section 2.5, but one example is the number of false alarms that the solution produced. The problem with false alarms occurs with the definition of what a false alarm is. Imagine a scenario where classes that should trigger an alarm are cars and people and that a car triggers motion in the scene. The car is then sent to the detector which instead of correctly classifying it as a car it detects a person. Since both these classes are classes that should trigger an alarm a seemingly correct alarm is sent but for the wrong class. The question is whether this should be considered as a false alarm or as a true alarm specifically when evaluating the solution.

2.1.4 The stretching problem

Most CNN based object detection networks require the input image to have the same width and height. Since the sub-frames acquired from motion detection do not have this constraint

they will have to be transformed in some way. One naive way to do this is to simply stretch the image by repeating pixels causing the proportions of the image to change. This results in humans in the transformed sub-frames (which are usually taller than they are wide) to be almost unrecognizable which can affect the motion detection result. Another approach might instead be to pad the sides of the image with black pixels until the image height and width are the same. This will still require introducing new pixels or removing them when the image is resized to fit the input layer of the network but will keep the original proportions. An example of the stretching problem can be seen in figure 2.2.

Figure 2.2: An example of before and after stretching a person

2.1.5 The Truck vs. Car Problem

Some objects are easy to identify but hard to classify even for humans. For example: is the object in figure 2.3 a car or a truck? What about the object in figure 2.4, is that a truck or is it a van? Lastly in figure, 2.5 is that another truck? All three of these images are taken directly from the COCO-dataset and they are all classified as trucks. The car vs. truck problem is a result of both incorrect classification in the COCO-dataset (see section 2.4.6) and of a very broad class definition. The latter is most likely the cause of the former due to the COCO-dataset being annotated by a lot of different people, all with slightly different opinions on exactly what a truck is. The problem this causes in this thesis is that when something like a pickup-truck or mini-van triggers motion detection it is just as likely that it gets classified as a car as it is that it gets classified as a truck. This will of course affect the results, leading to the network looking worse than it actually is for classifying a pickup-truck as a car, especially if cars are chosen as an alarm class.

2.1.6 Night Scenes

As many crimes such as breaking and entering or vandalism are easier for criminals to perform at night it is vital that surveillance systems are as operational at night as they are in the

Figure 2.3: A truck?

Figure 2.4: Also a truck?

Figure 2.5: Another truck?

day. The same goes for the alarm push-notifications described in the introduction and the filtering of these alarms should work as well at night as it does in the day. However as soon as the sun sets lighting conditions change drastically and the objects that would have been easy to detect in daylight can become almost impossible for even humans to detect at night. To handle this problem Axis cameras can use IR filter and in some cases even use IR illumination to see in completely dark scenes, albeit in grey-scale and not in color. The problem lies in how well the object detectors perform in darker and IR illuminated scenes and whether networks trained on separate datasets need to be used in order to obtain good results in night scenes. Figure 2.8 is an example of a frame from a night scene.

2.2 Method

In order to solve the problem described in the problem statement some way to evaluate the solution is needed. The evaluation data also needs to be the same in order for comparisons to have any meaning. In the following section the method used for evaluation will be described.

2.2.1 Annotated Data

To evaluate the accuracy and speed of the networks, a common dataset for different scenarios that is applicable for an alarm system is used. To create the dataset video and bounding boxes were recorded with no annotation. To minimize manual labour, an initial run with the network ResNet50 was performed which annotated the metadata. This step produces a file where each line corresponds to a single metadata-event with a list of sets of detection classes for each bounding box. In total more than 8000 bounding boxes were annotated with this approach. Following is an example of how the annotated data is structured.

[{*car*}]	▹ Each row corresponds to a video frame
[{*person*, *bicycle*}]	▹ Each set contains the detected classes for a bounding box
[{*car*, *bicycle*}, {*car*}]	▹ The sets are in order as they were recorded

After the data was recorded, it was manually corrected to ensure consistency and to correct any misclassifications that were found. This process was repeated for different lighting conditions and times of day. The manually corrected data establishes a ground truth that can be used to compare the output from different networks. The use of sets containing detected classes enables the comparison of inclusion, intersection and superfluous detections and therefore provide relevant measurements between networks and using plain motion events.

2.2.2 Benchmarking

The subjects for the benchmark videos were chosen to be intentionally complex and difficult in order to avoid situations were networks perform with perfect or near perfect scores. The videos, their subject and relevant conditions and attributes are the ones used for benchmarking and evaluating. Table 2.1 contains all the relevant video information and figures 2.6, 2.7 and 2.8 are frames from some of the videos. Do note that there are two videos for every one mentioned in table 2.1, one for VMD motion data (see section B.2.6) and one for VCS motion data (see section B.2.7).

Video name	Weather	Time of day	Subject	In/Outdoor	Attributes
sun-road	Sunny	Noon	Road	Outdoor	sharp moving shadows, good lighting, strong reflections
overcast-road	Overcast	Noon	Road	Outdoor	very soft shadows, moderate lighting, weak reflections
cloudy-road	Cloudy	Noon	Road	Outdoor	varying lighting & shadows, varying reflections
sun-afternoon-road	Sunny	Afternoon	Road	Outdoor	darker and lighter areas, sharp moving shadows, reflections
night-road	Clear	Night	Road	Outdoor	street light shadows, backlight, dirty window
night-road2	Clear	Night	Road	Outdoor	street light shadows, backlight, dirty window

Table 2.1: Benchmark video names and attributes.

Figure 2.6: A frame from the cloudy-road scene

2.3 Motion Detection Metadata

Metadata in this thesis is defined as the data received from the cameras that does not include the actual frames from the video. The only source of metadata used of this type are the motion-detection bounding boxes received via VMD or VCS (see sections B.2.6 and B.2.7). Both of these send information at around 5 FPS. The metadata received from these two sources provide the largest optimization in the entire thesis. This is because without bounding boxes the entire frame at the time of a motion detection would have to be sent through the detector even if only a small object in the frame actually moved. This not only saves a

Figure 2.7: A frame from the sun-afternoon-road scene

Figure 2.8: A frame from the night-road2 scene

huge amount of time since feeding an image through a CNN scales with the size of the image but it also helps with handling the parked car problem mentioned in section 2.1.1.

2.4 Pre-trained Neural Networks

There are several variations of networks out there that are capable of doing accurate object detection. Some networks focus on raw accuracy at the cost of performance while others try to balance the two. Since this thesis revolves around doing accurate object detection on limited hardware the networks that will be benchmarked against each other will mostly be of the latter kind. The networks that have been selected for comparison in this thesis were chosen because of their popularity and their advertised effectiveness in conditions similar to the conditions of the problem statement in this thesis.

2.4.1 MobileNet

MobileNet exists in multiple versions and with each new version greater performance and accuracy is advertised. Some changes between MobileNet versions include switching activation functions from ReLU to swish and hard-swish for example [8]. This thesis will use all versions available which at the time of writing which are MobileNetV1, MobileNetV2 and large MobileNetV3. Full size versions of MobileNet was chosen since it was developed for mobile embedded systems which is a very similar use case to the one described in this thesis [9].

2.4.2 YOLO

You Only Look Once or YOLO for short, is another network architecture like MobileNet that similarly also exists in several versions of incremental improvements. This thesis will only look at the two latest versions of YOLO which are YOLOv3 and YOLOv3-tiny. The former being developed with a focus on accuracy while the latter is a much smaller version with much faster feed forward time but at a very large loss in accuracy [13].

2.4.3 InceptionNet

InceptionNet is a slightly older network architecture than both YOLO and MobileNet. When InceptionNet was released in 2014 it broke records in the ImageNet Large-Scale Visual Recognition Challenge because of its new design allowing for deeper and wider networks while keeping the computational requirements constant [15]. Several versions of InceptionNet exists but in this thesis InceptionNetV2 will be used which is a slightly improved version of the original. It is worth noting that this is another network focused more on accuracy than on speed.

2.4.4 ResNet

ResNet which is short for Residual Network is another network architecture which relies on residuals, also called skip-layers, to train deeper networks in a more efficient and accurate manner [7]. There are many networks that use residual layers but this thesis will use ResNet-50 which is a residual network with 50 convolutional layers. ResNet-50 is not a network built for performance but rather it focuses on accuracy and was chosen in order to compare a very accurate network with much faster slightly less accurate networks.

2.4.5 Faster R-CNN

Faster R-CNN is the third iteration of the R-CNN region proposal network preceded by R-CNN and Fast R-CNN and is, not surprisingly, faster than the two previous iterations. Faster R-CNN and its predecessors aims to solve the difficult problem of region proposal in object detection [14]. Region proposal is the initial step in object detection when the detector finds the interesting regions of the input image where objects may be. It is a hard problem for CNNs to solve since there are no predetermined number of objects that can occur within

a given image meaning that the region proposing network must have outputs that can vary in length. The networks that use Faster R-CNN in this thesis will be InceptionNetV2 and ResNet50.

2.4.6 COCO Dataset

The Common Objects in Context dataset is a dataset created by Microsoft containing images scraped from the image service Flickr. These images are taken with many different types of cameras which means their quality is very varied. The images also have very varying subject matter and lighting conditions which makes for a very good diverse dataset for use in training object detection networks [10]. The dataset contains 80 classes varying from exotic animals such as elephants to utensils such as forks. The COCO dataset has been annotated by many different people and at the time of writing this contains over 200 000 labeled images. All of the networks compared in this thesis will be trained on the COCO dataset simply because it is a widely used dataset with relatively few classes compared to the likes of ImageNet which contains 1000 classes. Figures 2.9, 2.10 and 2.11 are some examples of images in the COCO dataset.

Figure 2.9: COCO example 1

Figure 2.10: COCO example 2

Figure 2.11: COCO example 3

2.5 Theory

2.5.1 Definitions

In order to compare different networks, preprocessing methods and lighting conditions relevant measurements has to be defined. These metrics have to show comparable values of performance and accuracy in such a way that it is easy to show which network works best in certain conditions.

Definition 1. True alarms are defined as

$$|Ground\ Truth \cap Network\ Output \cap Alarm\ Classes|$$

It is the number of correctly classified classes that are in a given set of Alarm Classes for a bounding box.

Definition 2. False alarms are defined as

$$|(Alarm\ Classes \cap Network\ Output) \setminus Ground\ Truth|$$

It is the number of detections by the network that were incorrectly classified and is in a given set of alarm classes.

Definition 3. Skipped frames are defined as the number of frames skipped because the network did not have enough time to start detecting the objects in the frame.

With these we can continue to define the measurements we will use to compare networks.

Definition 4. Mean true alarms is defined as

$$\frac{\sum_{i=1}^{I} \theta(True\ alarm_i)}{I}$$

where I is the total number of bounding boxes which were annotated as an alarm class. θ is the Heaviside step function.

This definition gives us that $mean\ true\ alarms \in [0, 1]$, where 0 represents no true alarms and 1 is that all alarms were correctly classified.

Definition 5. Mean false alarms is defined as

$$\frac{\sum_{n=1}^{N} \theta(False\ alarm_n)}{N}$$

where N is the total number of bounding boxes. θ is the Heaviside step function.

This definition gives us that $mean\ false\ alarms \in [0, 1]$, where 0 represents no false alarms were sent and 1 is that all bounding boxes produced a false alarm.

Definition 6. Mean skipped frames is defined as

$$\frac{Skippedframes}{F}$$

where F is the total number of frames.

This definition gives us that $mean\ skipped\ frames \in [0, 1]$, where 0 represents no skipped frames were sent and 1 is that no frames were processed.

2.6 Implementation

Since the executable is to be run on the recorder C++ was chosen as the programming language for its object-oriented design and ability for low level memory and resource management. All development is done on x86 but is then cross compiled to aarch64 using GCC. Aarch64 is the 64bit instruction set used on the recorder. The overall design that is used is a pipeline where the frames received are processed in multiple steps. Each significant step is processed on a separate thread of execution to minimize latency for steps that can be done in parallel. The following sections will illustrate each step and figure 2.12 illustrates the entire pipeline.

2.6.1 Preprocessing

When any motion event is sent by a camera the corresponding frame is grabbed by the pre-processor. Each bounding box that is received in the motion event is first re-scaled by a

configurable factor to ensure the object is captured in the re-scaled box and is then cut out from the the original frame and put on a worklist for detection to be performed. These are called subframes and, depending on the size of the object setting off the motion detection, are significantly smaller than an entire frame. An uncompressed 1920x1080 video image with three 8bit channels requires a minimum of $1920 * 1080 * 8 * 8 * 8 \approx 6Mb$ of data which for an embedded system is a considerable amount and can therefore introduce latency and thus this work is put on a separate thread on a separate core. The subframes can be either scaled or padded into a square with padding being the hypothetically best since they preserve the dimensions of the detected object. Figure 2.13 illustrates the input and output of the preprocessor with an example.

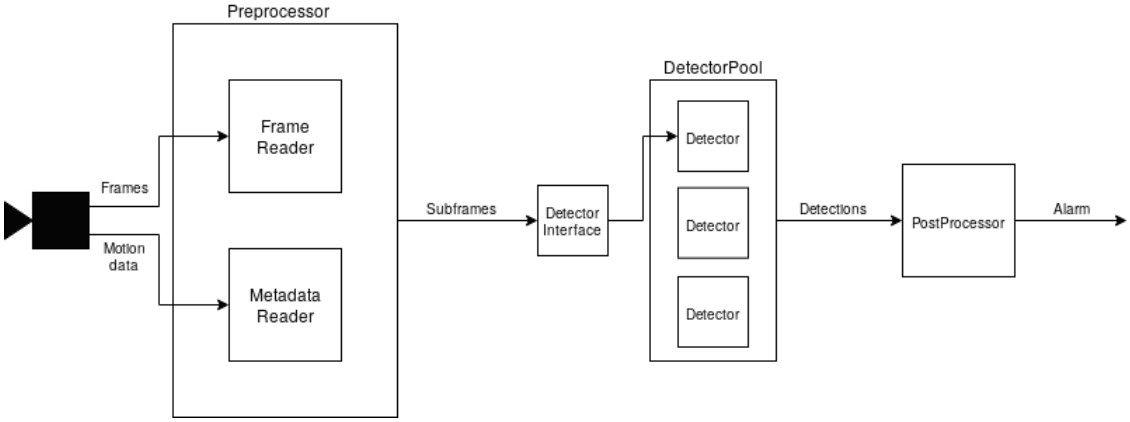


Figure 2.12: The complete pipeline

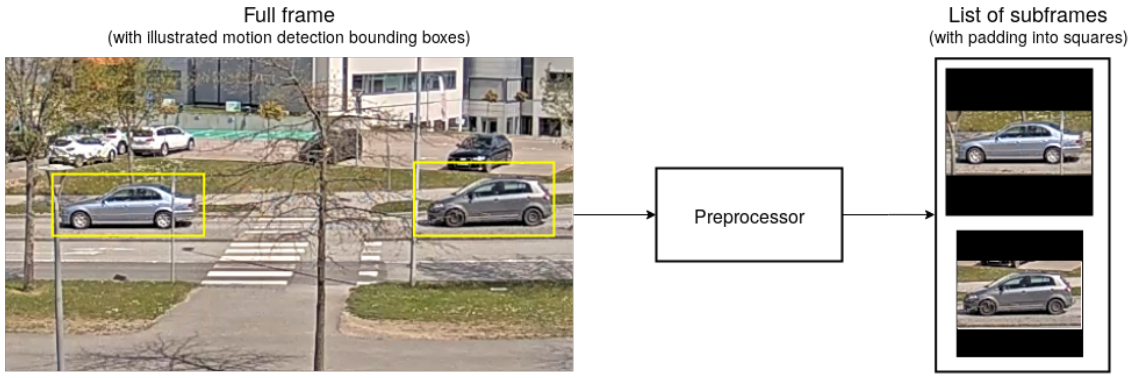


Figure 2.13: Preprocessor input and output

2.6.2 Object Detection

The result from the preprocessor is taken off the worklist and then sent through a common detection interface which accepts a list of subframes and returns the detection labels with their corresponding confidences. This step is always the bottleneck for the application since feeding the image through a neural network is very computationally expensive.

If a new frame with bounding boxes arrive to the detector before the previous frame has completed processing, the previous frame will be skipped. This choice was made to ensure

that the detector always has the latest frame possible and gives a more fair comparison between videos with different amount of bounding boxes in each frame. This does of course mean that the object detector can miss alarms that should take place in the skipped frames but it also stops the object detector from building up a large delay that would occur if each frame is placed in a buffer. If on the other hand the object detection were to run on recorded video instead of live then the dropping of frames would not be acceptable since it is no longer necessary to send the alarms in real time.

The DNN module in OpenCV divides the work between all cores available for inference, however if Tengine (see section B.2.4) is used as the back-end the computation will be performed on one core. This opens up the ability to have a pool of detectors which henceforth will be referred to as the **detector pool**, allowing multiple detections to take place at the same time and therefore reducing the total detection time for a frame with multiple bounding boxes. ComputeLibrary used by Arm NN will automatically detect the number of highest performing cores with the same type, meaning that for the hardware used in this thesis it will use one thread since there is one ARM Cortex-A72. Unfortunately even though Arm NN only runs on one thread, the library itself does not allow for multiple networks to run in parallel. Figure 2.14 illustrates the interfacing with the detector pool.

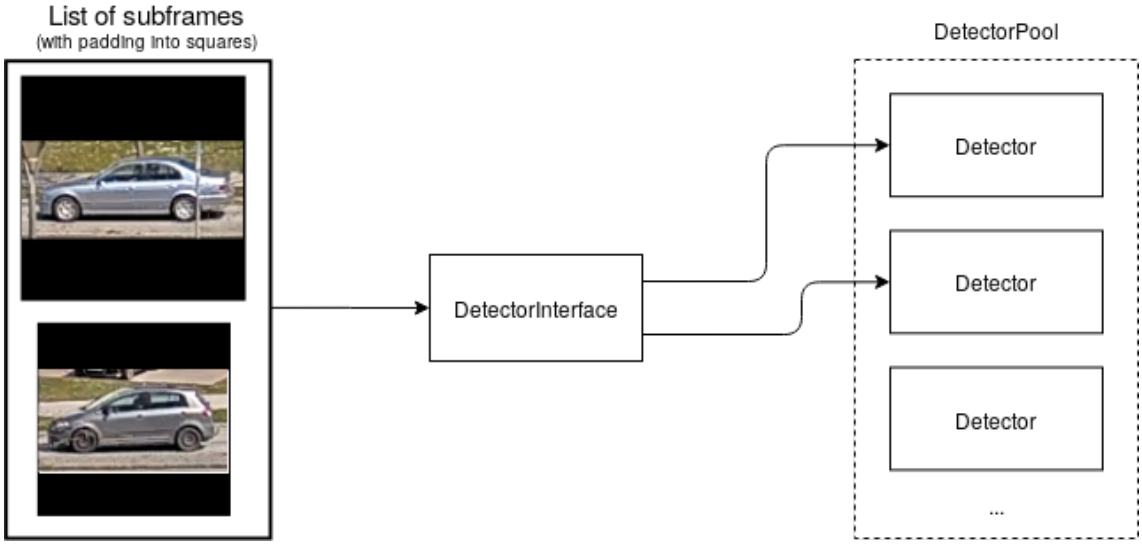


Figure 2.14: Detector interfacing

2.6.3 Postprocessing

After the subframes has been sent through the detector they are sent to the postprocessor. The postprocessor checks all the subframes with their detections and looks for classes that should send an alarm. Any class can be set as an alarm class and if an alarm class is detected above a configurable threshold. The postprocessor will also draw debug data into the frame if the GUI is enabled. The debug data in this case is the bounding box for the motion detection, the re-scaled bounding box, the names of the classes that were detected in the box as well as the detection confidences.

2.6.4 CPU Affinity

CPU affinity is a property of processes or kernel threads that allows it to be scheduled on a specific set of CPU cores. Specifically on Linux this can be programmatically controlled with the pthread library. This can allow programmers to heavily optimize a multithreaded application for a multicore system, for example reducing expensive core migrations performed by the operating system. Using the CPU affinity mask usually adds unnecessary complexity to applications since the number and type of cores is different between between computers which makes it hard to predict how it should be scheduled for optimal performance. However in this project, the hardware is known beforehand and can therefore be exploited.

The CPU cores in the hardware has as mentioned in appendix B.1, not all the same core type. This has the implication that scheduling of all threads is important depending on the type of workload. It becomes especially important on the thread that is responsible for object detection since it is the most computationally expensive. Therefore the backing thread pool of the detector pool will internally always schedule work on the Cortex A72 if it is not busy, otherwise it will use the other cores configured to be available for the pool. The other threads used for the applications will have their affinity mask set to share the remaining cores in the system.

Chapter 3
Evaluation

In order to be able to determine which networks work best and what other parameters to use some evaluation process is needed. The goal of the evaluation process will be to create a scenario as close to the reality of the problem statement as possible while being reproducible every run. That means not testing on live video but rather using the recorded data annotated by the authors as a ground truth for use in all comparisons. Details about this data can be found in section 2.2.

3.1 Experimental Setup

To evaluate each network, all networks are run on the pre-recorded benchmark videos. If the object detection for all bounding boxes in a frame would take longer than the time for a new frame to arrive in the queue, it will skip all new incoming frames until the current frame is processed. All skipped frames are marked as such to facilitate measurements of performance in relation to other networks. The output format is structured exactly the same as the pre-annotated data to allow for comparison. In total more than 8000 bounding boxes are annotated and used during benchmarking.

For each network and video, two separate runs are performed to test the different methods for resizing the image contained in a bounding box. The first method is to stretch the image to the correct height and width accepted by the network, thereby changing the proportions of everything contained. The second method is to put zero valued pixels to fill out the image to fit in the network input. This keeps the proportions of the image but introduces pixels that were not there before. Testing both of these methods is to determine which results better detection results and to answer the stretching problem defined in Section 2.1.4.

Scaling down input image to a network will decrease the amount of information available for a neural network, however, by the quadratic time complexity of performing convolutions on an image it will significantly change the amount of time required to perform object detection. By varying the input size for each network, a good balance between performance and

accuracy can be found.

3.2 Results

The following are the results from the benchmarking and the configuration that was used to generate them. The figures in this section are the most relevant figures and tables but only displays the mean performance, results for all the individual videos can be found in appendix A. The discussion of the results can be found in section 3.3. It is important to note that all networks except *mobilenetv1cocoarmnn* are running on Tengine as the backend unless mentioned otherwise.

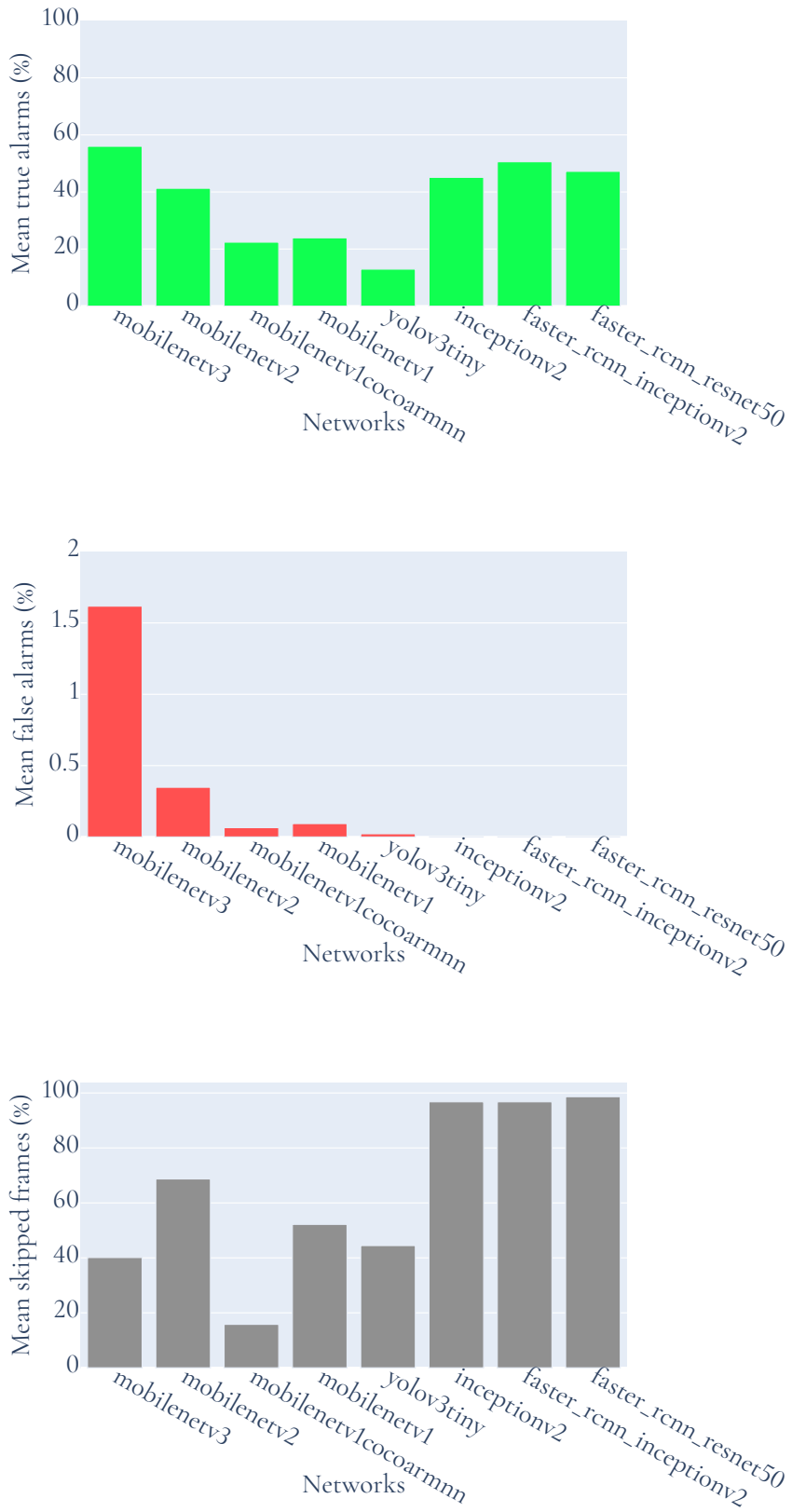


Figure 3.1: Mean performance and accuracy for all networks over all videos with frame stretching.

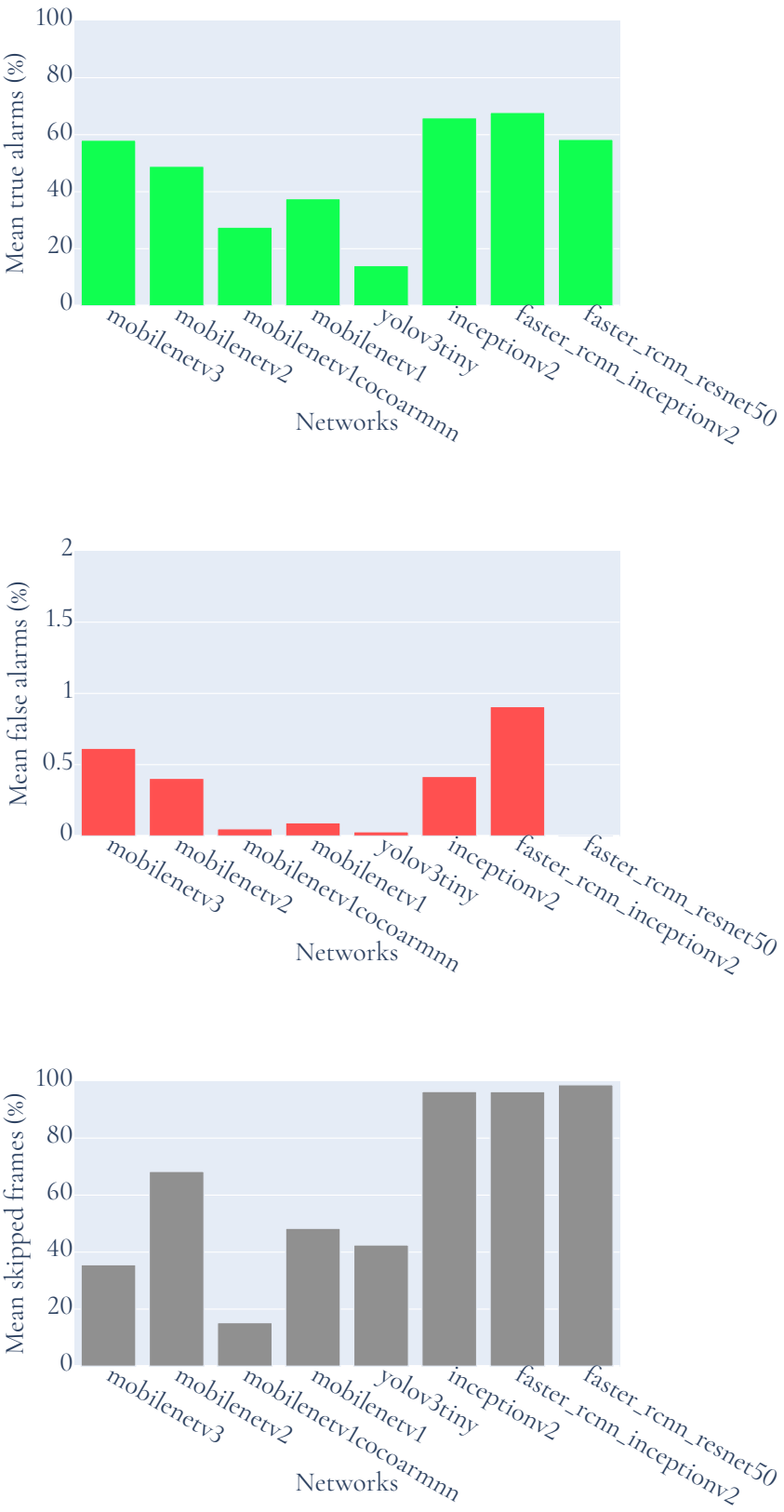


Figure 3.2: Mean performance and accuracy for all networks over all videos with frame padding.

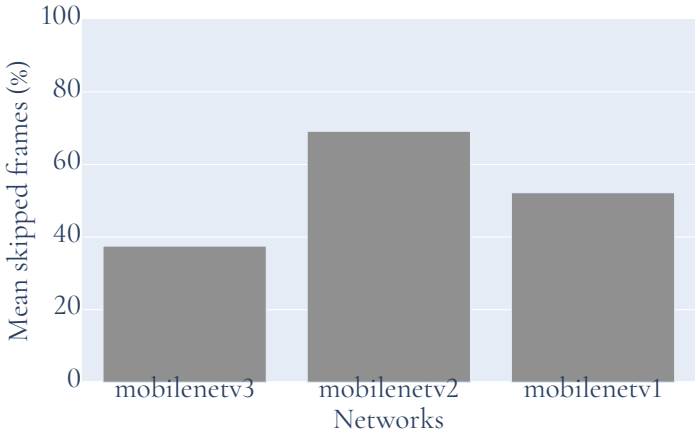


Figure 3.3: Mean skipped frames with detector pool

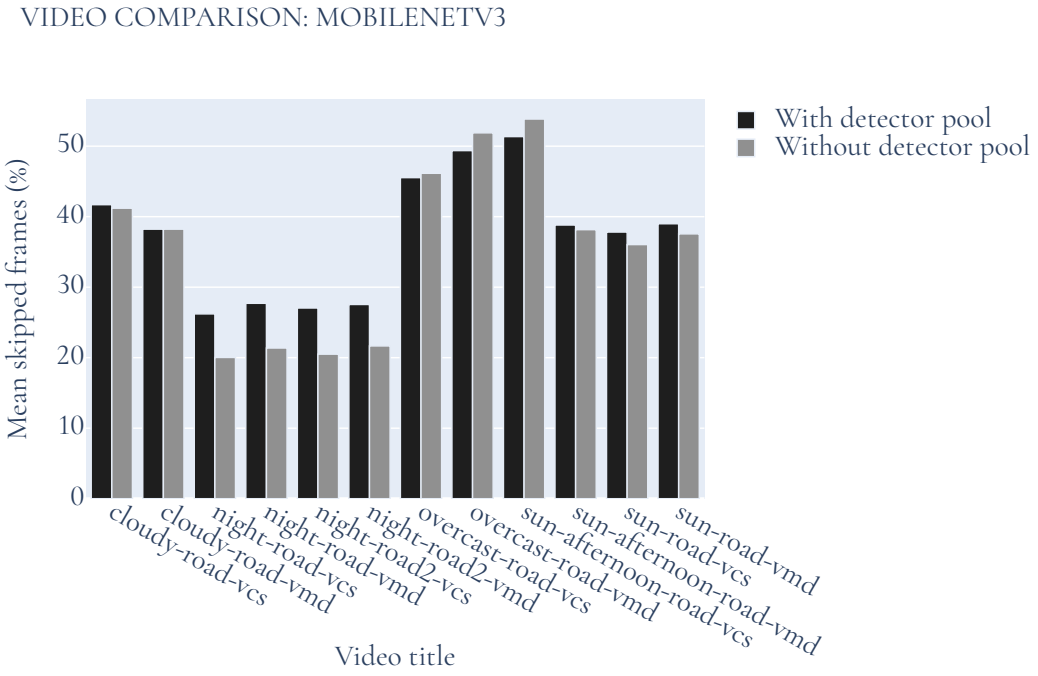


Figure 3.4: Mean skipped frames for MobileNetV3 with detector pool.

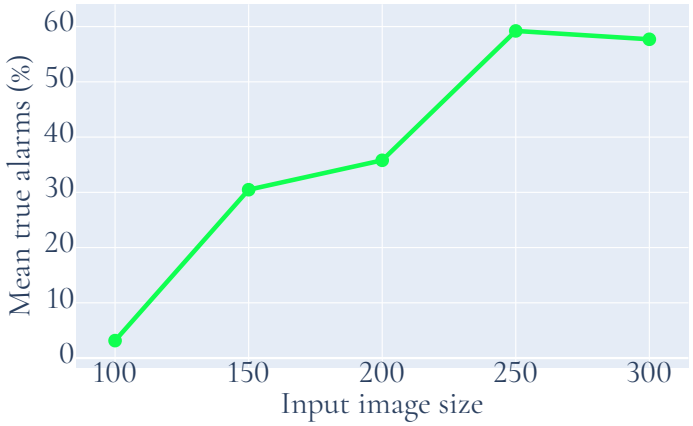


Figure 3.5: Mean true alarms vs the size of the input image. The x-axis represents the side of the square subframe.

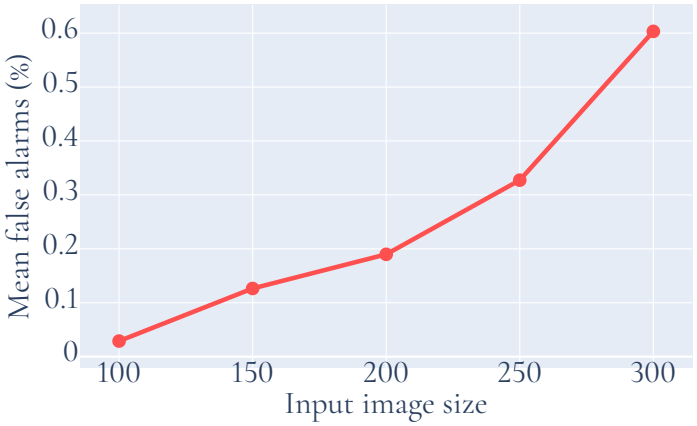


Figure 3.6: Mean false alarms vs the size of the input image for MobileNetV3. The x-axis represents the side of the square subframe.

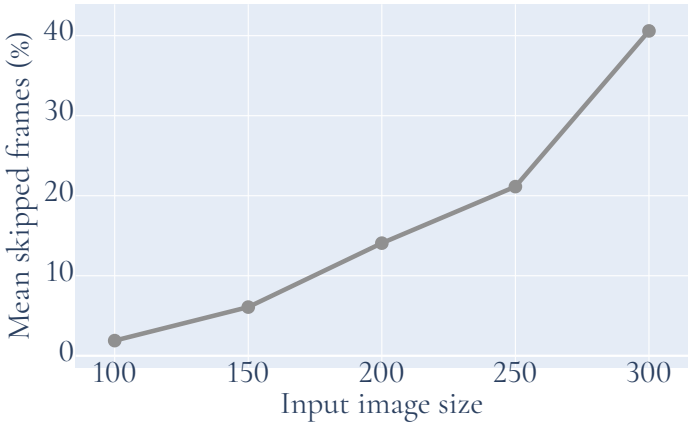


Figure 3.7: Mean skipped frames vs the size of the input image for MobileNetV3. The x-axis represents the side of the square subframe.

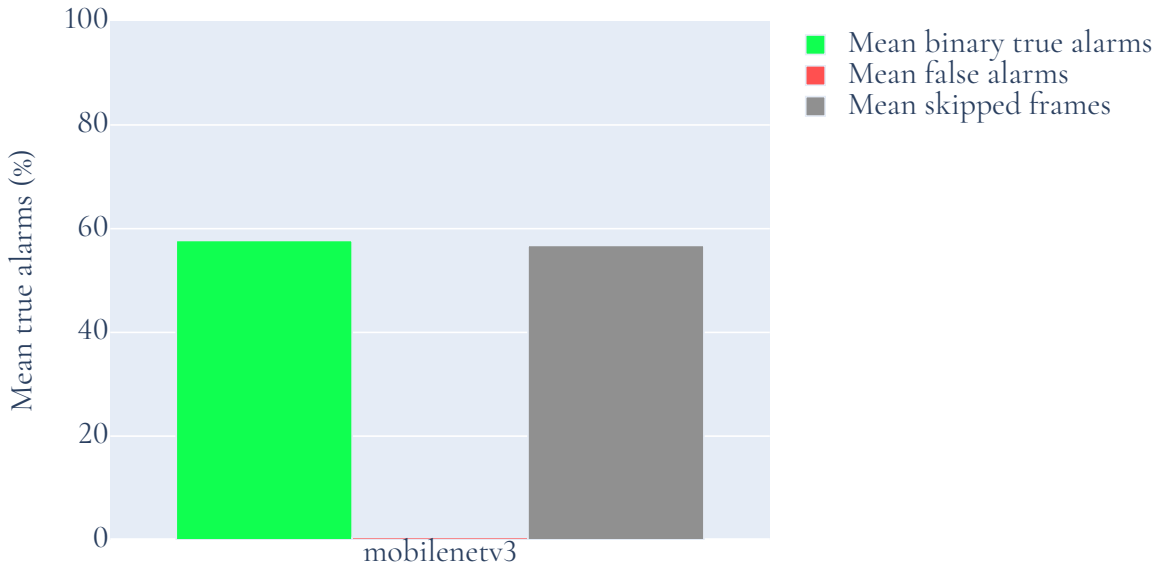


Figure 3.8: Mean performance for MobileNetV3 with 250x250 image size and frame padding with the OpenCV DNN module.

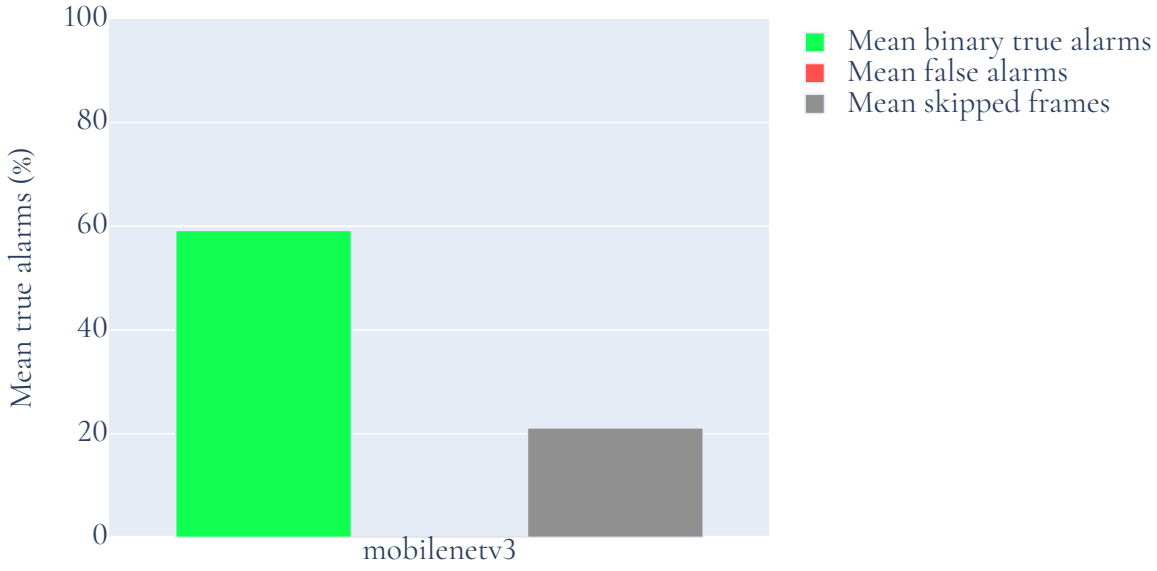


Figure 3.9: Mean performance for MobileNetV3 with 250x250 image size and frame padding using tengine

3.3 Discussion

With the results in hand it is time to do an analysis of which networks perform best. To achieve this the networks that are found to be nonviable for whatever reason will be removed from the comparison and subsequent analysis will focus on the still viable networks. Comparisons of the efficiency of the different optimizations will be done with the viable networks only since having too many networks to compare will become overwhelming, especially when there are several different optimizations.

3.3.1 Metrics

In section 2.5 the different metrics measured were introduced, these were: *true alarms*, *false alarms* and *skipped frames*. Because the purpose of this thesis as described in the introduction it is very important that a network does not generate many *false alarms*. However a problem arises when using only *false alarms* as a metric: the best way to not generate any *false alarms* is to not generate any alarms at all, hence the need for the *true alarms* metric. It is therefore necessary to take into account both the *false alarms* and the *true alarms* when comparing networks by their accuracy. The *skipped frames* metric is the measurement of how fast the network is, the fewer skipped frames, the faster the network is. These three will be the primary metrics used when evaluating the viability of a network.

3.3.2 Mean True Alarm Percentage

It is very important to note that even though the mean true alarms percentage might look low (with the best network scoring slightly above 60%) that these results are actually very good. The percentage of mean true and false alarms can be interpreted as the probability that the network will correctly or incorrectly classify an object. Since objects normally stay in the frame for several seconds and they only need to be detected as an alarm class **once** for the application to send an alarm this probability does not need to be especially high in order for the filtering to be efficient. For example: a detector with 30% mean true alarm percentage and an alarm object that stays in frame for 10 motion detection events (a very short time at around 2 seconds) the probability that this object **doesn't** get detected as an alarm is $(1 - 0.3)^{10} = 0.02825$ or 3%. For a detector with 50% mean true alarm percentage this becomes $(1 - 0.5)^{10} = 0.001$ or 0.1%. Of course objects that say longer in frame than 2 seconds, which they generally do, have even lower probabilities to not get detected. The inverse is of course true for false alarms, the longer an object stays in frame the larger the probability that it gets labeled as an alarm class even though it is not and thus generates a false alarm.

3.3.3 Nonviable Networks

As can be seen from the results some networks are simply nonviable to use in a scenario such as the one described in in the problem statement. Networks can be nonviable for several reasons: Having a large feed forward time and thus resulting in too many skipped frames or being too inaccurate thus missing too many alarms or generating too many false alarms. It is also possible that some networks are nonviable because of a combination of all the above. By first examining the result and removing the networks that fail on the criteria above it will be easier to do an in-depth comparison of the viable networks that remain.

As the slowest network it is comes as no surprise that the first network to be labeled as nonviable is ResNet50. Even though ResNet50 has great accuracy for almost every detection that it had time to make it skips way too many frames, almost every frame in every video in fact, to be used in any real-time capacity. The network is however far from unusable, with its great accuracy of 50% mean true alarms, as can be seen in figure 3.1, it is very good for annotating recorded video as described in section 2.2.1. ResNet50 is one of the networks that have 0% mean false alarms, but this does not change the fact that it gets labeled as nonviable due to its poor performance.

With mean skipped frames reaching above 0.95 (see figure 3.1) for both the InceptionV2 networks (with and without FasterRCNN) it is obvious that these networks are nonviable for real time detection. Both networks do however have extremely few false alarms (around 0%) in the benchmark set of videos which speaks for their accuracy. They both also perform at pretty average accuracy in the *true alarms* metric at 45 and 50% respectively meaning that they are not terrible networks, just not fast enough. This does not come as a surprise since InceptionV2 is one of the older architectures in this thesis and focuses more on accuracy than on performance as mentioned in section 2.4.3. An interesting observation that can be made from the data is that the FasterRCNN version of the network is slightly faster and almost 15% better in terms of accuracy than the one without FasterRCNN indicating FasterRCNN actually does improve performance and accuracy as described in section 2.4.5.

The next network to be eliminated is YOLOv3 and it is eliminated for the same reason that it does not appear in the data, because it is by far the largest network in terms of memory and is sadly not usable on the recorder since it takes up almost all available memory. This makes the recorder unable to perform its other tasks such as recording video and streaming it to AXIS Companion. If the recorder had more memory than the current 1 GB, YOLOv3 would not have been usable in any case since it is too slow to be run in real time but its performance is rather good for the motion detections that it actually gets to detect. This makes it useful in similar offline scenarios as ResNet50.

The tiny version of YOLOv3tiny is the last network to be eliminated due to its low accuracy. The network has average speed but unfortunately it under-performs when it comes to the *true alarms* metric where it misses a majority of the alarms scoring below 15% mean true alarms (see figure 3.1). If the network had much better performance this lack in accuracy might be forgiven since it would be able to run more detections but since it skips half of the motion events due to its speed it too gets labeled as nonviable and is removed from further comparison.

3.4 The Remaining Networks

The remaining networks are all MobileNet versions and this is no coincidence. As mentioned in both sections 1.4.3 and 2.4.1 MobileNet was developed to be run in just the kind of scenarios as the one in this thesis. Networks developed to be used on mobile devices will obviously outperform networks made to push the limit of accuracy in dataset competitions while being run on the most expensive GPU:s available. By looking at the results however there are some seemingly odd ordering of the remaining MobileNet networks that don't fit the *"newer versions are incremental improvements of the previous versions"* narrative. For example MobileNetV1 outperforms MobileNetV2 in performance but not accuracy while it would be expected that v2 would outperform v1 by a noticeable margin in both categories. Moving forward all comparisons will be made between the remaining networks as the nonviable networks are no longer interesting to compare.

3.4.1 MobileNetV1 vs. MobileNetV2

By looking at figure 3.1 and comparing MobileNetV1 and V2 it is immediately apparent that the MobileNetV1 outperforms MobileNetV2 in terms of performance. MobileNetV1 skips less frames while generating close to the same amount of false alarms but also generates less true alarms. This is odd because MobileNetV2 claims to have both better accuracy and performance than its predecessor which was described in section 2.4.1. It is possible that the reason for this is due to the backend. Backends have varying quality implementations when it comes to different layers and activation functions. There are multiple types of layers possible in a convolutional network and obviously different networks use different kinds of layers. The type of network a backend has been tested with can impact which layers have had more time to mature in regards to performance. This can pose a problem when comparing different neural networks on a single backend, for the simple reason that near optimal performance of layers and activation functions may exist for one network but not others. So the statement "network x is faster than y" may hold in theory when describing operations like the amount

of multiplications performed but can quickly collapse when testing out on a specific backend where the performance of layers can vary greatly.

3.4.2 MobileNetV1 vs. MobileNetV1 Arm NN

Looking closer at MobileNetV1 using OpenCV and MobileNetV1 using Arm NN reveals, unsurprisingly, that Arm NN yields much greater performance at the cost of accuracy. This is because the Arm NN version of MobileNetV1 is quantized while the OpenCV version is not. Arm NN also has hand-written NEON instructions which the OpenCV backend does not. In theory quantization should reduce the accuracy and it does but not by much as can be seen in figure 3.1. This is true for all videos and night scenes, as seen in figure A.4 and figure A.5, seem to affect both networks the same amount, the quantized version is slightly less accurate in all videos but it makes up for it in speed.

While both versions of MobileNetV1 are trained on the COCO dataset, the subframes used when doing detection are different than the images trained on, in the sense that they are all in the range of 100x100 pixels in size. This requires most subframes to be upscaled, therefore introducing information that was not there from the start. Non-quantized networks may be less sensitive to this added noise and this may therefore help to explain why there was a drop in quality when using Arm NN.

3.4.3 MobiletNetV2 vs. MobileNetV3

As expected MobileNetV3 outperforms MobileNetV2 by a wide margin both when it comes to performance and accuracy. Examining figure 3.1 reveals that MobileNetV3 has almost double the performance of V2 while having more than 25% higher accuracy. MobileNetV3 does however produce more false alarms (about 5 times more in fact) than its predecessor but this is because of the lower confidence threshold needed with MobileNetV3. While MobileNetV2 uses 0.8 as confidence threshold MobileNetV3 uses 0.7 because MobileNetV3 rarely produces confidences above 0.8 and thus needs a lower confidence. The results are therefore not directly comparable but using 0.7 as confidence threshold for MobileNetV2 results in way too many false alarms therefore the decision was made to compare the networks at their best performance, not identical parameters. In the end, despite the higher amount of false alarms, MobileNetV3 is the better network and will henceforth be used as the primary network to compare the effectiveness of the different optimizations.

3.5 Optimizations

As can be seen in the results there are also data that can be used to determine the improvements of different optimizations. Some optimizations are meant to increase the performance while others are meant to increase the accuracy.

3.5.1 Stretch vs. Pad

The padding optimization focuses on increasing the accuracy of the networks by preserving the scale of the objects. This is done by adding black bars to the image to make it square

instead of stretching it which is done when resizing. Intuitively stretching should result in worse accuracy since, for example, a person looks nothing like a person when they are as wide as they are tall. However it isn't quite that easy to determine which should yield the best accuracy. Looking at the figures 3.1 and 3.2 it is apparent that there is a general increase in *mean true alarms* across **all** networks. The increase in accuracy varies from network to network with MobileNetV1, MobileNetV1 Arm NN and MobileNetV2 improving the most of the viable networks while MobileNetV3 the least. It is worth noting that the nonviable networks InceptionV2, FasterRCNN InceptionV2 and FasterRCNN ResNet50 had a major increases in accuracy but are still too slow to be used. This does however speak for the validity of this optimization. As expected the performance is mostly unaffected since this optimization does nothing to reduce the image size and this has no effect on the feed forward time of the network.

Since MobileNetV3 is the best network it is interesting to take a closer look at its result with and without padding. The increase in true alarms is not very large as mentioned above but the decrease in false alarms on the other hand was quite dramatic. The false alarms were reduced by about **50%** which is a great improvement since MobileNetV3 was the network generating the most false alarms due to its lower confidence threshold. There is also a decrease in the skipped frames but this is most likely unrelated to the padding optimization.

3.5.2 Changing the Input Size of the Image

As mentioned in Section 3.1, varying the input size of a subframe should have an impact on the speed of the detection, this result is clearly visible in Figure 3.7. Decreasing the size from 300 to 250 decreases the number of skipped frames by more than half, while also slightly improving the accuracy for the alarm classes as can be seen in Figure 3.5. Decreasing the input size further decreases the number of frames that are skipped but at the cost of accuracy. Another thing to take notice of is the decrease in false alarms seen in Figure 3.6.

This relationship that is shown in Figures 3.6, 3.5 is very interesting and counter intuitive since the model is trained on image sizes of 300x300 which seems it should yield the best results. Although it should be noted the the way the networks are used here is quite different from their normal use case where a large image gets downscaled which looses information, while the method used here normally upscales the images from the bounding boxes to fit the network and therefore adds information. This difference between the training and data may help to explain why there is a slight increase in accuracy when going from 300x300 to 250x250 in input image size.

3.5.3 Detector Pool

The detector pool optimization was described in section 2.6.2 and should in theory improve performance for the faster networks when there are several objects triggering motion detection at the same time. It is worth noting that this optimization will have little effect on the benchmark videos recorded at night since they mostly contain single objects moving.

For hardware where all the CPU cores share the same type, this optimization should in theory have a significant improvement. A problem arises when using the hardware used in this thesis since this does not hold. For example, disregarding the overhead of synchronization and assuming only two bounding boxes, the time it takes for the Cortex-A53 to complete

has to be faster than $TimeForDetectionOnA72 * 2$. This fact arises for the simple reason that $total\ detection\ time = max(detection\ time\ A72, detection\ time\ A53)$ if running in parallel. As such, if the time to run two detections on the A72 in sequence is faster than dividing the work on the A72 and A53 to work in parallel, this optimization will slow down the system and will therefore not be worth it. In general the constraint relaxes as the number of bounding boxes increases in a frame, assuming no synchronization overhead and N bounding boxes where 1 thread runs on the A72 and $N - 1$ threads are divided on the other cores, the time requirement for detections an A53 to make the detector pool optimization profitable in terms of performance becomes $TimeForDetectionOnA72 * N$.

From the results shown in Figure 3.3 and 3.2 there is no significant difference in the number of skipped frames, suggesting that the speed of the A53 cores is not worth it. While the measured time for detection on A53 fits with the theoretical limit defined, other hardware effects can affect multicore performance. The most obvious is the size of the weights which on disk is $13Mb$. Having multiple instances of the network and simultaneously having images sharing the same CPU cache will inevitably result in a large amount of expensive cache misses. The results in Figure 3.3 does not show an improvement using a detector pool, comparing the bars between Figure 3.4 shows that for videos with a lot skipped frames, i.e. a lot of bounding boxes, there is a small improvement in regards to skipped frames. Although a slight decrease for videos with fewer bounding boxes.

Scaling is of course still bounded by the number of physical cores on the hardware and allowing all of the cores to be consumed for this single application is not an option if other real-time applications are supposed to share the system resources.

Chapter 4
Conclusion

With the results analyzed it is now possible to determine which network was the best and which optimizations to use with that network in order to make it even more accurate and give it better performance. Although this thesis and the thesis by Lundholm et al. [11] do not use the same kind of metrics which makes it hard to compare the results directly, it is still interesting to compare the conclusions. The thesis from Lundholm et al. [11] found that using neural network powered object detection to filter out false alarms looked promising. The same conclusion can be drawn from the results of this thesis where a few networks did a very good job filtering out false alarms while still managing to send a good amount of true alarms. Some limitations still exists which will be mentioned in section 4.2.

4.1 The Best Network and Optimizations

MobileNetV3 is by far the best network when weighing performance and accuracy. Running the best optimizations MobileNetV3 scores an average true alarm percentage of 60%, an average false alarm percentage of almost 0% and 20% skipped frames (figure 3.9) it performs well enough in all categories that is has no real competitor. Its only weak side is its performance as it has a feed forward time double that of MobileNetV1 Arm NN but it makes up for it by doubling the accuracy of MobileNetV1 Arm NN. However it is important to note that, like other networks, it has a much harder time detecting objects during night. This does of course mean that the network's day performance is even better than the average for all videos making it the ideal candidate for the network to use for day scenes. However, MobileNetV3 is still the best network for the night videos as well, although it probably is not good enough that it could be used reliably in the same capacity that it can be used for daylight scenes. Comparing figure 3.9 with 3.8, using tengine significantly reduces the amount of skipped frames.

The first optimization that can be used with MobileNetV3 that increased the accuracy was padding of subframes. Padding the subframes increased the accuracy of all networks,

even the nonviable ones, and also halved the number of false alarms for MobileNetV3. This points to it being a very good optimization for accuracy, at least in the conditions tested in this thesis.

Another optimization that increased both accuracy and performance was using 250x250 as the network/subframe size. This is of course dependent on just the specific scene for the benchmark videos but every scene will have an optimal network size that when found will get the best possible balance between accuracy and performance. Future work on dynamic network size is discussed later in section 4.3.10.

The detector pool did not improve the mean average skipped frames, except for videos with a high number of bounding boxes. The conclusion from this is that it in practice the optimization will not be worth it since it puts a high load on the system. An exception from this is if multiple cameras were to be used, then reducing congestion will be of utmost importance and therefore using the detector pool might be a viable option.

4.2 Limitations

There are a few limitations with the solution in this thesis as it stands right now, some of their possible solutions will be discussed in section 4.3. These limitations both affect the performance and the accuracy of the solution and are of varying difficulty to address.

4.2.1 Night Video

The first limitation has been described in part already in section 2.1.6 and it concerns night scenes. The problem that arises with night scenes is one of accuracy as all networks perform worse at night (as expected). Some networks handle night scenes better than others, for example the quantized Arm NN MobileNetV1 is almost completely unable to detect anything at night whilst the other MobileNet version only lower their accuracy by about **50%**. Looking at the nonviable networks we can see that the InceptionNet variants also lose about **50%** of their accuracy and the same goes for ResNet50 which indicates that even the more accurate networks have trouble with night scenes.

It is worth noting that the graphs can be somewhat misleading when it comes with performance on the night videos as for the most part the night videos only contain one motion detection bounding box per event, something that results in lesser skipped frames than if there are several motion detection bounding boxes.

4.2.2 Truck vs. Car

As described in section 2.1.5, the truck vs. car problem is something that is pretty hard to address. A problem that will occur if networks classify something like a pickup-truck in different ways (meaning one network classifies it as a car and another as a truck) is an unfair difference in accuracy. In order to solve this problem the two classes truck and car are treated as equals when evaluating the results from the benchmark, meaning that a car is a truck and a truck is a car. The reason why this is possible is because in the benchmark videos there are no proper trucks (meaning long-haul or tow type trucks), there are only classes within the intersection of the two classes (ie. pickup-trucks and vans). Without this change, every

network that leans toward detecting pickup-trucks as trucks will get many missed alarms and networks that lean the opposite way get many false alarms. With this change the networks are more equal and no network should be adversely affected by detecting vans as cars instead of trucks.

4.3 Future Work

Due to time or resource constraints there were several things not included in this thesis that we (the authors) would have liked to try or implement. The following sections will describe the future work in no particular order as well as what the theoretical benefits of such work would be.

4.3.1 Testing More Networks

One obvious expansion to this thesis would be to compare more neural networks against the ones already done. The main reason why this was left out of the thesis is because, at the time of writing, running one benchmark with all the networks and all the videos takes around 3 hours and that does not include comparing any custom parameters such as padding or scale factor. With more time and access to more pre-trained networks (or networks trained just for this application) it might be possible to increase both the accuracy and the speed of the object detection on the hardware ultimately lowering the number false alarms. Specifically training custom versions of MobileNetV3 or MobileNetV2 might increase accuracy and speed by a significant margin by using fewer classes than the 90 that the COCO-dataset has (after all the probability of an oven triggering motion detection is rather low).

4.3.2 Productify

In order for the work done in this thesis to actually decrease the number of false alarms AXIS Companion, the code written for this thesis would need to be turned into a final product that can be run on the new recorder. For this to happen several things need to be done. Firstly memory and CPU usage constraints need to be investigated so that the recorder can perform its main purpose (to record video) without any interruptions because the object detector used too much memory or CPU.

Decoding video has been a substantial performance issue because FFmpeg has done it all in software without any hardware acceleration which takes a substantial amount of CPU power, although somewhat mitigated by NEON accelerated decoding functions. As mentioned in Section B.1, the board has a VPU (Video Processing Unit) targeted specifically for solving this issue. However interfacing against it requires special firmware that was not available at the time of implementation. To allow the solution proposed in this thesis all video decoding should be done on the VPU to reduce the overall CPU load. FFmpeg includes bindings for OpenCL to accelerate video on a GPU, which also could be a solution for this problem, especially if more video streams would have to be decoded simultaneously.

The context for this work is security, therefore a decision regarding what should happen if the amount of bounding boxes suddenly goes up dramatically. This is a problem because a lot of frames will start to get skipped by the system and therefore potential alarms will get

lost. A potential easy solution is to fallback to the naive motion detection if too many frames get skipped and notify the user directly.

As the results indicate, in darker settings all networks perform significantly worse. This is intuitively true but poses a problem where productfication of this work is concerned. Consider a small store owner that would use this product, the time of day where accurate detection notifications would be most important is probably during the night. Mitigations for this could include using the IR sensor of Axis cameras to get better images and running networks of higher quality with slower detections during night, assuming that there is less motion activity during the night than during the day.

The last thing that needs to be done is to integrate the application with the Axis ecosystem to make it run like existing applications that run on the devices (ACAPs). This includes things such as having the application start when camera boots and restart in case of crashes etc. Some way to signaling the output (ie. alarm or not alarm) is also needed and might be done using an RTSP stream that can be subscribed to from wherever the information is needed.

4.3.3 Tengine vs OpenCV vs Arm NN Comparison

Speed is an important factor when deciding which machine learning library to use for an application. However, as with all software it is important to verify if the product performs as advertised. At the time of writing the latest version of Tengine, OpenCV and Arm NN was used because of recent performance improvements of executing Neural Networks on aarch64 hardware. Since they are all open source libraries, caution should be used as a consequence of that no accuracy guarantees or warranty is provided. Therefore more thorough testing of these libraries should be performed, including comparing the output of these libraries against each other and verifying that each layer of the network actually produces the correct result. Verifying the accuracy of an implementation of a neural network is a time consuming task and is therefore not in the scope of this work but should be performed in the future.

4.3.4 Hardware Acceleration

One key area and reason for modern development of hardware acceleration is machine learning, in particular neural networks. GPUs are particularly good at performing this task since they are massively parallel computational devices, specialized at performing floating point operations. All object detection in this work has been performed on a CPU since proper drivers for the GPU in the i.MX 8 QuadPlus board did not exist at the time of implementation. Utilizing all the compute power, in particular in an embedded system, is very important and therefore using the GPU should be examined in the future. OpenCV and Arm NN both have built-in support for OpenCL, which means that in theory if OpenCL drivers would be implemented for the GPU, then extra compute resources would instantly be available. OpenCV can dynamically be configured to run on both CPU and GPU which means that the existing detector pool implementation could remain intact running parallel detections on the CPU and GPU simultaneously.

4.3.5 Concatenate Subframes

One possible optimization that did not get done was to concatenate all subframes into a larger frame and then doing only one feed forward instead of several. It is hard to say whether this would have lead to better accuracy but in theory it should lead to better performance. This is because most of the subframes obtained from the benchmark videos have around 100 pixels as their longest side meaning that after padding they end up being 100x100 pixels. They are then resized to whatever network size is used for the particular network. So for example say that the network takes 200x200 images as inputs, then it would be possible to feed forward 4 subframes concatenated into one 200x200 image instead of doing 4 separate feed forwards. This would yield a theoretical 4x performance increase. More subframes could be fit into the 200x200 image if no padding was done but at the cost of a more complex algorithm to create the concatenated image. There is however the problem of determining which image contained which object but using the location output from the network solves this albeit with some extra work.

4.3.6 Detect Objects Once

When using AXIS VCS (described in section B.2.7) as the metadata source for motion detection there is one vital piece of information that is not available in VMD metadata: object id's. Object id's can be used to keep track of objects that have already been detected and thus allowing for possibly the largest optimization of the ones described in this thesis. If an object is detected successfully as an object that should not send an alarm it can be ignored for the rest of its lifetime and if it merges with another uninteresting object that new object can also be ignored. In theory this could mean that objects that spend 10 metadata events in the frame only need to be fed through the network once, leading to 10 times performance increase. It might also increase the accuracy in cases such as the tree problem described in section 2.1.2 since once an object is detected it will keep its detected class even if it gets partially obstructed by something else. Of course there are potential problems with this approach in regards to incorrect initial classifications leading to the incorrect class being applied to an object for the duration of its lifespan but this might be solved with having a separate higher threshold limit when detecting objects for the duration of their lifespan.

4.3.7 Multi Object Classifiers vs Object Detectors

Object detectors was the logical choice for this thesis because of their widespread use and easily available pre-trained networks but there are other kinds of networks that might be applicable for the use case of this thesis. Instead of using an object detector like MobileNet which produces both detections and the location of the detections it might be possible to use networks that only classify the image. The main reason this is possible is because of the use of motion detection which generates subframes. Sending these subframes through a classifier and getting only the objects within the image might be sufficient for alarm filtering but a problem arises with several objects in one subframe. It might also be possible to train a binary classifier which classifies a subframe into alarm or not alarm but that would mean losing the ability to configure which alarm classes to use after training.

4.3.8 Creating a Custom Dataset

Another time consuming but presumably worthwhile expansion to this thesis would be to train an existing or new network on a custom dataset created from images taken on Axis cameras or surveillance video in general. Since the COCO-dataset described in section 2.4.6 has images mostly taken with cellphones or more expensive cameras one would expect the networks trained on the COCO-dataset to perform worse on images taken with surveillance cameras which do not have the same quality. Training on a dataset which contains images with more grainy images from angles that are more usual in surveillance situations might increase the accuracy of the networks and pairing that with using fewer classes might also increase the performance.

4.3.9 TensorFlow Lite as Backend

This thesis has only used OpenCV, Tengine and Arm NN as backends but there is another alternative that has not been explored and that is TensorFlow Lite. TensorFlow Lite is developed for use on devices such as cellphones and might be a good candidate for use on the recorder. It is hard to say whether this would lead to an increase in performance but it is not unreasonable that it would since the recorder uses the same architecture (aarch64) as modern day cellphones which can run object detection in real time. Many of networks used in this thesis were trained with TensorFlow and their models converted to OpenCVs format and would therefore already be compatible with TensorFlow Lite as well.

4.3.10 Adjust Input Size According to Scene

There are many possible scenes where object detection might be useful and all of these different scenes will have their own perfect parameters for use in filtering alarms. One parameter which would be interesting to vary depending on the scene is the network size. For example: imagine a camera positioned far from the subject which it records. Anything that would trigger motion detection would likely take up a small part of the frame and it is therefore unnecessary to take the subframe from the bounding box and upscale it every time. Upscaling the image adds no new information and makes it take longer to feed through the network. A better approach would be for the application to "learn" what the typical size of a subframe is and adapt its detectors network input size to be around that size. This might be a huge improvement in both accuracy and performance in some scenes while much smaller in others but in general should in theory lead to a better solution. The largest hurdle to overcome to implement this is to figure out when to change the parameters since it will require creating a new detector which can be time and memory consuming. Extracting one detector from the pool, deleting it and inserting a new detector might be the best way to achieve this.

References

[1] Opencv change logs. https://github.com/opencv/opencv/wiki/ChangeLog#version430.

[2] Post-training quantization. https://www.tensorflow.org/lite/performance/post_training_quantization.

[3] Yocto project website. https://www.yoctoproject.org.

[4] ARM. Neon programmer's guide. https://static.docs.arm.com/den0018/a/DEN0018A_neon_programmers_guide_en.pdf.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Yunhui Guo. A survey on methods and theories of quantized neural networks, 2018.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[8] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[11] Jonathan Lundholm and Paul Maxwell Steneram Bibby. False alarm filtering within camera surveillance using an external object classification service, 2017. Student Paper.

[12] Prateeth Nayak, David Zhang, and Sek Chai. Bit efficient quantization for deep neural networks, 2019.

[13] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

Appendices

Appendix A

Results

A.1 All Networks and Videos

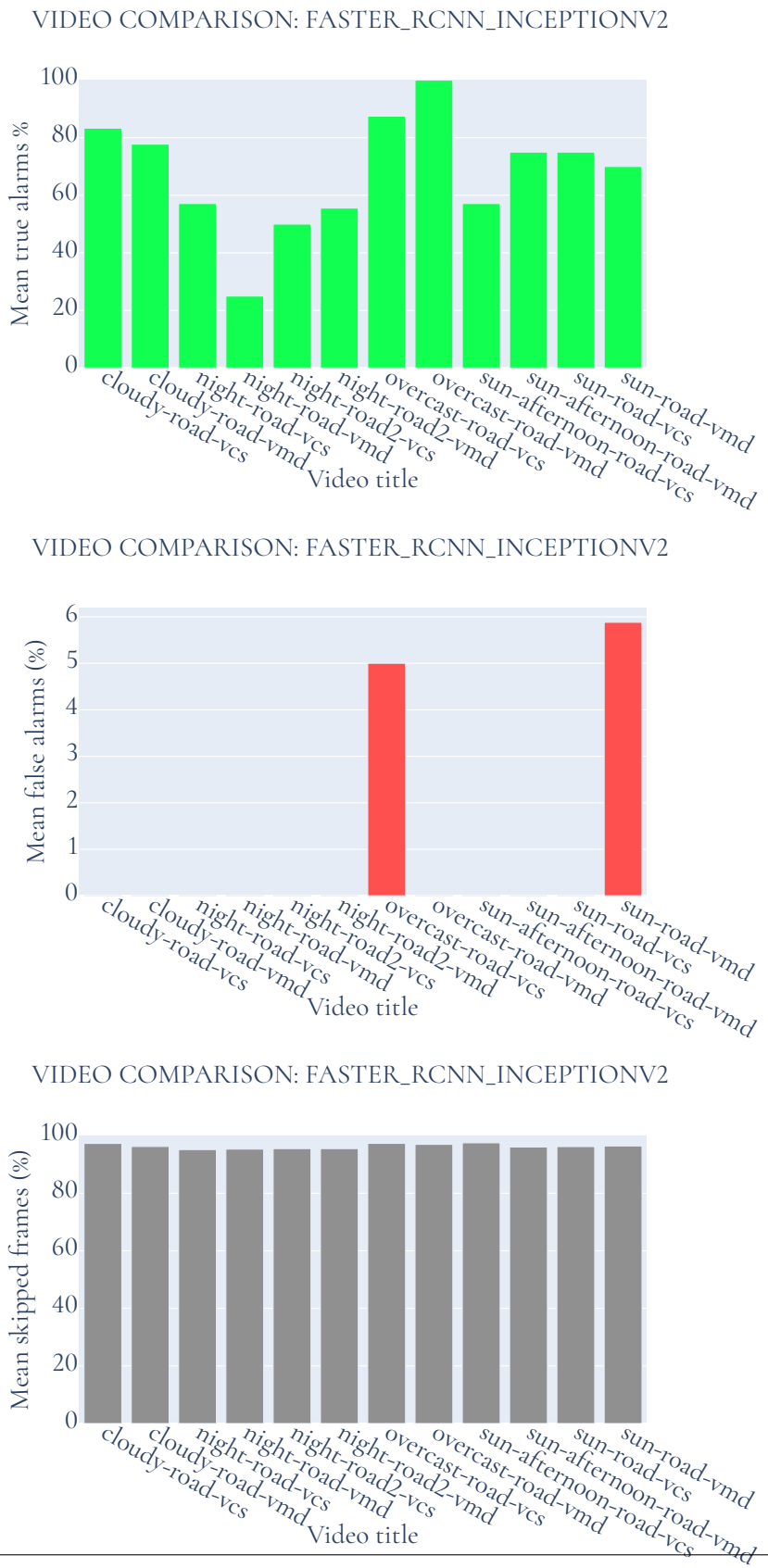


Figure A.1: Individual results for all videos for faster_rcnn_inceptionv2

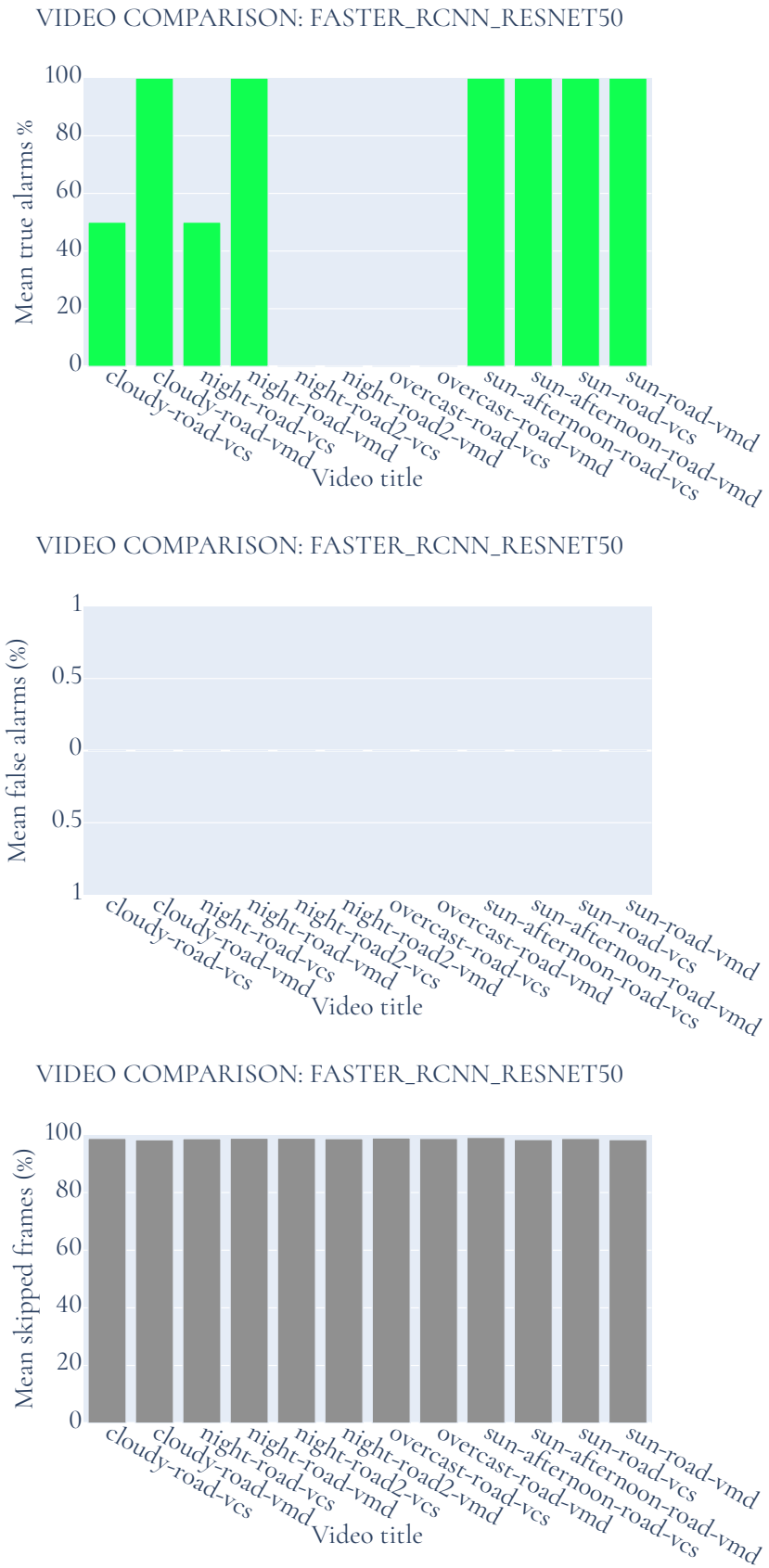


Figure A.2: Individual results for all videos for faster_rcnn_resnet50

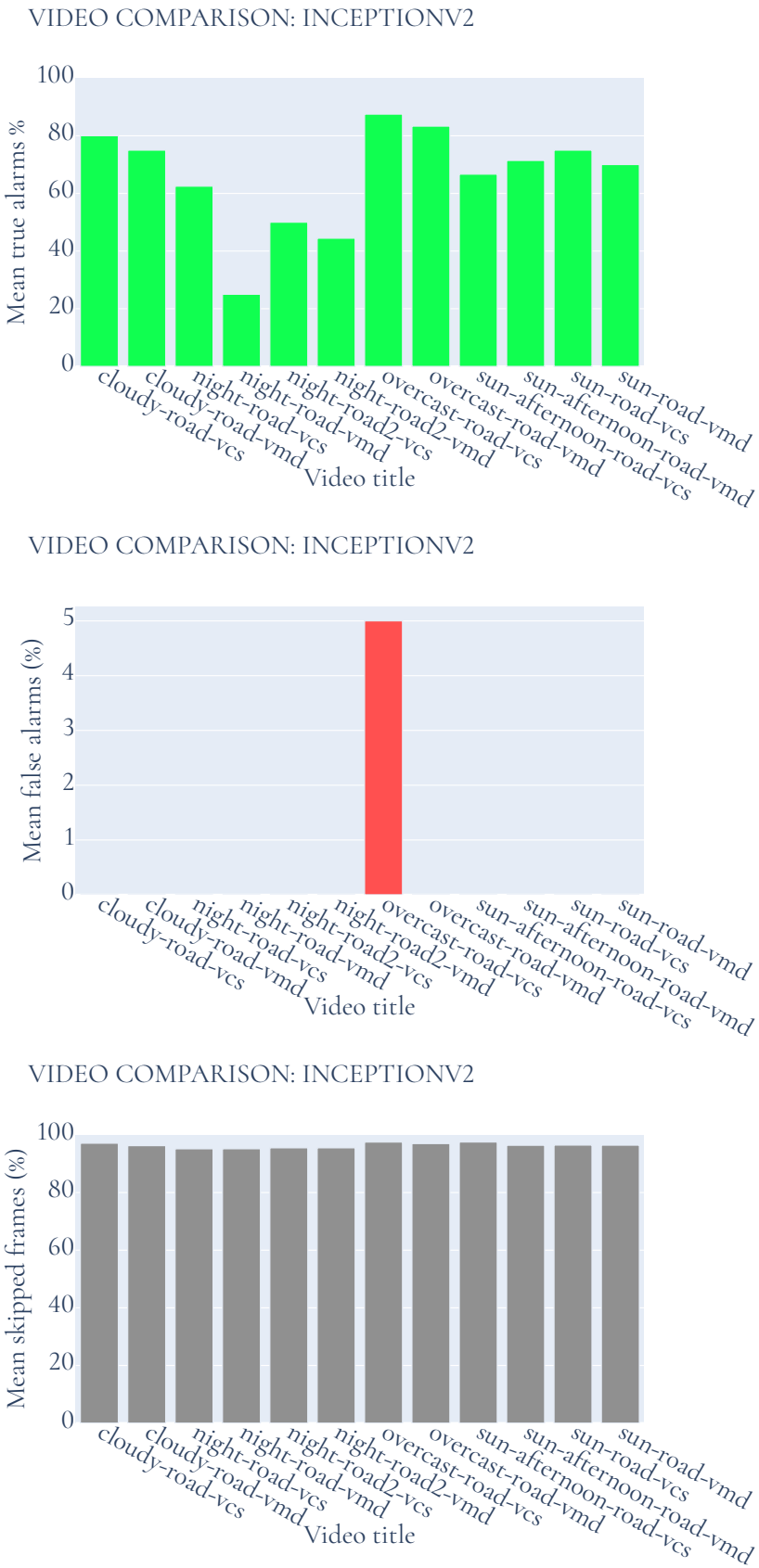


Figure A.3: Individual results for all videos for faster_rcnn_inceptionv2

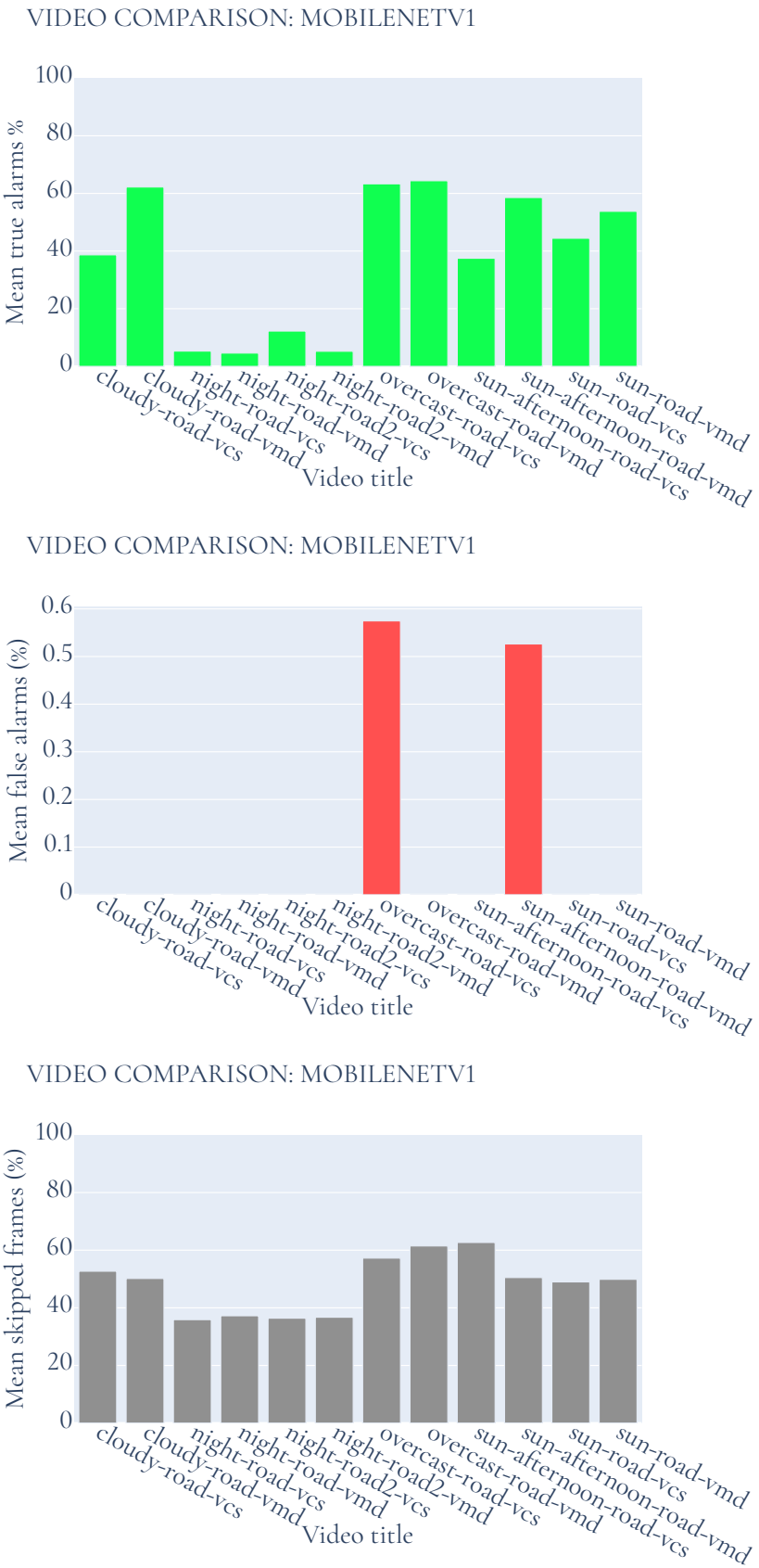


Figure A.4: Individual results for all videos for mobilenetv1

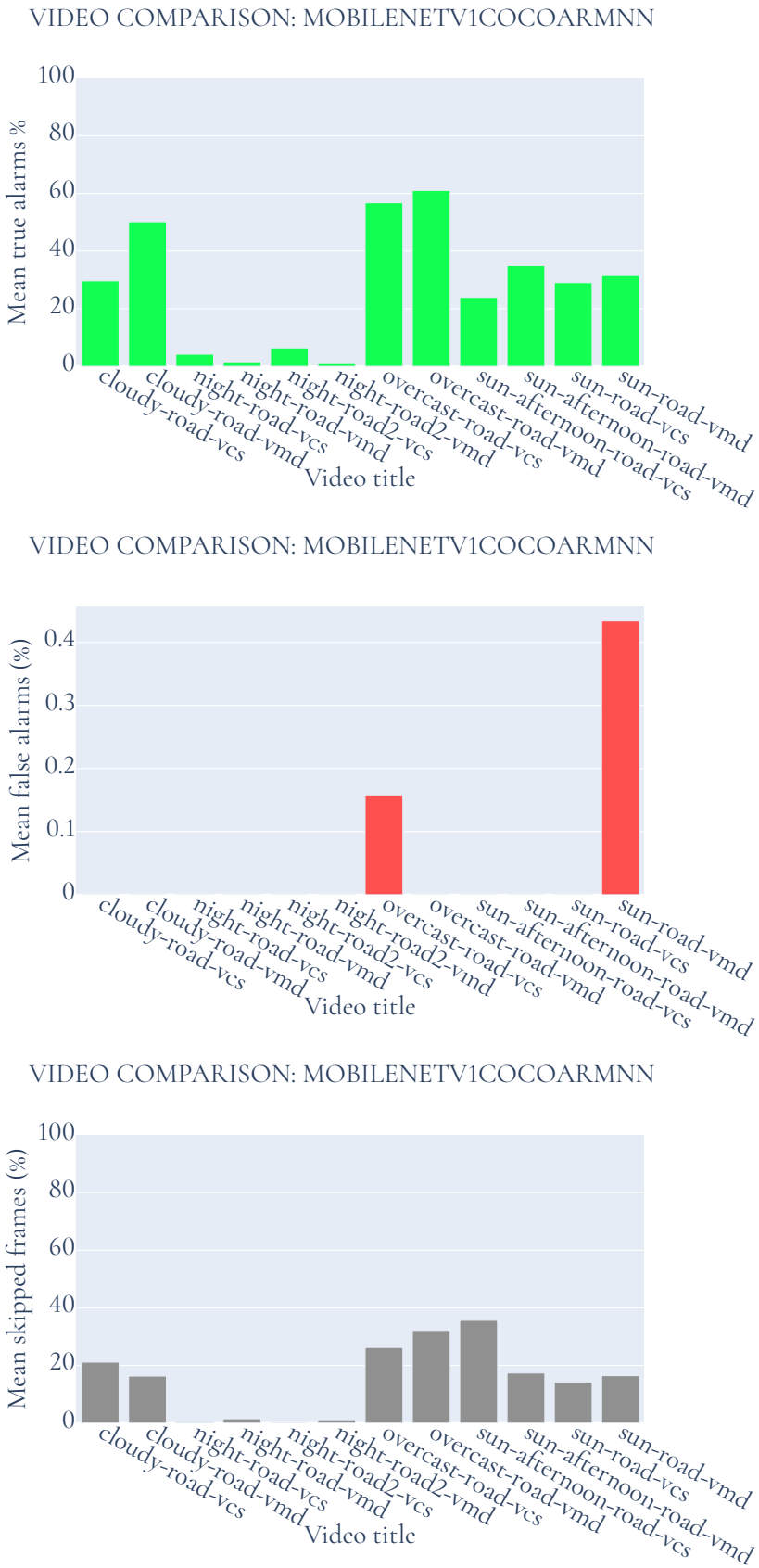


Figure A.5: Individual results for all videos for mobilenetv1cocoarmnn

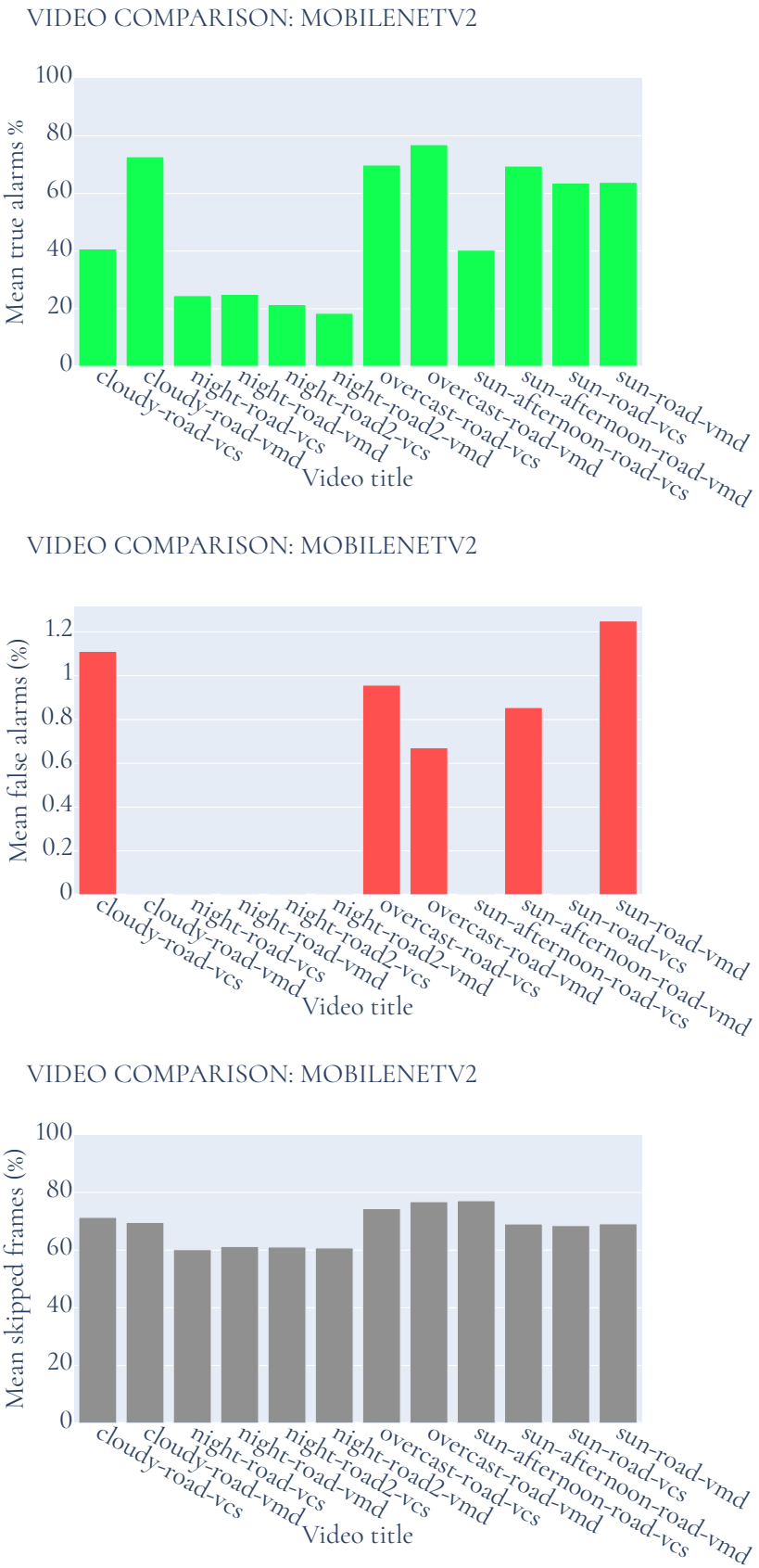


Figure A.6: Individual results for all videos for mobilenetv2

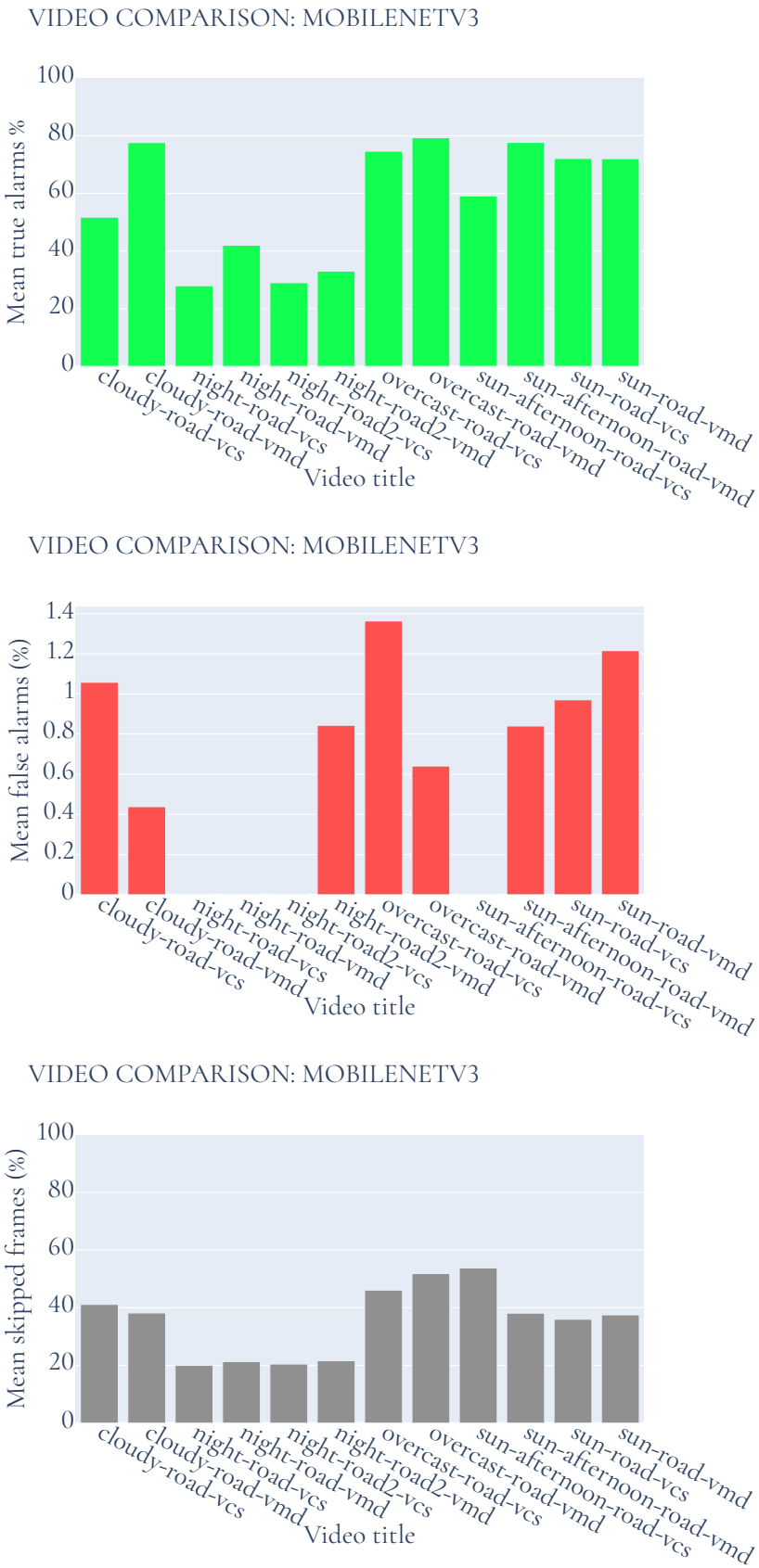


Figure A.7: Individual results for all videos for mobilenetv3

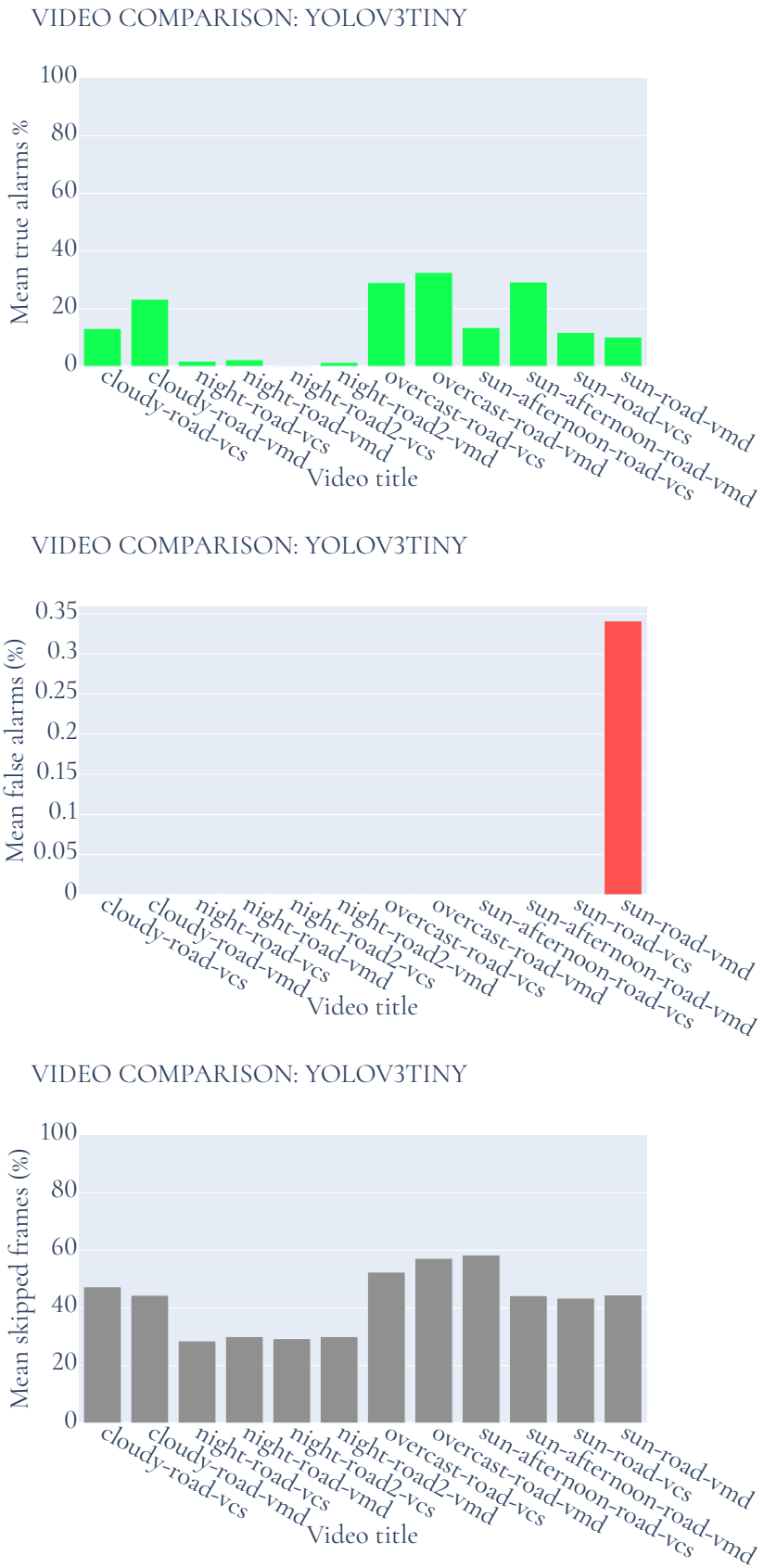


Figure A.8: Individual results for all videos for yolov3tiny

Appendix B
Prerequisites

B.1 Hardware

The new recorder mentioned previously uses a chip from the company NXP Semiconductors based in the Netherlands. The chip's name is i.MX 8 QuadPlus and it features the following relevant hardware components:

- 4x ARM Cortex-A53 CPUs at 1.2 GHz each
- 1x ARM Cortex-A72 CPU at 1.6 GHz
- 2x Vivante GC7000Lite GPUs
- 1x VPU

The Cortex-A72 is designed to be a more powerful core used for more demanding computations but uses more energy than Cortex-A53. It allows for scheduling more demanding work that benefits of a lower latency.

B.2 Software

On the recorder axOS is run as the operating system, it is an Axis specific Linux platform built on the Yocto Project [3]and contains the bare necessities for debugging the applications that the recorder runs during everyday use. The following are the higher level libraries that we used in order during this thesis.

B.2.1 OpenCV

OpenCV is large and popular open source and is used in this thesis to handle image processing, video streaming and running some of the neural networks. OpenCV was chosen for its

ease of use and its many image processing, GUI and neural network features but mostly the Deep Neural Network (DNN) module and base module (containing such data structures as matrices for video frames for example) is what is used in this thesis.

B.2.2 FFmpeg

FFmpeg is a popular cross-platform open source video decoding and encoding library. It was used as the back end for OpenCV to retrieve and record the RTSP stream of video sent by the camera.

B.2.3 RTSP

The Real Time Streaming Protocol (RTSP) is a protocol for use when streaming real time data. It was created to stream video and audio from either live data feeds or from stored files and can be used over both TCP and UDP. Axis cameras feature the ability to get live streamed video directly from the camera through an RTSP stream as well as have the capability to send metadata over RTSP.

B.2.4 Tengine

Tengine is an open source library developed by OPEN AI LAB targeted for AIoT (Artificial Intelligence in IoT). It contains parsers for popular machine learning libraries as Tensor Flow and Caffe with hand vectorized convolution layers for x86 and aarch64. In version 4.3 of OpenCV, Tengine was integrated as an optional backend for the DNN module, mainly to increase the performance of applications targeting aarch64 [1].

B.2.5 Arm NN

Arm NN is a high performance open-source inference engine specially developed for Arm processors by Arm. It relies on the Arm Compute Library to provide high performance convolutions and network layers that are hand tuned for and implemented with Arm NEON instructions. When performing convolutions on a CPU in embedded systems it is important to utilize all of the power available and this can therefore improve performance significantly. It exposes the ability to use both fully quantized 8bit and 32bit floating point convolutions with the trade-off being a balance of speed, accuracy and size of the trained models. Also provided by the library is the ability to translate and run models trained from popular machine learning libraries such as Caffe and TensorFlow.

This library has the advantage that it is significantly faster than OpenCV but has the disadvantage that it is not as mature and more complex to use. At the point of writing there is no option to use Arm NN as a back-end for OpenCV and it must therefore be integrated separately.

B.2.6 AXIS Video Motion Detection (VMD)

Axis Video Motion Detection, or VMD for short, is what AXIS Companion uses for motion detection. VMD can be configured to filter out small, swaying and short lived objects. The data from VMD used in this thesis is called visual confirmation and is accessed via an RTSP stream where the camera will send a list of bounding boxes, one for each moving object, and a timestamp when the motion occurred. To enable VMD to send bounding boxes the application has to send a HTTP POST request to the camera which will tell the camera to send visual confirmation data for 15 minutes. In order to keep visual confirmation data on indefinitely the application needs to send these HTTP requests around once every 10 minutes to ensure there is no downtime in the RTSP stream.

B.2.7 AXIS Video Content Stream (VCS)

Axis Video Content Stream or VCS is similar to VMD in that it also sends motion detection information over an RTSP stream from the camera but it differs in some aspects. Primarily it sends more detailed data. VCS sends everything VMD does but adds object id's, object splitting/merging information (when bounding boxes cross over each other for example) and polygons for each detected object. VCS also does not need to be enabled every 15 minutes and will simply always be running.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-06-11

EXAMENSARBETE Motion detection alarm verification using deep learning in surveillance systems
STUDENTER Jonathan Strandberg, Erik Rosengren
HANDLEDARE Jörn Janneck (LTH)
EXAMINATOR Mathias Haage (LTH)

Smartare videoövervakning för det moderna samhället

POPULÄRVETENSKAPLIG SAMMANFATTNING **Jonathan Strandberg, Erik Rosengren**

Videoövervakning är traditionellt ett jobb reserverat för människor eller den mest kraftfulla hårdvaran som pengar kan köpa, det här arbetet undersöker om det finns en robust och mer kostnadseffektiv lösning som kan vara tillgänglig för alla.

I Axis övervakningsystem idag finns det lösningar som existerar för att ge användare möjligheten att få notifikationer till sina telefoner om det är något som rör sig i kamerans vy. Det uppenbara problemet som uppstår är att om en plastpåse skulle blåsa förbi mitt i natten, då kommer användaren att få en notifikation helt i onödan. Eller så är kameran riktad mot en väg där många bilar åker förbi men de kanske inte är så intressanta. Kanske skulle man kunna kombinera den existerande lösningen med modern maskininlärning för att endast få den information som användaren faktiskt bryr sig om? Det lade grunden för arbetet som vi har gjort. I systemen så finns det en sk. **recorder** som har ansvar att spela in video, men har resurser som tidigare har varit till stor del outnyttjade.

Med hjälp av moderna neurala nätverk så sänkte vi mängden falska alarm ner till 0,3%. Det här samtidigt som systemet kunde korrekt klassificera korrekta alarm till 60%. Att detektera 60% kan intuitivt kännas som lite, men på grund av att objekt rör sig i flera sekunder på en video så närmar sig sannolikheten 100% desto längre objektet finns i vyn.

Neurala nätverk är kraftfulla verktyg för maskinlärning då de drar inspiration från hur hjärnan till synes kan lära sig väldigt komplicerade uppgifter. Med den styrkan följer tyvärr motsvarande krav på hårdvaran och de bästa nätverken körs på otroligt dyra datorer. Det här självklart inte accepterbart för ett enkelt övervakningsystem! Speciellt när det ska köras på recordern som har begränsad prestanda.

För att uppnå så bra resultat som möjligt så jämförde vi många olika typer av moderna neurala nätverk mot varandra. Alla nätverk testades på 12 olika videoinspelningar med olika ljusförhållanden för att se vilka omständigheter som hade störst inverkan på hur bra de presterade. Den överlägsna vinnaren som hittade den bästa balansen var MobileNetV3 som kan optimeras väldigt mycket för hårdvaran.