Bachelor's thesis

Information and Communications Technology

2021

Daniel Kusnetsoff

MOBILE REAL-TIME OBJECT DETECTION WITH FLUTTER

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2021 | 42 pages, 1 page in appendices

Daniel Kusnetsoff

MOBILE REAL-TIME OBJECT DETECTION WITH FLUTTER

The utilization of computer vision has significantly increased in everyday devices such as mobile phones. Computer vision, such as object detection, is based on deep learning models. These models have traditionally needed high-performance graphics processing units for training and utilization. However, even if it is still not possible to effectively train the models with lower-performance devices, it has lately become possible to use the models with them. It is critical for model performance to develop the mobile application optimally. The choice of the ideal framework and user interface is crucial as the user interface architecture sets the constraints for the model's performance. The framework chosen in this thesis, Flutter has an architecture that benefits real-time features in object detection better than other frameworks.

A mobile application was developed for this thesis to research the possibilities of using Flutter in mobile real-time object detection. The application presents two forms of computer vision: object detection and image captioning. For object detection, the application provides real-time predictions using the camera. The object detection feature utilizes transfer learning and uses two object detectors: Tiny-YOLO v4 and SSD Mobilenet v2. They are the most popular detectors and provide a balance between detection accuracy and speed.

As a result of the thesis, a successful Flutter-based mobile application was developed. The application presents the differences between the YOLO-based and SSD-based models in accuracy and speed. Furthermore, the image caption generator shows how an external deep learning model can be utilized in mobile applications. As importantly, the image caption generator works near real-time by predicting the image caption with high accuracy. Both computer vision features function optimally due to the Flutter-based architecture and structure. Flutter provides high performance and reliability in both computer vision tasks featured in the application.

KEYWORDS:

Computer vision, object detection, Flutter, image captioning, deep learning, YOLO, SSD

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2021 | 42 sivua, 1 liitesivua

Daniel Kusnetsoff

FLUTTERIN HYÖDYNTÄMINEN MOBIILISSA HAHMONTUNNISTAMISESSA

Tietokonenäön käyttö on kasvanut huomattavasti matkapuhelimissa. Tietokonenäön tärkein osa, hahmontunnistus, perustuu syviin oppimismalleihin. Nämä mallit ovat perinteisesti vaatineet tehokkaita grafiikkaprosessoriyksiköitä mallin koulutukseen ja käyttöön. Vaikka mallien tehokas kouluttaminen pienitehoisemmilla laitteilla, kuten matkapuhelimilla, ei ole mahdollista, on viime vuosina teknologia kehittynyt niin, että malleja voidaan hyödyntää mobiilisovelluksissa. Mallin suorituskyvyn kannalta on ratkaisevan tärkeää kehittää mobiilisovellus tekemällä oikeita valintoja. Sopivan kehysympäristön valinta on ratkaisevan tärkeää, koska ympäristön määrittämä käyttöliittymäarkkitehtuuri asettaa rajoitukset mallin suorituskyvylle. Opinnäytetyössä käytetty Flutter-kehysympäristön arkkitehtuuri hyödyntää reaaliaikaisia hahmojen tunnistustoimintoja paremmin kuin muut kilpailevat kehysympäristöt.

Tätä opinnäytetyötä varten kehitettiin mobiilisovellus, jolla tutkittiin mahdollisuuksia käyttää Flutteria ja sen arkkitehtuurin mahdollistamaa nopeata reaaliaikaista kohteen tunnistusta. Sovelluksessa tutkittiin kahta tietokonenäön muotoa: esineiden havaitseminen ja kuvan tekstitys. Kohteen havaitsemiseksi sovellus näyttää reaaliaikaisia ennusteita kameran avulla. Kohteen tunnistusominaisuus hyödyntää siirto-oppimista ja käyttää kahta kohdeilmaisinta: Tiny-YOLO v4 ja SSD Mobilenet v2. Ne ovat suosituimpia ilmaisimia ja tuovat tasapainoa tunnistustarkkuuden ja nopeuden välillä.

Opinnäytetyön tuloksena kehitettiin toimiva Flutter-pohjainen mobiilisovellus. Sovellus esittelee eroja YOLO- ja SSD-pohjaisten mallien välillä tarkkuuden ja nopeuden osalta. Lisäksi kuvatekstitysgeneraattori näyttää, kuinka ulkoista syväoppimisen mallia voidaan käyttää mobiilisovelluksissa reaaliaikaisesti. Tärkeintä on, että kuvatekstitysgeneraattori toimii lähes reaaliajassa ennustamalla kuvatekstin tarkasti. Molemmat tietokonenäköominaisuudet toimivat erinomaisesti Flutter-pohjaisen arkkitehtuurin ja rakenteiden ansiosta. Flutter-käyttöliittymä tuottaa korkean suorituskyvyn ja luotettavuuden molemmissa sovelluksessa esitetyissä tietokonenäkötoiminnoissa.

ASIASANAT:

Tietokonenäkö, hahmontunnistus, Flutter, kuvatekstitys, syväoppiminen, YOLO, SSD

CONTENTS

APPENDICES

FIGURES

TABLES

PROGRAMS

LIST OF ABBREVIATIONS

AI	Artificial Intelligence. Technology that resembles human intelligence by learning.
CNN	Convolutional neural network. A deep learning network resembling the human brain. Often used for image recognition tasks
COCO	Common objects in contexts. A large well-versed dataset containing everyday objects.
CPU	Central processing unit. Processor or core on which the microprocessor implements functions.
FPS	Frames per second. The amount of times a device updates the view on its display.
GPU	Graphics processing unit. The component on computers that is in charge of rendering images and videos.
iOS	Mobile operating system developed for Apple devices.
mAP	Mean Average Precision. The mean of the area under the precision-recall curve. (Yohanandan, 2020)
NMS	Non-maximum suppression. A technique to choose the best bounding box of the proposed ones in object detection.
OEM	Original equipment manufacturer. A company that produces components to be used by other companies in their products.
pb	Protobuf. A TensorFlow file type containing a graph definition and model weights.
ROI	Region of interest. Image part where the identified objects are thought to be and whereas result bounding boxes are added.
SDK	Software development kit. A package with collected tools for software development.
SSD	Single-shot detector. A single-stage object detector known for its accuracy.
UI	User Interface. The contact point a user interacts with a device or an application.
YOLO	You only look once. A single-stage detector known for its speed.

1 INTRODUCTION

Due to the current rising popularity of interest in artificial intelligence or AI, major advancements have been made in the field, and the use of AI has become an indispensable part of everyday life. These advancements have led to transferring AI and Machine Learning features from high-performance devices to lower-performance devices. However, AI and Machine Learning have traditionally relied on considerable computing power. The developments in Machine Learning have transferred from the use of central processing units (CPU) - to the use of servers and more effective graphics processing units (GPU). CPUs are processors or cores on which a processor implements functions and GPUs are components on computers in charge of rendering images and videos. These advances in performance have advanced the field from the limitations of just the local calculating power to the calculating power of servers end external GPUs. This growth in calculation power has made it possible for complex Machine Learning models to be developed. However, the development in technology has only partly made it possible to transfer Machine Learning features to lower-performance devices such as mobile phones due to the fact that the training still needs to be carried out on high-performance devices. (Council of Europe, 2020.; Dsouza, 2020.)

The primary goal of this thesis is to combine and research complex Machine Learning models and lower-performance devices through building a cross-platform mobile application that uses Machine Learning. The application provides object detection -and image caption predictions by pointing the mobile phone camera towards the targets. Before the advancements in Machine Learning in recent years, this had been impossible as mobile phones have not had enough calculating power to show the results in real-time. The application is built using Flutter, which is a relatively new user interface or UI developed by Google and has an architecture that should benefit object detection. Searching through the work in Finnish universities, not a single research or thesis could be found about combining the Flutter user interface and object detection. A secondary goal for the thesis is to research the differences in real-time performance between the object detection feature that is based on local Machine Learning models and the image caption generator that uses a prebuilt external model. For the image caption generator, the external online-based model should affect the speed of predicting the situation. The developed application should provide concrete results if the Flutter UI framework is optimal for mobile object detection. (Flutter, 2021a.)

2 DEEP LEARNING METHODS AND TECHNOLOGIES

2.1 Artificial intelligence and Machine Learning

Artificial intelligence or AI can be thought of as an autonomous tool between an input such as sensors and images and an output such as action and text. The idea is that using AI, machines are capable of performing tasks that previously would have required human interference. AI can be divided into two categories: narrow AI and Artificial General Intelligence. Narrow AI consists of applications within a limited context and usually focuses on completing a single task according to the instructions. Applications such as autonomously driving cars or a basic Google search are referred to as narrow AI. On the other hand, artificial general intelligence are systems that can solve many or all tasks and resemble human intelligence. (Built in, 2019.)

Machine Learning can be defined in different ways, but a relatively simple definition is that Machine Learning is the programming of computers so that they can learn from data. Machine Learning is based on feeding a computer as much data as possible for the computer to learn the patterns and relationships of the data. Machine Learning is often thought of as very difficult, as it is based on complex data. However, the main goal of Machine Learning is to be able to predict the results without needing to understand all parts of the data complexity. Artificial intelligence can be thought of as almost all learning a computer does. However, there is a thin line between what simply has been stored in memory and what a computed has processed and learned. For a system to have AI, it needs to work independently. Additionally, this means the computer needs to learn to utilize the data for it to be called Machine Learning. The main difference between Machine Learning and AI is that Machine Learning is heavily instructed by humans. There are still differences in the methods of different types of Machine Learning, for example supervised and unsupervised learning. (Iriondo, 2018) As can be seen in Figure 1, Machine Learning is a significant component of artificial intelligence that also contains deep learning and neural networks. (Géron, 2017, 3-4.)

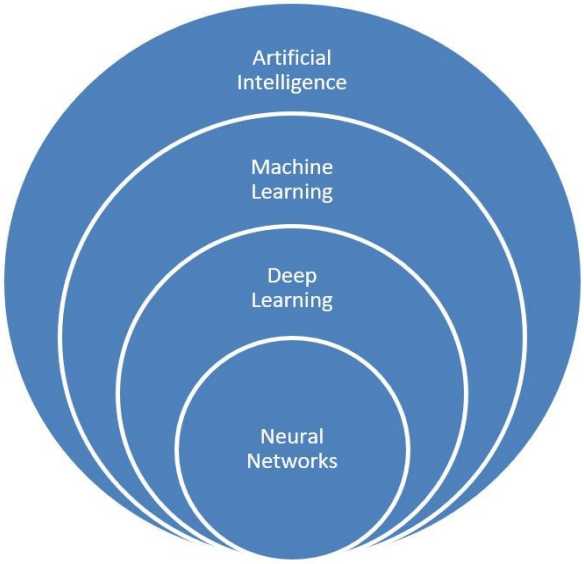

Figure 1. Artificial intelligence and Machine Learning.

In Machine Learning, the data is usually divided into three datasets: training set, testing set, and validation set. The training set is the largest of the sets, and it usually is around 70-80% of all data. The function of the training set is for an algorithm to learn the priorly mentioned patterns and relationships from the data. After the algorithm has learned from the training set, it is introduced to the validation set. The validation set is used to estimate the model's effectiveness unbiasedly while the parameters are tuned. After the validation set, the model will go through the test set. The idea of the test set is that the final performance of the model is assessed with the data it has not previously seen. After going through these three datasets, the final model should provide as much accuracy in its predictions as possible. This process will take a varied amount of time depending on the choices and amount of data trained in the process. Training Machine Learning models can often take days or weeks, depending on the setup. (Brownlee, 2017.)

Deep learning and neural networks

Deep learning is a part of both AI and Machine Learning and has been the key to developing technologies such as autonomous driving and facial recognition. Deep learning is based on the idea of successive layers in neural networks. Deep learning models are usually built of tens or hundreds of these layers and have been trained using massive labelled datasets and neural networks. These layers work like filters, as they consist of mathematical functions that separate the features. For example, a single level of layers can determine if a hand-drawn number has the features of number 0 or 1.(Géron, 2017, 87-88.)

Neural networks are structures that resemble the neural system of the human brain. Each layer is built out of neurons that work as units. These neurons can be thought of as mathematical functions that take an input and calculate together a sum using the weight and activation function of the output at that layer. The output is then moved to the following layers until it reaches the output layer. These networks are constructed out of different layer types. The first layer in a neural network is an input layer, and the last layer is an output layer. Between these layers are many hidden layers. The hidden layers are built out of nodes with their determined weights and thresholds. Weights are the connection strength between neurons and thresholds can be thought of as filters, that according to the activation, decide if the input signal is sent to the next node. Starting from the input layer, the nodes in the following layer are activated according to their properties and send the data forward to the next layer. (Géron, 2017., 279-280; Kavlakoglu, 2020.; Yiu, 2019.)

Deep learning and neural networks are often discussed as synonyms even though neural networks is a subcategory of deep learning. A neural network with more than three layers is considered a deep learning algorithm. As seen in Figure 2, the added layers reside in the hidden layers and build up the deep learning algorithm. Adding and altering the properties of the hidden layers results in more effective nodes in a model. Additionally, it improves the training of the model to function more accurately. However, adding too many layers might cause the model to overfit. Overfitting is when a model learns details from a dataset too well and starts generalizing the details. (Brownlee, 2019b; Brownlee, 2019c.; Géron, 2017, 28-29.)

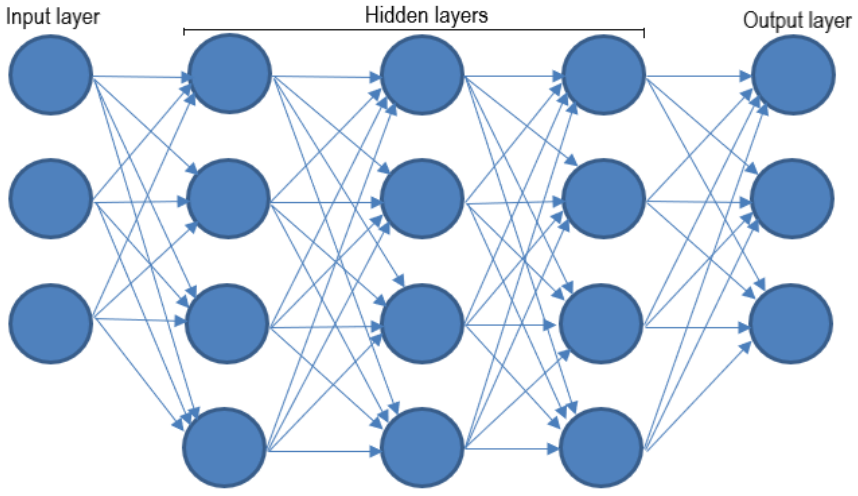

Figure 2. Example of a neural network.

2.2 Object recognition

Object recognition is the task of identifying objects from images, and it usually takes place via a neural network. It is essential to understand that videos are thought of as a series of images, and all video recognition tasks can be considered image recognition tasks. These tasks can be divided into image classification, object localization, and object detection. The image classification task focuses on identifying the object's class in an image. These objects in the image are given class labels. In object localization, the objects in the image are located, and a bounding box is set around the object. In the third task, object detection, the objects with bounding boxes and the class labels are located in the image. (Brownlee, 2019a.; Fritz AI, 2020.)

Object detection is a combination of both image classification and object localization, as can be seen in Figure 3. This process of object detection is possible using neural networks. Object detection is usually carried out with the help of one or more convolutional neural networks (CNN). A CNN is a neural network that has added convolutional layers to the hidden layers. The convolutional layers differ from standard hidden layers as there is an assumption that the input coming to the layers are images. This means that the neuron architecture is built for specific properties such as width, height, and depth. There are usually quite a few convolutional layers in a CNN, and they have the function of transforming the input before it travels to the next layer. (Brownlee, 2019a.)

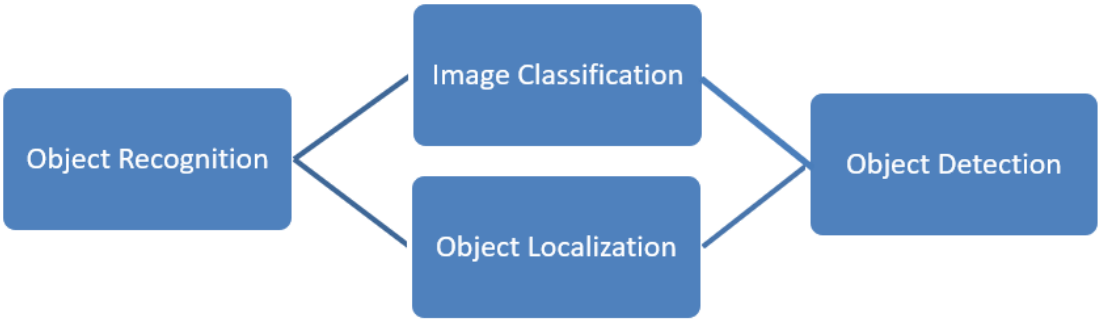

Figure 3. Object recognition structure.

2.2.1 Object detection frameworks

Object detection frameworks are combinations of tools that reduce the need to develop every aspect of object detection and deep learning. An object detection framework is based around neural networks, and it is usually built of four components. The first component is the region proposal. During the region proposal, the deep learning model thinks there might be an object in the image and proposes regions of interest (ROI). In these regions, there are added bounding boxes, which are fed to the next layer of the CNN. Bounding boxes are rectangles defined by x and y coordinates that surround the objects in images. The second component of an object detection framework is the feature extraction and the network predictions. At this point of the object detection process, the visual features that are in the bounding boxes are focused on for a closer look. The objects found in the bounding boxes' visual features are then classified so that after this step, there are several proposals for classified objects. The third component is the non-maximum suppression (NMS). NMS combines the bounding boxes on top of each other into a single bounding box for every classified object. The fourth and final part of the object detection framework is the evaluation metrics. In the evaluation metrics part, the model receives the metrics to find the quality of the measurements. The most usual metrics are mean average precision (mAP), precision-recall curve, and intersection over union. The mAP is the most important of these metrics. It is calculated by determining the average precision of all measured classes separately and then calculating the mean of all these average precisions. (Elgendy, 2019, 310.; Yohanandan, 2020.)

Object detection models can be divided according to how many stages they need for the detection. Multi-stage detectors usually need two stages for the detection as single-stage detectors need only one. The advantage of multi-stage detectors is the accuracy they provide. However, the multi-stage detectors are too slow for real-time object detection. Single-stage detectors are often several times faster than multi-stage detectors but have had a relatively low object detection accuracy. The introduction and development of single-stage object detection algorithms such as You only look once (YOLO) and Single-shot detector (SSD) have made real-time object detection possible. (Hui, 2018.)

The architecture of the single-stage detectors resembles each other on a level that can be compared to a human upper body. The network starts from the input layer and leads to the backbone component of the network. The backbone is used for feature extraction. As the efficiency of the backbone is critical for object detection performance, it often

consists of a model that has been priorly trained by a known successful deep learning model. The backbone is followed by the neck component. The primary function of the neck is feature extraction. Followed by the neck is the head component. The head is in charge of the object detection as it does both the image classification and the image regression by determining the properties of the bounding boxes. (Anka, 2020.)

2.2.2 YOLO

You Only Look Once (YOLO) is a real-time object detection model used in object recognition. The name You Only Look Once is based on how the algorithm only looks once at an image while many other algorithms need two looks. Technically, YOLO uses only forward propagation for the prediction, while other models might also use backward propagation for the prediction. Forward propagation means that data goes only from the input layer to the output layer, while in backward propagation, the data goes from output layer to input layer. In these multi-stage models with both forward and backward propagation, the first look is for generating the region proposals, the second look for detecting the objects for the proposals. (Redmon, 2018.)

YOLO, a single-stage model, uses a convolutional neural network (CNN) to make its prediction and proposals. In the CNN, the input image is divided by YOLO into S x S grid cells. These grid cells are all individually responsible for the objects. Dividing the grid cells means that each of the cells will predict the bounding boxes, confidence scores, and conditional class probabilities. The bounding boxes with the confidence scores and the class probabilities are combined, and as a result, the correct class labels and bounding boxes are presented. (Periwal, 2020.)

Overall, there is usually a trade-off between speed and accuracy in every deep learning model. YOLO provides fast object detection, but the accuracy often falls short of its competitors. To sum up, the strengths of YOLO are that it is sufficiently fast and accurate for reliable real-time object detection. YOLO is often compared to SSD as they are the most used detectors due to their accuracy, speed, and performance. The main difference between YOLO and SSD is their structure. YOLO architecture is built out of two fully connected layers, while SSD is built out of convolutional layers that are organized from the largest to the smallest size. (Busireddy, 2019.)

2.2.3 SSD

Single-shot detector (SSD) is a neural network model designed for real-time object detection. The strength of SSD is the accuracy it provides. However, compared to YOLO, SSD is usually slower in its object detection process. The slower speed is due to the architecture of the two detectors. (Busireddy, 2019.)

The SSD model architecture is built out of three parts. The first part is the base network that has been pre-trained. The primary function of the base network is to extract feature maps from images. The second part of the SSD model architecture is the multi-scale feature layers. These feature layers are responsible for filtering the data into smaller scales allowing detections to be more flexibly predicted. The third and final part of the SSD model architecture is the non-maximum suppression. The non-maximum suppression filters and eliminates bounding boxes that overlap each other. (ArcGIS Developers, 2019; Elgendy, 2019, 336.)

The SSD model architecture differs from the object detection framework presented in Section 2.2.1. These differences can be mainly explained by the fact that the model architecture presented earlier considers a multi-stage object detection model. The single-stage models have partly eliminated the first component, region proposals from the architecture. (Jordan, 2018.)

2.3 TensorFlow and TensorFlow Lite

TensorFlow is a software library that is often used for Machine Learning and deep learning. It is primarily used for training large datasets that are used in deep learning. TensorFlow is also used for computations on dataflow graphs. To ease and improve the Machine Learning and deep learning modeling and training, TensorFlow uses its own data graph visualizer, Tensorboard. TensorFlow was developed by Google, and it is known for its architectural flexibility as it provides computational benefits across several platforms. (TensorFlow, 2021b.)

Models built for TensorFlow models can be thoughts of as rulebooks for the interpreter on what to do with the data to receive the correct output. TensorFlow models are normally designed to be run on desktop computers with powerful graphics processing units. As machine and deep learning rely on GPU performance, so does TensorFlow. TensorFlow

requires an Nvidia GPU with a relatively recent Cuda architecture to work. Cuda architecture is used for training in most object recognition models. Devices with Cuda architecture have completely different kinds of components and performance than the portable devices that run mobile applications. Consequently, a light, weight-optimized version of TensorFlow called TensorFlow Lite was designed for smaller devices to run the models. (TensorFlow, 2021a.)

TensorFlow Lite is built out of two main components: an interpreter and a converter. The interpreter runs optimized models on lower-powered devices. The converter transforms the TensorFlow models to a form that the interpreter can use. Additionally, the converter improves optimizations and performance. (TensorFlow, 2021a.)

TensorFlow Lite does not currently support training models. The model has to be trained on a computer with more performance than the relatively low-performance end device and then converted to a TensorFlow Lite-file. Alternatively, The TensorFlow models can be trained using Google Colab that provides an external online-based GPU with Cuda architecture. The trained TensorFlow Lite-file is after the conversion sent to the device's interpreter. (TensorFlow, 2021a.)

3 FLUTTER FRAMEWORK

As the previous chapter has presented the theory and methods behind Machine Learning, this chapter concentrates on Flutter, the framework used to develop the application user interface and front end. More specifically, Flutter offers a user interface (UI) Toolkit developed by Google in 2017 but actually released the first stable version in 2018. Flutter is used to develop natively compiling applications to desktop, mobile devices, and the web. However, it is currently mainly used for mobile development. Both Flutter development for desktop and web applications have though been announced by Google to be developed further in 2021. As Flutter uses the same codebase for Android and iOS applications, the application can be developed to both systems using the same code. Flutter can be thought of as a tool that comprises of two parts. The first part is a software development kit (SDK). The SDK makes it possible to use a single codebase with the programming language Dart and compile the code to native machine code. This process enables the code to work both on Android and iOS. The second part of Flutter is a widget/framework library that provides widgets used to build the applications. Widgets can be thought of as UI-building blocks, and they are most often, for example, buttons, text, or containers. (Gaël, 2019.)

Flutter uses the Dart language to build the applications. Dart-language was developed in 2011, and partly because of the rising popularity of Flutter, the language has developed faster in recent years than before. The Dart language was also developed by Google, and therefore, there was a clear connection between Dart and Flutter during the development of Flutter. Dart is a strongly typed object-oriented language and has often been compared to languages such as Java and C#. (Ford, 2019) While looking at the structure of Flutter applications, there is also a great resemblance to JavaScript. Additionally, the code structure for Flutter is relatively simple as the applications do not need data-, style-, or template separation. (Moovx, 2020.)

As seen in Figure 4, the object detection feature of the application developed in this thesis consists out of stateful and stateless widgets. These states are classes and define the interactivity of the widgets in the application. Stateful widgets are widgets that can change due to interaction with the user, and oppositely the stateless widget will not have any changes with user interaction. The third widget class is the state, and it defines the widgets state, and the widgets build() method. Flutter application structure is relatively

simple to create smaller applications and features such as object detection. The later discussed image caption generator shows a bit more developed structure of a Flutter application. The UML diagram of the image caption generator in appendix 1 shows how the applications are built around stateless widgets, stateful widgets, and states. (Flutter, 2020a

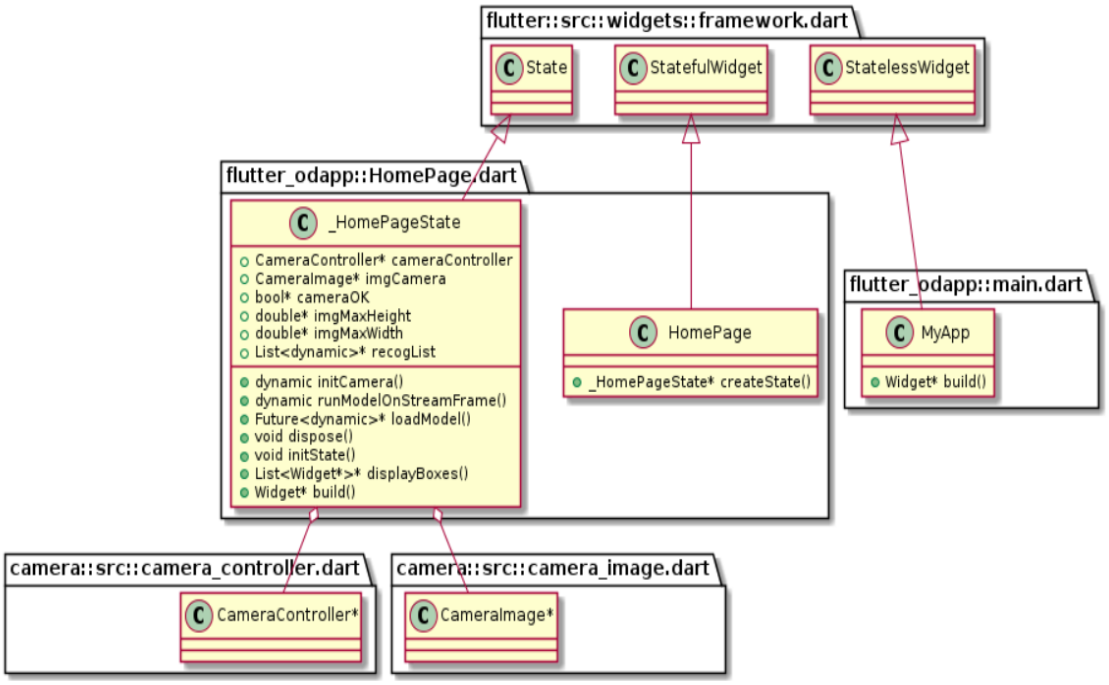

Figure 4. Structure of the object detection-application.

Flutter is strongly typed, which means that it does not have a graphic design system that can be used for designing the layout. Everything that can be seen in the application is typed. This makes designing applications slower to start with than its competitors. However, as everything is controlled through code, the developer has better cross-platform control, and the applications can easily look exactly the same on both Android and iOS. There are minor negative aspects in the UI look that Flutter provides. As Flutter uses its own single codebase, it does not get the exact same iOS look as its competitors, such as React native. Flutter works around this issue by having its own iOS type widgets. These widgets are named Cupertino widgets and work both on Android and iOS. Cupertino widgets let the UI imitate the platform-specific look, so it is nearly impossible to recognize the differences between an app designed with Flutter or its native competitors. What Flutter loses in some design aspects, it wins with its design flexibility. (Flutter, 2021d.)

As Flutter does not have a visual developing interface that shows a layout of the developed application, to ease developing Flutter application, there is a hot reload feature. This means that applications do not need to be rebuilt for the minor changes to show up on an emulator or debugging mobile device. The hot reload feature makes it especially easy to debug on a real-world device, as the minor changes are shown in merely just a couple of seconds. For major changes to be seen in the application, rebuilding the application is necessary. (Flutter, 2021c.)

There are a few alternatives to Flutter when designing cross-platform mobile applications. The main competitor is React Native that is more popular than Flutter. However, React Native was published years earlier than Flutter. React Native also has the advantage in the number of users as it uses a far more popular language, JavaScript. A different kind of competitor also based on JavaScript is Ionic, as it takes the development aspect to a more WebView-based direction. This direction has its benefits, but it makes it nearly impossible to reach high speeds in real-time object detection. As seen in Table 1, Flutter has the edge over its competitors when it comes to designing cross-platform applications needing high speed and performance. However, React and Ionic do have their benefits in other fields. (Demedyuk & Tsybulskyi, 2020; React Native, 2021; Ionic, 2021.)

The reason why Flutter is relatively fast compared to its competitors is that it takes advantage of the Skia graphics library. It makes it possible for Flutter applications to update the application view every time there is a change in the view. In addition to the utilization of Skia-library, the Flutter architecture eliminates the use of a bridge and minimizes the unnecessary data flow going forward and back. The architecture and speed of updating the view make Flutter a top candidate for applications needing real-time features. (Shah, 2020.)

Table 1. Framework comparison between Flutter, React Native, and Ionic.

	Flutter	React Native	Ionic
Language	Dart	JavaScript	JavaScript
Framework	Flutter	React.js	Any or no framework
Application Compilation & Nativity	Compiled native applications	Only partly compiled native applications	Not compiled, hosted web applications inside native applications
Cross-platform compilation for UI components	None for both Android and iOS	Yes, for both Android and iOS	None for both Android and iOS
Platforms	Mobile, web, desktop	Mobile	Mobile, web, desktop
Performance	Nativity of the application gives an advantage in performance	React is only partly compiled to native code, and the use of JavaScript bridge makes the performance less competitive	Wrapping the application causes performance issues
Advertised Framerate	Up to 120 frames per second (Flutter, 2021)	Up to 60 frames per second (React Native, 2021)	Up to 60 frames per second (Ionic, 2021)

A significant advantage for Flutter is that it requires less manual testing than other alternatives. As it functions with Dart, there are many automated testing alternatives available. Flutter also provides automated testing features that can be used at unit,

widget, and integration levels (Shah, 2020). Unit tests test on the smallest level as they can test functions, methods, and classes. Widget tests test the widgets on a component level. Integration tests test the application as a whole or at least a large part of the application. The provided several levels of automated testing make the debugging and testing process relatively simple as the need of manual testing has been cut to minimum. (Flutter, 2020a.)

4 APPLICATION PREREQUISITES AND GOALS

This chapter explains what are the planned technical requirements for the thesis and clarifies what the targets are for creating the finished application successfully. The intent is that all the requirements that are set in this chapter are achieved in the application.

The main goal of the thesis is to create a cross-platform mobile application that would provide both object detection based on image classification and object localization using the phone camera. The application should be fast enough to provide near real-time detection to maximize the usability of the application. To achieve these goals, the architecture should follow the Flutter architecture so that it can take advantage of the optimized system performance. The Flutter-based application is programmed using the Android Studio IDE due to the emulator it provides. Android Studio supports the necessary plugins, and additional packages can be imported from pub.dev, a Dart package hub. The testing of the application is done using both emulators and real-world devices, so the application and specifically the UI works on all tested devices and emulators.

A crucial part of the thesis is to find out if Flutter provides an optimal environment for real-time object detection. The application needs to be able to be seen with a high enough framerate as well it needs to provide a higher framerate than its competitors. The higher framerate should be viewable in testing the application. The framerate is tested on each deep learning model.

The used deep learning models are trained prior to the program calling them. Training the models requires a computer with a reasonably new GPU. The use of a computer with a GPU is not be necessary as the training is done via Google Colab Pro. Google Colab provides an online environment used for Machine Learning. Using Google Colab, the training is done online via an external GPU. This choice of Machine Learning environment also makes the training faster than with a few years old GPU. After training the models on a high-performance GPU, the trained models are converted to formats that can, later on, be used on lower-performance mobile devices. Training on a separate device also results in minimizing the calculations and performance needed by the mobile device.

The real-time object detection feature in the mobile application utilizes two algorithms: YOLO and SSD. Comparing the algorithms on accuracy and speed while using the application should tell if the differences in accuracy and speed are as expected or does the flutter application architecture tighten the gap between the expected speed of Yolo and the accuracy of SSD. The versions of the used algorithms are Tiny-YOLO and SSD Mobilenet. Both versions of the algorithms are designed for mobile devices as they use relatively little CPU power and have a performance according to the mobile device's CPU. Both Tiny-YOLO's and SSD Mobilenet's accuracy and speed do unfortunately differ from the bigger-sized original detectors even though the models are initially trained using the original models. As the application is designed for mobile phones, the size of the application is also critical. The recommended size constraints affect the design of the application. The trained models are trained on a computer and then transferred to the end device.

As the object detection models YOLO and SSD are trained to a limited amount of objects, the detection of objects are limited. The number of detectable objects shall be decided according to the training constraints. As both algorithms are designed for detecting several objects simultaneously, the goal is to be able to recognize the objects in the same image simultaneously. The accuracy of the models are measured using mAP.

To widen the perspective of object detection, an image caption generator is added to the application features. The MAX-image caption generator is trained by IBM using Common objects in context (COCO) 2017-dataset and accessed by contacting a remote Kubernetes container. A Kubernetes container is a way of packaging applications, so they are extracted from the environment they are run in. Therefore, the application is architecturally different in the image caption generator part compared to the object detection part. All in all, the application has three sections of interest: Object detection with YOLO, Object detection with SSD, and Image caption generator.

5 DEVELOPMENT OF THE OBJECT RECOGNITION APPLICATION

This chapter describes the development of the mobile application in this thesis. It presents the choices and solutions used in the execution of application. In addition, the structure and operation of the computer vision application components are presented. As seen in Figure 5, the technical aspect of the object detection part of the thesis is built out of several components that together result in a real-time object detection application. The development of the image caption generator feature for the application is described later in this chapter. Furthermore, the results of the tests on the application are explained.

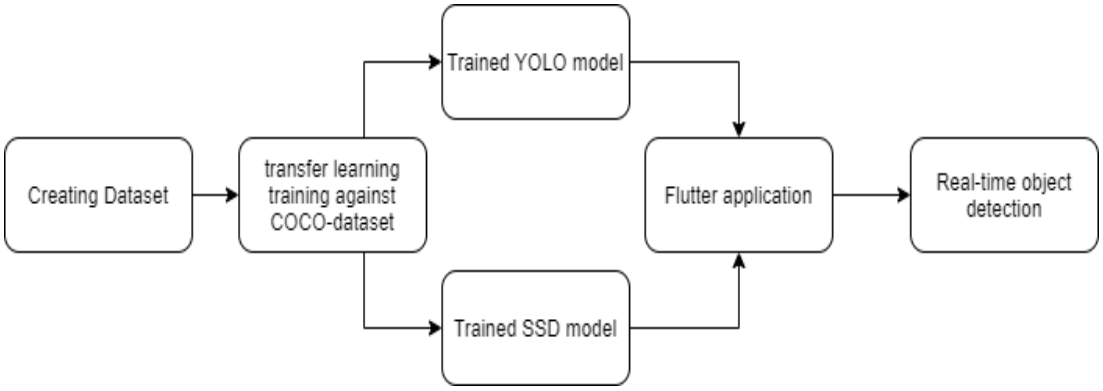

Figure 5. Main components of mobile object detection.

5.1 Flutter-based application development

The UI of the application was built using Flutter, a UI toolkit that is only a couple of years old. A significant benefit of Flutter is that even a single developer can build applications with it relatively fast using the easily modifiable Flutter widgets. The widgets work as the basic building blocks for all applications, as can be further seen with the object recognition application created.

The main reason Flutter was chosen in this project was that it works best for cross-platform systems that need quick calculations as real-time object detection needs. Even though creating the Flutter application is timewise after the training of the deep learning models, it is essential to inform why it is the base for the thesis and why here it is explained before the deep learning. Flutter provides an architecture that should provide

an optimal environment for the previously mentioned deep learning models. Its most significant competitor React Native is more popular for cross-platform development but is usually several times slower than Flutter when processing data. This advantage in performance is largely due to that Flutter follows the same reactive development architecture as its competitors but does not use a JavaScript bridge to access the original equipment manufacturer (OEM) widgets. Hence, As Flutter utilizes custom widgets, it gains choices the OEM do regulate. The object recognition application takes advantage of the fact that Flutter has its own widgets and platform channels. As seen in Figure 6, the application does not need to access the OEM widgets using a bridge, and therefore it has better performance than others using the reactive development architecture. By eliminating the data flow forward and back from a bridge and the widgets, Flutter saves precious time in the dataflow of the application.

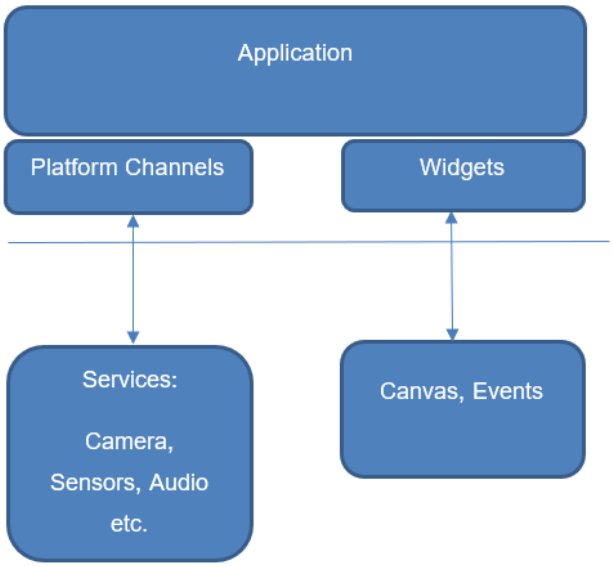

Figure 6. Flutter architecture used in application.

The layout for the application is developed using the layout widgets. As the application layout is built entirely in code and not using any visual tool, creating the layout was a little trial and error. However, Flutter uses an easily understandable hierarchal system to build the visual layout for the applications. By using the hierarchal design and the hot reload function, it is relatively easy to create a clear and functional layout for a smaller screen. If there are any minor layout issues as there were in the development of this application, the debugger shows relatively clearly where the issue is located in code and on the debugging device's screen.

5.2 Transfer learning the dataset

As training any computer vision application, it is possible to build the CNN from the beginning and train the model from scratch. This, however, takes a very long time and often requires weeks or months of testing that would not be possible for this thesis. A more effective method in model training is to download a pre-trained neural network that has already been trained on a large dataset. This way of training is called transfer learning, and in this thesis, the model is trained using transfer learning and then converted and added to the developed object recognition application.

The transfer learning in this thesis utilizes the COCO-2017 dataset. This means that the training of our dataset benefits from the COCO datasets already trained and tested labelled images. The COCO dataset is built out of approximately 120 000 pre-labelled images and it was chosen because it has a large variety of images that have labelled everyday objects. Additionally, the dataset is large enough to have relatively accurate results on the objects. Without using transfer learning, there would have also been a computational issue as the training of the images would have taken several weeks or months, even if there would have been several GPUs available. Even then, the results would most likely been worse than without using transfer learning. Using transfer learning also had the benefit of avoiding overfitting, which is a common issue in deep learning.

The dataset trained to COCO-2017 was initially built using Roboflow, a service that makes dataset building easier and quicker. As seen in Figure 7, by gathering the dataset images into Roboflow, labeling is made relatively simple as the bounding boxes can be set and labelled. Roboflow also allows transforming the dataset to the correct format, so utilizing YOLO and SSD is possible. Utilizing the dataset building using Roboflow, the dataset is divided into training, testing, and validation set.

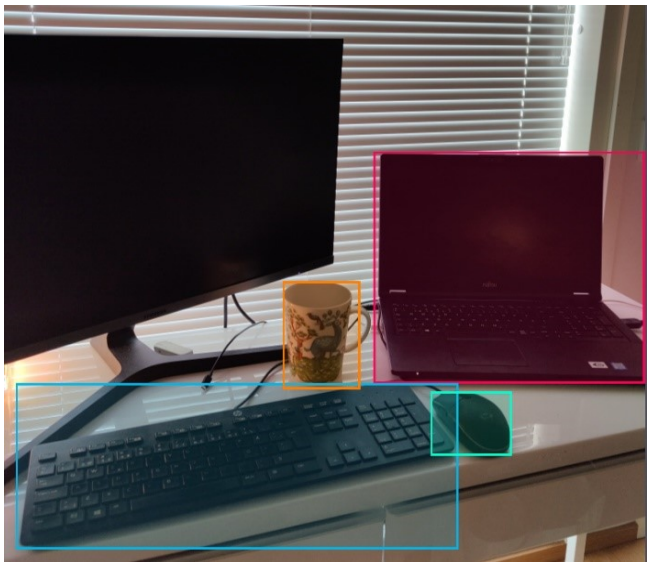

Figure 7. Dataset labeling using Roboflow.

The initial idea of the dataset contents was to build it entirely from everyday images taken from web image searches and label them using Roboflow. This turned to be a too time-consuming task, and by testing the transfer learning with about 200 self-labelled images on objects in offices, it turned out that the mean average precision 0.5 (mAP 0.5) for SSD was around 0.15. This result leads to the conclusion that either the dataset would need to be several times larger or a larger prelabelled dataset could be used.

The necessary choice was to transfer to a prelabelled dataset. The best overall dataset found for the purpose was the PASCAL VOC 2007-dataset, as it contains thousands of everyday images. To utilize the self-labelled images from Roboflow, the images were added to the Pascal VOC 2007-dataset. The final used dataset contains the 200 self-labelled images and the thousands of images from the Pascal VOC 2007-dataset. The transfer to a prelabelled dataset affected the accuracy of the models positively. As can be seen from Figure 8, the transfer to a larger dataset proved to work, as the SSD-based model had the mAP 0.5 or mean average precision 0.5 of almost 70 % at its best. The Yolo-based model reaches nearly the same results in training as the SSD-based model. As shown in Figure 9, at the end of the training with tens of thousands of images, the best result the YOLO-based model reaches is around 65 %. The average loss (avg loss) in both models is around the same 1,34-1,47, proving the model is working. The average loss measures the distance of a single example in the model to the correct prediction on a curve based on all predictions. Hence, the lower this value is, the better.

```
 Last accuracy mAP@0.5 = 63.29 %, best = 69.27 %
 1752: 1.046728, 1.463492 avg loss, 0.000100 rate, 2.667239 seconds, 84096 images, 0.332936 hours left
Loaded: 0.000071 seconds
```

Figure 8. Training results of SSD-based model.

```
 Last accuracy mAP@0.5 = 61.67 %, best = 65.01 %
 1766: 1.588390, 1.346048 avg loss, 0.000100 rate, 2.157091 seconds, 84768 images, 0.336106 hours left
Loaded: 0.000036 seconds
```

Figure 9. Training results of YOLO-based model.

Roboflow also provides Google Colab notebooks that can be used for training the different format datasets using transfer learning. The created dataset is trained using the notebooks after making some changes to them. The transfer learning guarantees much better results in accuracy, as the training of the relatively small, labelled dataset is done to the enormous COCO dataset. The transfer learning also shortens the time used for training. (Roboflow, 2020a; Roboflow, 2020b)

The dataset training utilized the COCO dataset early in training, so the trained model for both Tiny-YOLOv4 and for SSD Mobilenet v2 should have better results than with just without it. The training process for both does resemble each other. However, they have their issues. As a Tiny-YOLO model is not meant for training, the initial training is done using YOLO v4 and then transformed to YOLOv4-Tiny. This causes the biggest issue with Tiny-YOLO. Even though the model has been trained using YOLO, the accuracy and mAP can not be repeated on Tiny-YOLO. After getting the converted Tiny-YOLO file, the file with the model must be transformed to a protocol buffers-file (pb-file). The pb-file is used to store TensorFlow-based models. However, as a pb-file can not directly be used in a mobile application, it has to be converted into a TensorFlow Lite -file that can be directly embedded into the application. A TensorFlow Lite-file is a TensorFlow model-file that can be used on mobile devices. The mAP0.5 of the final converted dropped sharply to averaging around 22%.

The transforming process for the SSD Mobilenet v2 is much simpler compared to YOLO as a trained SSD model can be transformed into a TensorFlow Lite model-file with just one step. However, the model does suffer a bit from the transformation to the TensorFlow Lite model as the architecture of SSD has to go through minor adjustments to function on a mobile platform. After the conversion to a TensorFlow Lite-file, the models mean average precision with a 0,5 threshold for correct detection (mAP0.5) was around 53% on average. This means the conversion resulted in a ten percent drop in accuracy. The trained models can be fed to the application but do need some adjustments to work as planned. These adjustments will be described later in the thesis.

5.3 The structure of the application

Figure 10. Application home screen.

At the beginning of development, the object recognition application was built as two separate applications: Object detection application and image caption generator. Both the object detection and the image caption generator display the different features of mobile object recognition. To make the object recognition application more versatile and consistent for testing purposes, both object recognition features are combined into a single application. As seen in Figure 10, the application home screen icons offer the promised object recognition alternatives in a single application. The home screen icons are divided into two rows. The first row provides the object detection alternatives and the second row the image captioning alternatives. Combining the features into a single application makes the testing results more comparable to each other. In addition, the application structure was developed to present how Flutter makes these features as effective as possible.

5.3.1 Object detection

Creating the object detection feature in the object recognition application starts with building the front-end for the application. Building a layout using Flutter, where it is possible to call for the deep learning models by pushing buttons, does not need a particularly complex layout structure. As earlier mentioned, the Flutter-based widgets provide building blocks where the deep learning components can be inserted into.

The most important part of application functionality is to connect the camera and gallery to the deep learning models. Both the SSD- and YOLO-based deep learning models do work without changes necessary to the application itself but to give the model the best chance to work. However, some optimization is necessary to get the same level results gotten during the training and testing of the models. The models need optimizations depending on what UI and end-platform they will be used on. A major optimization is to normalize the models so they will work on mobile devices. In this case, the SSD-model needs normalization for it to work. As the model is trained and converted, the color values are changed to values between -1 and 1. However, the model is designed to have values between 0 and 255. Hence, a normalization of the color values is necessary. This optimization is especially necessary due to the use of Flutter as the UI. As Flutter is in charge of updating every pixel, the normalization of making 127,5 the center value especially important. If the value would not be set to 127,5 the model would see the image overly bright and could not detect or recognize all the objects. This means the object detection feature in the mobile application would have poor accuracy and miss objects that it can recognize normally. As seen in program 1, the Tflite.detectObjectOnFrame-method is called, and it runs the camera stream through the trained model. Without the normalization value correctly set, the image would be much brighter than normal, and the object detection would give a bad result.

Program 1. runModelOnStreamFrame-function for object detection using SSD.

```
runModelOnStreamFrame() async
  {
    imgMaxHeight = imgCamera.height + 0.0;  //default set 1280
    imgMaxWidth = imgCamera.width + 0.0;  // set 720
    recogList = await Tflite.detectObjectOnFrame(
      bytesList: imgCamera.planes.map((plane) {
        return plane.bytes;
      }).toList(),

      model: "SSDMobileNet",
      imageHeight: imgCamera.height,
      imageWidth: imgCamera.width,
                                              Program continues.
```

```
    imageMean: 127.5, //input normalization(model trained -1 - 1)
    imageStd: 127.5,  // Otherwise -> too bright -> lower accuracy
    numResultsPerClass: 1,  // Gives only one prediction
    threshold: 0.45, // Tested on 0.4 - 0.6
  );
```

The code structure of the YOLO feature of the application resembles the SSD feature. However, the YOLO-part does not need the same normalization as the SSD. As the model is trained with the non-negative values 0,0-255,0, the pixel color value stays the same from the original training device to the mobile end-device. Some minor adjustments to the model are necessary. The YOLO model automatically rotates the camera by 90 degrees. The rotation is easily fixed by changing the model-values in Tflite.detectObjectOnFrame. A larger issue with the YOLO-Tiny-model is with both the accuracy and the speed it provides. The accuracy of the trained YOLO model is not good. Unfortunately, the accuracy can not be helped, as many of the original YOLO-model features are not available in the converted tflite-based model. As the model does not recognize the detected object with a high probability, the detection threshold must be set low to 0,15. Compared to the SSD model threshold at 0,45, it is clear that SSD is a clear winner in accuracy. However, the YOLO-based object detection feature works sufficiently in the application with objects that are easy to recognize.

The YOLO model provides approximately 200-250 frames per second (FPS) and therefore creates problems with the application keeping in the pace of the model. Without any changes in the application properties, there rises an issue with scheduling. Mobile phones can currently show a maximum of 120 FPS on their screens. Therefore, the model can not display 200-250 FPS. By minor changes in asynchronous scheduling, the issue with scheduling is fixed, and the model works as planned.

A crucial task of the thesis was to implement object detection successfully into a Flutter mobile application, as shown in Figure 11, and test if Flutter provides an optimal environment for it. A secondary part of the task was to test if the Flutter architecture provides a better environment for computer vision and object detection than its competitors, React Native and Ionic. During testing, Flutter offers a frame rate up to 120 FPS for object detection, as seen in Figure 12 and Figure 13. Testing the object detection application frame rate separately on SSD, the results varied from 60 FPS to 120 FPS with a couple of minor quick slips down to 40 FPS. While testing the Tiny YOLO-based feature, Flutter did provide a more stable stream of frame rate between 60 and 120 frames per second. Newer mobile phones do currently have a 120 FPS maximum framerate for their screens, as did the tested device. Therefore, it can be stated that both SSD and YOLO are successful in providing near maximum frame rate. During further testing of the UI average frame rate, the average is between the values of 3,5 ms and 4,5 ms. This means that, on average, Flutter would provide clearly more than the 120 frames per second. Both React.JS and Ionic have a promised maximum framerate of 60 FPS, and therefore the results are clear. (React Native, 2021; Ionic, 2021) Flutter offers the best object detection speeds and mobile applications if the CPU provides over 60 FPS for the model, as the application is capable of displaying predictions over 60 times per second. However, these high frame rates are currently scarce on mobile devices and therefore often irrelevant.

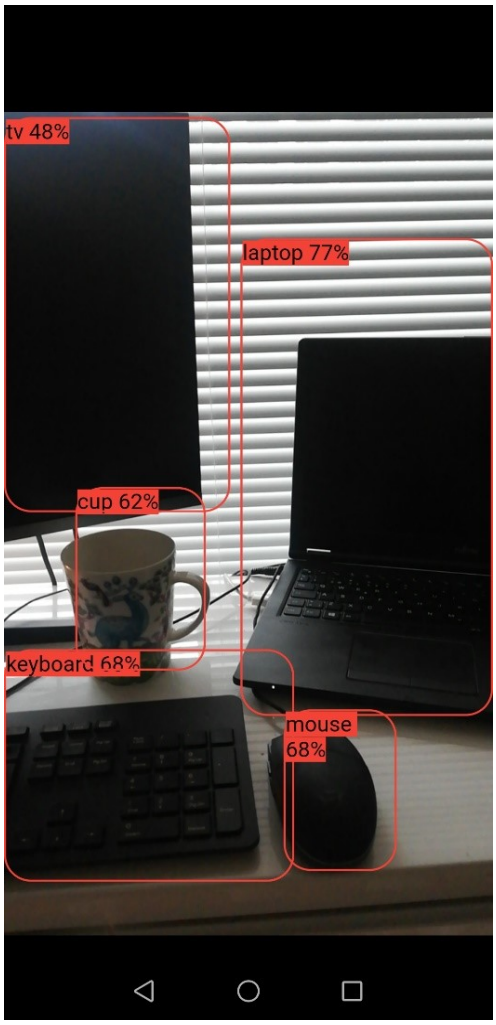

Figure 11. Working object detection using SSD-detector.

The results of the object detection application were as expected but did provide some surprises. As expected, the SSD-based model performed better in both accuracy and mAP. It provided around 50-70% accuracy when detecting objects correctly. The recognition threshold was set to 45% to improve the viewing of the results. As the reliability of the prediction varied every frame per prediction, the bounding box showing

the result would disappear when the result would dip just below 50%. Setting the threshold to 45% made the viewing more pleasurable, but at the same time, a bit more unreliable. However, as a goal was to create a mobile application that would show the differences in accuracy and speed between the SSD model and YOLO model, a decision was made to have the threshold 5% lower than normal.

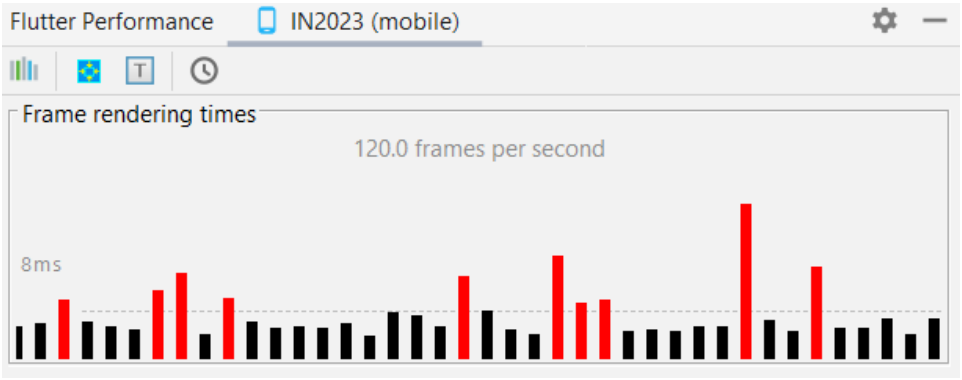

Figure 12. Object detection using SSD Mobilenet (40.0-120.0 FPS).

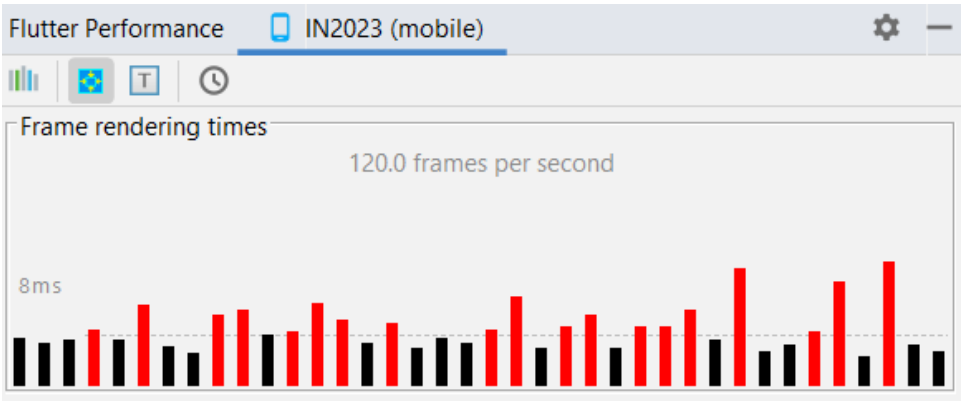

Figure 13. Object detection using Tiny-YOLO (60.0-120.0 FPS).

5.3.2 Image caption generator

The image caption generator has a more complex structure than the object detection parts as it actually has both an application front-end and a back-end. The application shows three ways to use the image caption generator. These three ways or features are caption predictions on camera stream, caption predictions on an image taken using the application, and caption predictions on an image from the mobile phone gallery. All three features can be seen in Figure 14, and are based on the most crucial part of the program, the camera stream feature that predicts the situation through a video stream of the camera.

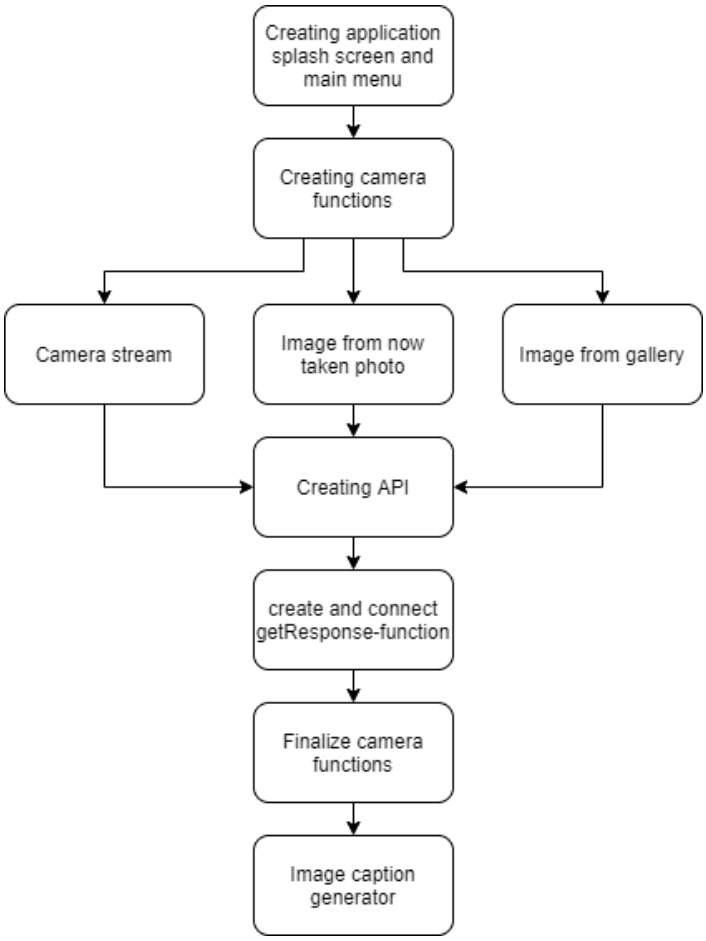

Figure 14. Flowchart of the image caption generator-part of the application.

The application shows the three best predictions on screen. The current version of the application displays an updated prediction every four seconds. This is due to the fact that if the application showed the prediction faster, there would be two issues. The first issue is that as there are three predictions on screen, the reading time of the predictions would

be too fast. The second issue is that these four seconds give the application more than enough time to perform all its functions flawlessly. If the time for reading would not be necessary, the four seconds could be changed to one second and still work. While testing the image caption generator, there should be a relatively significant difference between the speed of the object detection and the image caption generator.

The second feature of the image caption generator is that it will predict the situation from a picture in the mobile phone's gallery. However, as it would not benefit anyone, the prediction from an image from the gallery does not change every four seconds. The third feature of the application is that it will also predict the situation from a picture just taken using the application. After creating these features based on the mobile phone's camera, the image caption model itself needs to be connected to the application.

From a more technical aspect, the prediction works by capturing images from the camera feed and storing them into the mobile phone's local storage. From the local storage, the images are then sent using an HTTP POST request to the image caption generator model in the getResponse-function. This means that the images are passed to the model, and the generated caption predictions are sent back and shown on the screen of the device.

To display the effectiveness of the image caption prediction, the application offers three alternative choices as it predicts the situation. These prediction alternatives often resemble each other quite closely, as can be seen in Figure 15. Compared to the object detection model, the application does not show the probability of the prediction. The choice of not showing the probability was made primarily because the model has a significant amount of prediction alternatives, and the probabilities are relatively small. As shown in Figure 15, the application accurately predicts the situation in front of the mobile phone camera.

Figure 15. Screen capture of the working image caption generator.

The back end of the application is based on utilizing the IBM-Max Image Caption Generator model container that is called over a server. Creating the model of an effective image caption generator by oneself is an enormous feat. As a working image caption generator needs hundreds of thousands of labelled pictures and even more distinct objects, even the work collecting, creating, and generating the dataset for the model would take a long time. The used IBM Max Image caption generator consists of over 200 000 labelled images and has more than 1 500 000 distinct labelled objects (Singh, 2020). As well as creating and labeling the dataset for a long time, the training of such a dataset would take months for average GPUs. The Max image caption generator has been trained at IBM using supercomputers. Therefore, it is understandable that the image caption generator used in the thesis is prebuilt and not developed for the use of the application.

The Max image caption generator used is stored in the IBM cloud. To gain contact with the model in the cloud. the model of the image caption generator has been modified into being stored in a Kubernetes container. By contacting the created container, the application can ask the model what is viewed in the camera view and what captions it predicts. As the IBM cloud does only support a free Kubernetes container for a limited amount of time, the completed application does contact a Kubernetes container upheld by IBM and not the previously made container. This container has the same Max image caption generator model as the previously created and used container. Therefore, it is important to be reminded of the differences in the features in the application. The main

difference to the object detection feature in the application, the image caption feature contacts the container using the POST method and receives the captions from the model while the object detection feature is entirely local.

Flutter UI provides approximately 24.0 frames during the use of the image caption generator, as seen in Figure 16. During a half dozen measurements the frame rendering times were between 24.0 to 30.0 frames per second. As the frame rate for real-time detection is usually thought of being around these framerates and up, the Flutter UI provides an excellent environment for real-time object detection with an external model. However, the limitation of the image caption generator is not only the frame rate but the processing time built out of application performance and sending and receiving them from the model in the online container. Flutter provides both the frame rate and the performance for the image caption generator to work successfully.

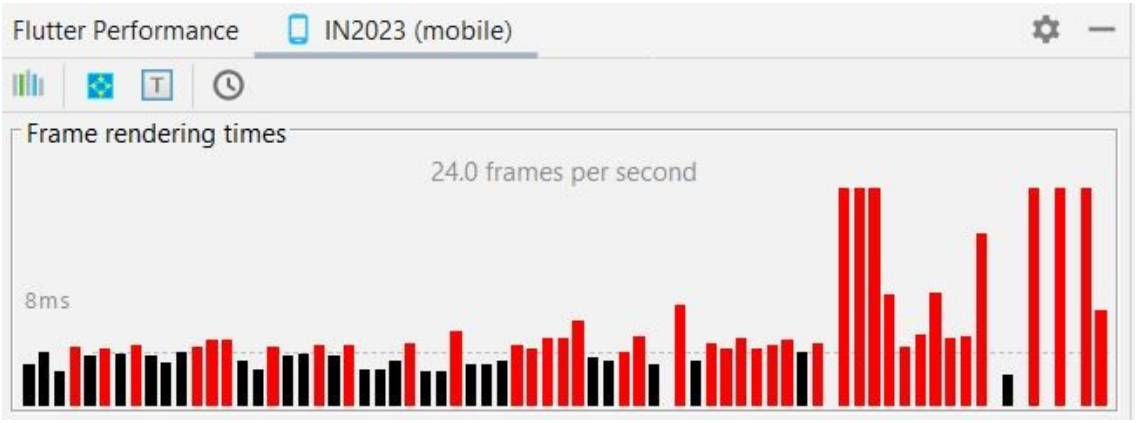

Figure 16. Image caption generator (24.0-30.0 FPS).

6 CONCLUSION

The main goal of the thesis was to create a Flutter-based mobile application with object detection features as well as to research the suitability of the Flutter UI for real-time object detection. To research the suitability, two types of computer vision features were created and comprised into a mobile application. The object detection feature presents the local deep learning model's performance on the Flutter UI, whereas the image caption generator presents an approach using an external model. The differences in Flutter performance for the local and external features are noticeable. However, they both provide real-time object recognition.

The Flutter UI provided an optimal environment for computer vision features in the developed mobile application. The Flutter architecture presented also a stable near maximum frame rate for both a Tiny-Yolo-based and SSD Mobilenet-based object detection. Furthermore, Flutter also triumphed in frame rate over its most significant competitors with a twofold frame rate. The testing of the image caption generator displayed the lowered frame rate of 24 frames per second. As the goal was to create a near real-time system, the performance of the image caption generator was a success.

Another significant question in the thesis was to find out how the two object detection models perform. The differences between SSD- and YOLO-based object detection is viewable in the application as an accuracy difference. As a result of training the dataset and converting the original models into a TensorFlow Lite-based mode, the differences in accuracy are amplified. The SSD-based model provides more consistent and accurate object predictions on screen than the YOLO-based object detection model.

The final version of the object recognition application has some areas of improvement that were not implemented. Visually, the bounding boxes in the object detection application are slightly too large on the top side. The initial thought was that the bounding box size is the wrong size due to the screen resolution, screen size, or the TensorFlow Lite detection on frame. However, changing the properties of the screen resolution, screen size, and TensorFlow Lite detection on frame-values did not fix the issue. More improvements could also have been made to the dataset used in object detection. Due to the time constraints concerning the dataset labeling, an optimal dataset would have been self-labelled. However, labeling the thousands of images would have taken weeks or months to do. The decision to gain a more accurate model over a completely self-

made dataset was hard but correct. As a result of having to change to training a tested dataset, the application functions according to the expectations. Currently the final improvement to the application would be testing the application on iOS. The current version of the application works only on Android but could function on iOS with a few changes in the dart-project properties. As Flutter provides the option of easily creating cross-platform mobile applications, the testing on iOS would be a logical step in development for the application.

All in all, both the object detection and image caption generator features function as planned. The differences between SSD- and YOLO-based object detection are viewable in the application as a clear difference in accuracy. Furthermore, the image caption generator shows how an external and sizeable deep learning model can be utilized successfully in mobile applications via a cloud container. As importantly, the image caption generator works near real-time by predicting the image caption with high accuracy. Both computer vision features function optimally due to the Flutter-based architecture and structure. Flutter provides high performance and reliability in both computer vision tasks featured in the application and should be considered a top candidate when building cross-platform applications needing real-time features.

REFERENCES

Altexsoft, (2018). *The Good and the Bad of Flutter App Development*. [online] Altexsoft. Available at: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-flutter-app-development/ [Accessed 11 Mar. 2021]

Anka, A., (2020). *YOLO v4: Optimal Speed & Accuracy for object detection*. [online] Medium. Available at: https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50#:~:text=YOLO%20v4%20achieves%20state%2Dof%2Dthe%2Dart%20results%20(,FPS%20on%20a%20V100%20GPU. [Accessed 7 April 2021].

ArcGIS Developers. (2019). *How single-shot detector (SSD) works? | ArcGIS for Developers*. [online] Available at: https://developers.arcgis.com/python/guide/how-ssd-works/ [Accessed 19 March 2021].

Built In, (2020). *What is Artificial Intelligence? How Does AI Work? | Built In*. [online] Available at: https://builtin.com/artificial-intelligence [Accessed 19 March 2021].

Busireddy, C., (2019). *is YOLO really better than SSD?*. [online] Linkedin.com. Available at: https://www.linkedin.com/pulse/yolo-really-better-than-ssd-chandrakala-busireddy [Accessed 3 April 2021].

Brownlee, J., (2017). *What is the Difference Between Test and Validation Datasets?*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/difference-test-validation-datasets/ [Accessed 20 March 2021].

Brownlee, J., (2019a). *A Gentle Introduction to Object Recognition With Deep Learning*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/object-recognition-with-deep-learning/ [Accessed 13 March 2021].

Brownlee, J., (2019b). *How Do Convolutional Layers Work in Deep Learning Neural Networks?*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/ [Accessed 14 March 2021].

Brownlee, J., (2019c). *How to Avoid Overfitting in Deep Learning Neural Networks*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/ [Accessed 3 April 2021].

Council of Europe. (2020). *History of Artificial Intelligence*. [online] Council of Europe. Available at: https://www.coe.int/en/web/artificial-intelligence/history-of-ai [Accessed 24 March 2021].

Demedyuk, I. & Tsybulskyi, N., (2020). *Flutter vs Native vs React-Native: Examining Performance*. [online] Inveritasoft. Available at: https://inveritasoft.com/blog/flutter-vs-native-vs-react-native-examining-performance [Accessed 11 March 2021].

Dsouza, J., (2020). *What is a GPU and do you need one in Deep Learning?*. [online] Towards Data Science. Available at: https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d [Accessed 26 March 2021].

Elgendy, M., (2019). *Deep Learning for Vision Systems*. . [online] Flutter. Available at Early access: Manning Publications

Flutter. (2020a). *Testing Flutter apps*. [online] Flutter. Available at: https://flutter.dev/docs/testing [Accessed 19 April 2021]

Flutter. (2020b). *Adding interactivity to your Flutter app*. [online] Flutter. Available at: https://flutter.dev/docs/development/ui/interactive#:~:text=A%20widget%20is%20either%20stateful,are%20examples%20of%20stateless%20widgets.[Accessed 17 April 2021]

Flutter. (2021a). *Flutter architectural overview*. [online] Flutter. Available at: https://flutter.dev/docs/resources/architectural-overview [Accessed 01 April 2021]

Flutter. (2021b). *Flutter performance profiling*. [online] Flutter. Available at: https://flutter.dev/docs/perf/rendering/ui-performance#:~:text=Flutter%20aims%20to%20provide%2060,UI%20doesn't%20render%20smoothly. [Accessed 05 April 2021]

Flutter. (2021c). *Hot reload*. [online] Flutter. Available at: https://flutter.dev/docs/development/tools/hot-reload [Accessed 02 April 2021]

Flutter. (2021d). *Platform-specific behaviors and adaptations*. [online] Flutter. Available at: https://flutter.dev/docs/resources/platform-adaptations [Accessed 04 April 2021]

Ford, S., (2019). *The Dart Language: When Java and C# Aren't Sharp Enough.* [online] Toptal. Available at: https://www.toptal.com/dart/dartlang-guide-for-csharp-java-devs [Accessed 23 March 2021]

Fritz AI. (2020). *Object Detection Guide*. [online] Fritz AI. Available at: Accessed 19.03.2021 https://www.fritz.ai/object-detection/#part-basics [Accessed 19 March 2021]

Gaël, T., (2019). *What is Flutter and Why You Should Learn it in 2020.* [online] freeCodeCamp. Available at: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/ [Accessed 23 March 2021]

Géron, A., (2017). *Hands-On Machine Learning with Scikit-Learn & TensorFlow – Concepts, tools, and techniques to build intelligent systems.* Sebastopol: O'Reilly Media, Inc..

Hui, J., (2018). *SSD object detection: Single Shot MultiBox Detector for real-time processing.* [online] Medium. Available at: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06 [Accessed 20 March 2021]

Iriondo, R., (2018). *Machine Learning (ML) vs. Artificial Intelligence (AI) — Crucial Differences*. [online] Towards AI. Available at: https://pub.towardsai.net/differences-between-ai-and-machine-learning-and-why-it-matters-1255b182fc6 [Accessed 23 March 2021]

Ionic,. (2021). *Fast apps. Out-of-the-box*. [online] Ionic Framework. Available at: https://ionicframework.com/ [Accessed 02 April 2021]

Jordan, J., (2018). *An overview of object detection: one-stage methods.* [online] Jeremy Jordan. Available at: https://www.jeremyjordan.me/object-detection-one-stage/ [Accessed 03 April 2021]

Kavlakoglu, E., (2020). *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the difference?* [online] IBM. Available at: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks [Accessed 16 March 2021]

Moovx, (2020). *What's best in 2020? Flutter vs React Native.* [online] Moovx. Available at: https://moovx.mobi/whats-best-in-2020-flutter-vs-react-native/ [Accessed 23 March 2021]

Open Data Science, (2018). *Overview of the YOLO Object Detection Algorithm*. [online] Open Data Science. Available at: https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0 [Accessed 14 March 2021]

Periwal, S., (2020). *Real-Time Object Detection with YOLO.* [online] LatentView Analytics. Available at: https://www.latentview.com/blog/real-time-object-detection-with-yolo/ [Accessed 20 March 2021]

React Native, (2021). *Performance Overview.* [online] React Native. Available at: https://reactnative.dev/docs/performance [Accessed 02 April 2021]

Redmon, J., (2018). *YOLO: Real-Time Object Detection.* [online] Pjreddie. Available at: https://pjreddie.com/darknet/yolo/ [Accessed 03 April 2021]

Roboflow. (2020a). *Roboflow-TensorFlow-object-detection-mobilenet-colab.ipynb*. [online] Google Colab. Available at: https://colab.research.google.com/drive/1wTMIrJhYsQdq_u7ROOkf0Lu_fsX5Mu8a#scrollTo=YjtCbLF2i0wl [Accessed 26 March 2021]

Roboflow. (2020b). *Mobile Object Detection.ipynb*. [online] Google Colab. Available at: https://colab.research.google.com/drive/1sJHfrz2DhfEziEZJav_ueWJtTHnlooH3#scrollTo=zbniFj-eSimL [Accessed 27 March 2021]

Shah, H., (2020). *React Native vs Flutter: Choose The Best Framework For Your Next Project.* [online] Simform. Available at: https://www.simform.com/react-native-vs-flutter/#testing [Accessed 23 March 2021]

Singh, A. & Bhadani, R., (2020). *Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter*. Birmingham, United Kingdom, Packt Publishing Ltd.

TensorFlow, (2021a). *TensorFlow Lite guide*. [online] TensorFlow. Available at: https://www.TensorFlow.org/lite/guide [Accessed 26 March 2021]

TensorFlow., (2021b). *TensorFlow 2.4.1* [online] PyPi. Available at: https://pypi.org/project/TensorFlow/ [Accessed 02 April 2021]

Theseus. (2021). *Theseus*. [online] Theseus. Available at: https://www.theseus.fi/ [Accessed 26 March 2021]

Yiu, T., (2019). *Understanding Neural Networks*. [online] Towards Data Science. Available at: https://towardsdatascience.com/understanding-neural-networks-19020b758230 [Accessed 25 March 2021]

Yohanandan, S., (2020). *mAP (mean Average Precision) might confuse you*!. [online] Towards Data Science. Available at: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2 [Accessed 04 April 2021]

Image caption generator architecture

(Cut in half due to size)