Chi Nguyen

Utilizing Google's Machine Learning Kit in Developing Android Application

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

15 April 2024

Abstract

Author:	Chi Nguyen
Title:	Utilizing Google's Machine Learning Kit in Developing Android Application
Number of Pages:	45 pages + 1 appendices
Date:	15 April 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Mobile Solutions
Supervisors:	Peter Hjort, Senior Lecturer

In recent years, machine learning technology can be found in almost every of area of life. As the need for machine learning integration into mobile application has become increasingly relevant, a question arises: Do developers need an in-depth understanding of machine learning to utilize Google ML Kit in their application? This thesis aimed to answer this question.

The thesis began by reviewing the key concepts and techniques of machine learning to identify the common process of utilizing machine learning in mobile application development. Furthermore, the thesis evaluates existing frameworks that are available for developers to utilize machine learning solutions in mobile application development.

As the result of this thesis, three basic mobile applications for Android platform were developed using Google's ML Kit framework. One application's main functionality is detecting human pose from an image. Another application can classify yoga poses via a live camera feed. The third application allows users to translate text from English to Finnish.

The significant finding of this thesis was that development process with Google ML Kit is relatively simple. The basic understanding of machine learning is often good enough as Google ML Kit provides detailed guidelines.

Keywords: ML foundation, Google ML Kit, pose detection, yoga pose classification

Contents

List of Abbreviations and Key Concepts

AI:	Artificial Intelligence
Android:	Mobile operating system based on the Linux kernel, developed by Google, and mainly used for mobile devices
API:	Application Programming Interface
Colab:	Google Colaboratory
CSV:	Comma Separated Values
Dataset:	Collection of data, can be treated as a single unit
IDE:	Integrated Development Environment
iOS:	iPhone Operating System
ML:	Machine Learning
OS:	Operating System
Prediction:	What comes out of machine learning model
SDK:	Software Development Kit
URI:	Uniform Resource Identifier

1 Introduction

In the past ten years, machine learning (ML) has increased its popularity. Businesses are leveraging ML to gain insights into customer behaviour, optimize sale strategies, and enhance user experiences. Industries such as manufacturing are using ML to streamline production, improve cost efficiency, forecast demand, and ensure quality control. From healthcare, finance, security, logistics to marketing, ML is driving innovation and reshaping the way people approach problems. (O'Donnellan 2024.)

Integrating artificial intelligence (AI) into mobile applications offers various opportunities to improve their functionality, enrich users' experiences, and provide personalized interactions for the user. The popularity of smartphones makes them optimal platforms for deploying ML's algorithms, enabling tasks such as image recognition, speech comprehension, and text analysis. Applications such as Google Translate, Netflix, YouTube, Facebook have already applied ML in functionalities such as "Recommendation" or "People you might know". (Zaheer 2023.)

Additionally, mobile devices, with their multiple sensors and powerful computing capabilities, provide an ideal ground for implementation of ML applications. The challenge lies in developing efficient and lightweight applications that do not compromise devices' performance or drain battery life. Fortunately, the advancement in mobile technology and the availability of tools such as Google's ML Kit, TensorFlow Lite have made it easier for developers to create ML based applications. These tools offer pre-trained models optimized for mobile devices, reducing the barrier to entry and empowering developers to create innovative solutions. (Google 2018.)

This thesis focuses on developing Android mobile applications utilizing Google's ML Kit framework. It seeks answer to the question: Do developers need an in-

depth understanding of machine learning when utilize Google's ML Kit framework in their mobile application development process?

The first chapter is devoted to introducing the readers to machine learning. An overview of machine learning and its concepts are explained in detail here. The following chapters introduce popular machine learning frameworks as well as other tools and resources that are used in building the demo applications. The thesis then goes over the process of developing mobile applications using Google's ML Kit framework. The final chapter concludes the thesis with its finding and discussion.

2 Background

2.1 Machine learning: an overview

Machine learning is considered as a subset of AI. It was originally defined as the "field of study that gives computers the ability to learn without explicitly being programmed" in the 1950s by Arthur Samuel. The goal of ML is to create computer models that behave like humans and can perform actions without detailed instructions. (Brown 2021.)

ML learning programming is different from traditional programming and is used to solve problems that cannot be addressed with traditional programming. In traditional programming, developers provide logic and step-by-step instructions on how to perform a specific task for the application and produce a pre-determined outcome when the program was executed correctly. However, in ML programming, algorithms are used to analyse the input data and make predictions or decisions without explicitly guidance. The difference between traditional programming and machine learning programming is depicted in figure 1. (Brown 2021.)

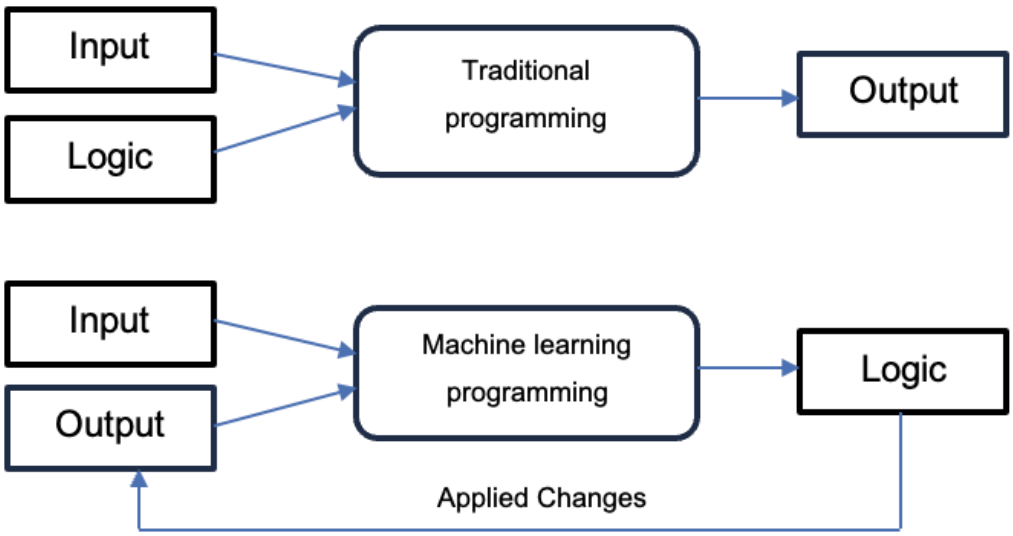


Figure 1. Difference between Traditional programming and ML programming. Based on Brown (2021).

Chollet (2021: 5) stated that “machine learning discovers rules for executing a data processing task”. The process involves input data points, examples of the expected output, and a measurement to access whether the algorithms is performing well. Essentially, machine learning is “an automatic search process for data transformations that produce useful representations of some data, guided by some feedback signals”, which can be used to solve the task at hand. (Chollet 2021: 6.)

Chollet (2021: 432) emphasized the role of data in ML by stated that ML develops models “purely from exposure to training data”. According to Brown (2021), data can be of any types: images, text, sound, video, numerical data and so on. The type of data is decided based on the features of the applications. The quality and quantity of data significantly influence the performance of ML program.

ML Gopalakrishnan and Avinash (2018: 9) suggested to use machine learning programs in the following scenarios:

- very complex tasks that are difficult to program
- very complex tasks that deal with huge amount of data
- adapting to changes in environment and data.

Most current real-world machine learning based applications are: “optical character recognition, email spam filtering, image classification, computer vision, speech recognition, machine translation, group segmentation and clustering, generation of synthetic data, anomaly detection, cybercrime prevention, credit card fraud detection, internet fraud detection, time series prediction, natural language processing, board game and video game playing, document classification, recommender systems, search, robotics, online advertising, sentiment analysis, DNA sequencing, financial market analysis, information retrieval, question answering, and healthcare decision making.” (Patel 2019.)

2.2 Categories of Machine Learning

As highlighted in figure 2, ML is usually categorized into three broad categories: supervised, unsupervised, reinforcement learning. The type of machine learning can be chosen based on the problems that need to be solved. ML is also associated with several other AI subfields such as natural language processing, neural networks, and deep learning. (Brown 2021.)

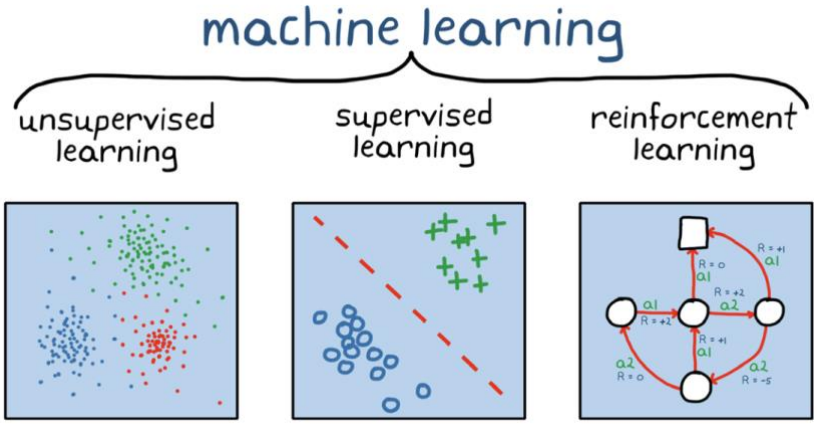


Figure 2. ML categories. Copied from MathWorks (2020).

2.2.1 Supervised learning

Supervising learning is a technique that uses labelled datasets to train algorithms to predict outcomes and recognize patterns. Supervised learning has two

phrases: training and prediction. In the first phrase, a collection of labelled data is fed to an algorithm. The algorithm is designed to look at the data over and over, notice the common characteristics and learn to detect patterns. In the prediction phrase, the trained algorithm can make prediction on new unlabelled data. A basic example for supervised learning is a model that can recognize what animal it is from different in images as seen in figure 3.

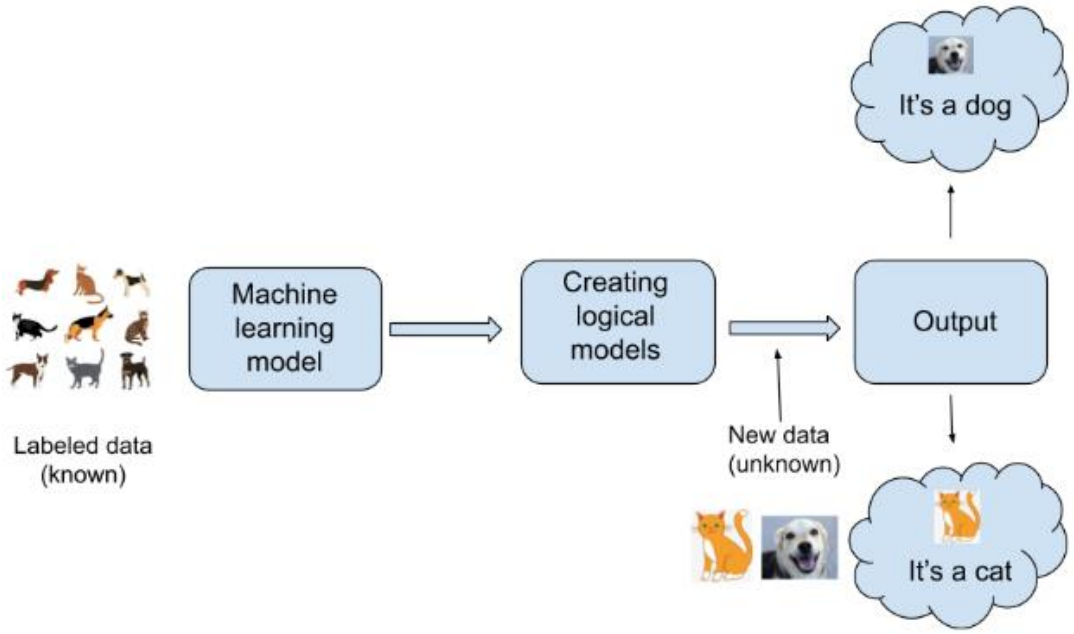


Figure 3. Supervised learning example. Copied from Banoula (2023).

This method of learning is based on repetition. By repeating its execution and making adjustments until a certain set of conditions is met, the algorithms learn to recognize specific traits of labelled data. This helps them to get better at providing the output of new data correctly. By refining the process over and over, the model keeps getting better at predicting while minimizing errors. (Google 2024.)

Supervised learning's problems are usually felled under two primary categories: classification and regression. In classification problem, the goal is to assign input data sets into predefined classes/ categories. Regression problem is more focused on forecast numerical values by finding a relationship between the

dependent variable and the independent variable. (Dutt, Sandramouli and Das 2018.)

2.2.2 Unsupervised learning

As the name suggests, the model learns by itself without human intervention in unsupervised learning. The objective of unsupervised learning is not to predict an outcome based on input data like supervised learning, but rather detect hidden patterns and find similarities from unlabelled data sets. Figure 4 below describes how unsupervised learning works. (Gopalakrishnan and Avinash 2018: 52)

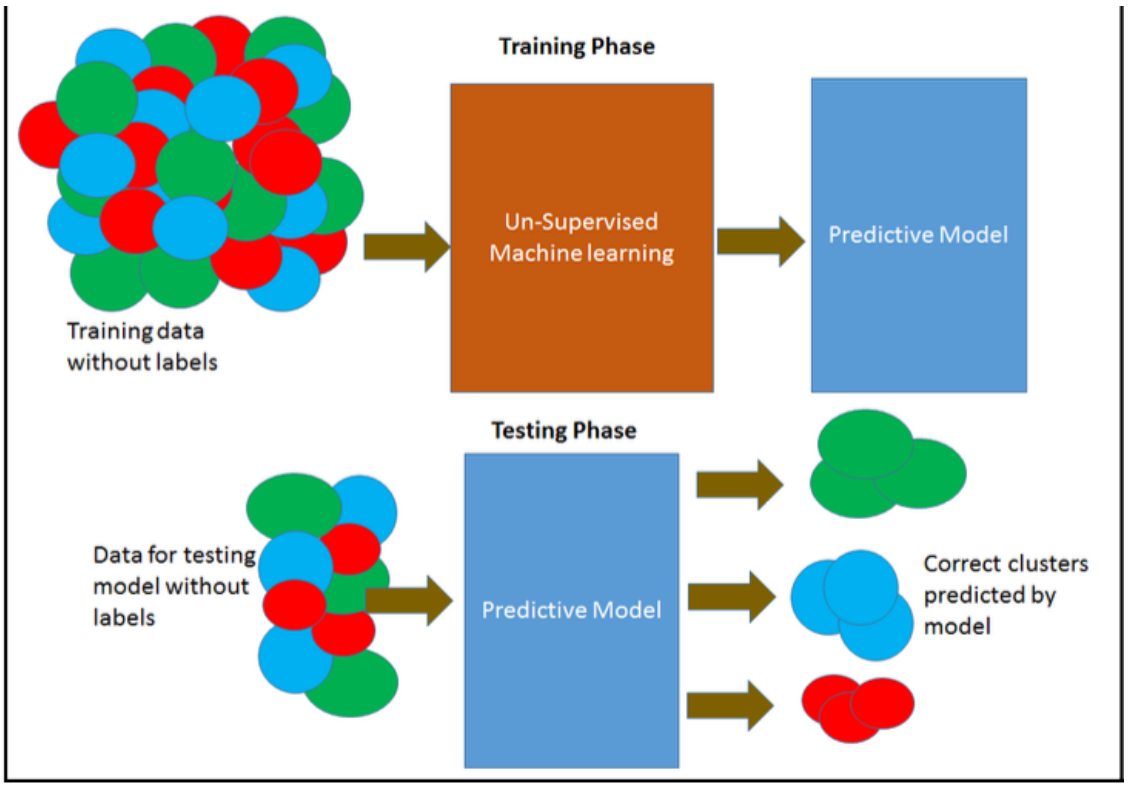


Figure 4. What is supervised learning. Copied from Gopalakrishnan and Venkateswarlu (2018: 20).

Unsupervised learning can be further grouped into types: clustering and association. Clustering is the main type of unsupervised learning. Its goal is to group unlabelled data into groups/clusters based on similarities or patterns.

Association, on the other hand, looks for the dependency between data elements so their relationship can be map accordingly. (Banoula 2023.)

Unsupervised learning thrives in scenarios where it needs to identify patterns in a large volume of data. Examples of real-world application are recommendation engines, object recognition, image detection and classification, and many more. (IBM 2024.)

2.2.3 Reinforcement learning

Reinforcement learning has gradually become a large research area in machine learning since its first introduction in 1979. It is a “computational approach to learning from interaction” and “is much more focused on goal-directed learning from interaction than other approaches to machine learning”. (Sutton & Barto 2018: 1.)

Reinforcement learning focuses on learning how to make decisions in dynamic environment rather than make predictions or finding patterns. Even though a reinforcement learning model does not utilize pre-labelled data sets, it does not fit into the category of unsupervised learning either. This is because the primary goal of reinforcement learning is to learn the optimal course of actions - map situations to actions – in order to maximize a numerical reward signal, rather than trying to uncover structurers. (Sutton & Barto 2018 :1.)

The decision-making process in reinforcement learning, described in Figure 5, involves “interaction between an active decision-making agent and its environment, within which the agent seeks to achieve a goal despite uncertainty about its environment”. Re (Sutton & Barto 2018: 5.)

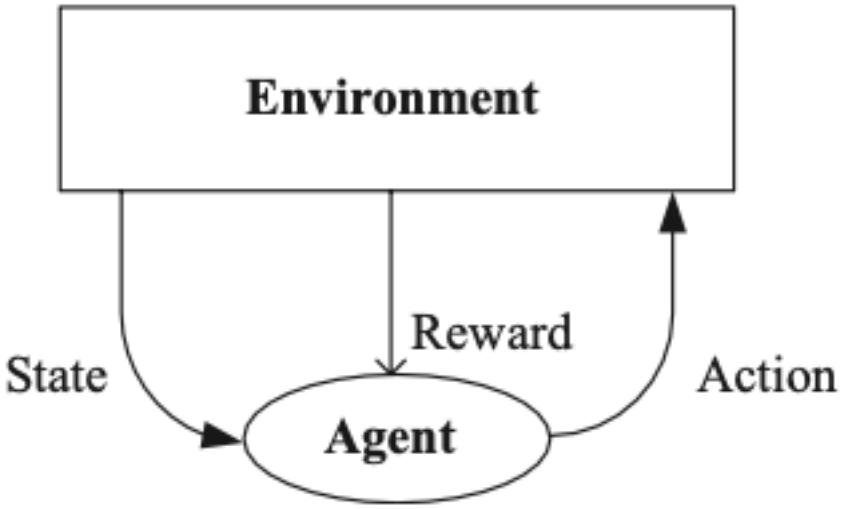


Figure 5. Reinforcement learning process. Copied from Alpaydin (2021).

Reinforcement learning process is a complex, iterative process. The first step is observation. The reinforcement learning agent observes the current state of its environment. After that, it must decide what action to take based on a policy. The concept of policy in reinforcement learning is defined as “a mapping from perceived states of the environment to actions to be taken when in those states”. Once an action is selected, the agent then executes the action, and the environment transition to a new state. Following the execution of the action, the agent receives “a single number called the reward” from the environment. The goal of the agent is “to maximize the total reward it receives over the long run”. The policy might be altered based on the reward signal. This cycle of steps continues until the learning agent has reached its goal or until a certain number of steps have been taken. (Sutton & Barto 2018: 6.)

2.3 Transfer learning

In traditional machine learning, models are trained from scratch even if the tasks are similar. This approach can be time-consuming and expensive from the use of computational resources. Transfer learning is a machine learning technique which addresses those issues by leveraging the learned knowledge to enhance

the learning of new tasks. Figure 6 illustrates the connection between transfer learning and machine learning. (GeeksforGeeks 2023.)

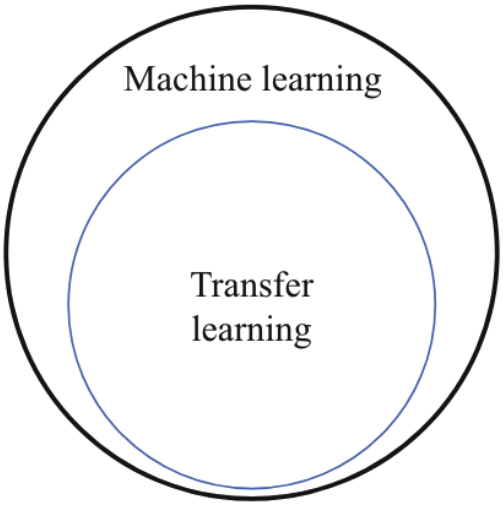


Figure 6. The relationship between transfer learning and machine learning. Copied from Wang and Chen (2023).

Wang and Chen (2023: 5) stated that “transfer learning aims to solve the new problem by leveraging the similarity of data (task or models) between the old problem and the new one to perform knowledge (experience, rules, etc.) transfer”. In general, transfer learning can be used in various scenarios, including computer vision, natural language processing, activity recognition, indoor location, and human-computer interaction. (Wang & Chen 2023: 17.)

2.4 Machine learning on mobile devices

Computers are evolving rapidly, and their form factors are expanding. Previously confined to office, nowadays computers can be seen on home desks, in people’s laps or pockets, and even on wrists. Nearly every adult carries a device, and smartphones are checked by their owner at least 50 times a day. Given this accessibility and usage, it makes sense to run ML models directly on mobile devices (Ng 2018.)

According to Singh and Bhadani (2020), mobile manufactures have been consistently enhancing their hardware to accommodate the computational

demands on mobile devices. Kirin 970 SoC from Huawei, A11 Bionic from Apple, Exynos 9810 from Samsung are examples of dedicated chips enabling on-device machine learning tasks.

Gopalakrishnan and Venkateswarlu (2018: 23) stated that running ML models on mobile devices offers several advantages. The model is stored locally on the device, allowing it to function offline without requiring a network connection. By eliminating the need for data transmission, the network bandwidth cost is reduced. Additionally, on-device ML provides faster performance and enhances privacy.

2.5 Implement machine learning in mobile applications

Gopalakrishnan and Venkateswarlu (2018: 10) presented four main activities of the universal workflow of a machine learning: define problem, prepare the data, build the model, and deploy the model. Figure 7 provides a quick overview of the development process.

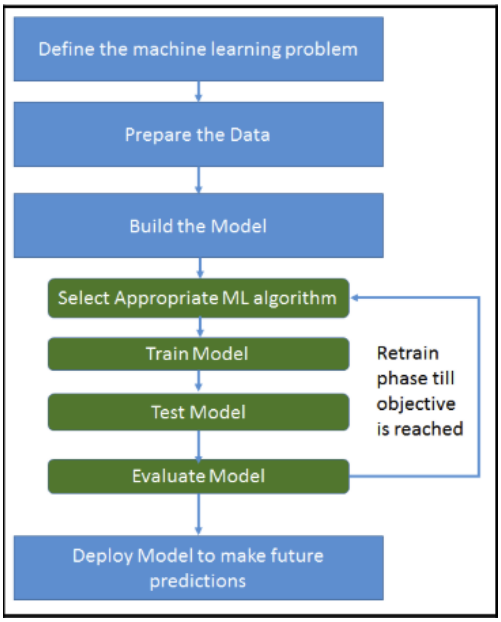


Figure 7. The machine learning process. Copied from Gopalakrishnan and Venkateswarlu (2018: 10).

Gopalakrishnan and Venkateswarlu (2018: 25) concluded that using a trained model to solve a problem is much easier than training a new model. In the mobile

application development process, there are some decisions need to be made in early stages by answering the following questions:

- Is there a need for training and creating a custom model or can it be sufficient with prebuilt model?
- If the need for training a custom model, where the training process can be done: desktop, cloud or on the mobile device itself?
- Once the model is available, is it going to be stored on the device or on the cloud?

3 Popular ML frameworks for mobile development

3.1 Apple: CoreML

Core ML is a framework developed by Apple, which can be used to integrate ML models directly into applications across Apple's operating systems. It was introduced in the Worldwide Developers Conferences 2017 and is tightly integrated with XCode. (Apple Inc 2024.)

Core ML provides supports for machine learning features such as analysing images, processing text, converting audio to text, identifying sounds in audio. Sample use cases of the framework are depicted in the following figure. (Apple Inc 2024.)

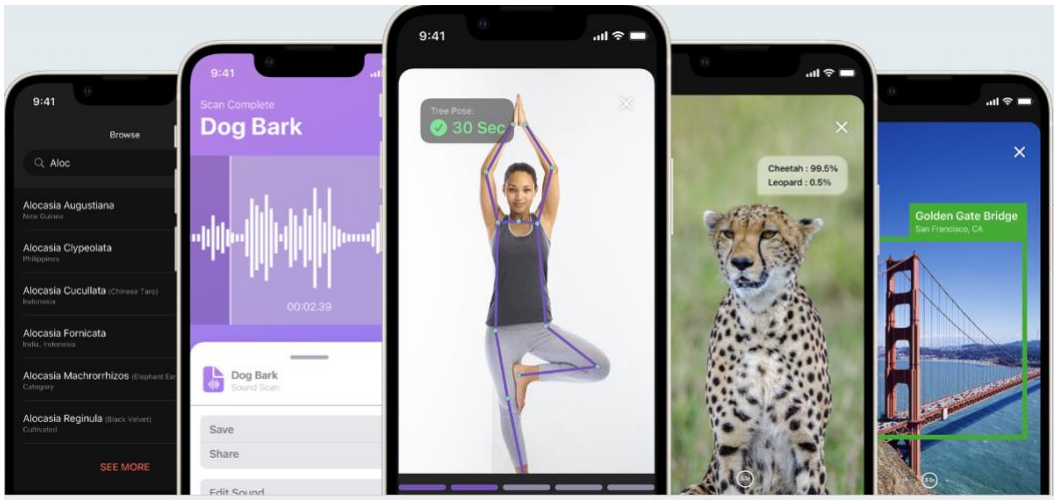


Figure 8. Framework Core ML. Copied from Apple Inc (2024).

By utilizing Apple hardware such as CPU, GPU, and Neural Engine, Core ML reduces the battery consumption, memory usages and optimizes devices' performance. Moreover, they can run on the device without a network connection which helps the models can learn user's behaviour and protects user's privacy. (Apple Inc 2024.)

Supported mobile platforms:

- iOS

3.2 Google: TensorFlow Lite

TensorFlow Lite is a powerful framework developed by Google. It aims to help developers run ML models on mobile, embedded and edge devices. TensorFlow Lite, which consists of five high-level components as shown in figure 9, currently supports multiple platforms such as Android and iOS devices, embedded Linux, and microcontrollers. (TensorFlow 2024.)

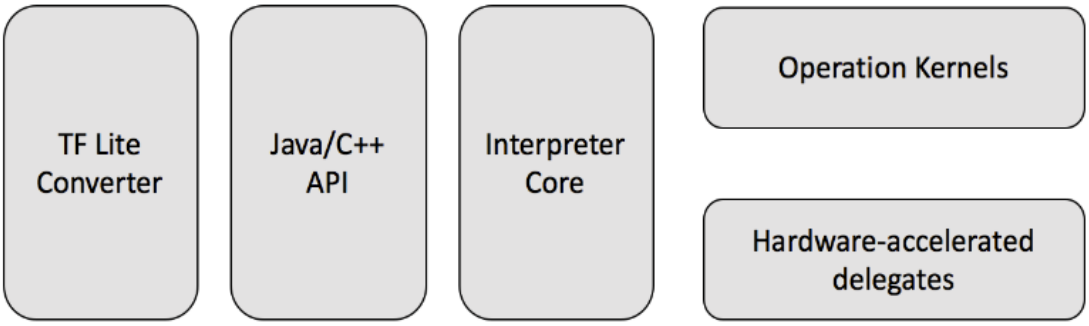


Figure 9. TensorFlow Lite Architecture. Copied from Ng (2018).

TensorFlow Lite is built on the foundation of TensorFlow, Google's open-sourced machine learning platform. It provides a lightweight solution with optimization for on-device machine learning by addressing 5 key constraints: latency, privacy, connectivity, size, and power consumption. (TensorFlow 2024).

TensorFlow Lite can improve performance using hardware acceleration and model optimization. With its multiple delegates, developers can choose

applicable delegates based on two major criteria: the targeted platform (Android or iOS) and the model type. (TensorFlow 2024)

Supported mobile platforms:

- iOS
- Android

3.3 Google: ML Kit

ML Kit is a comprehensive set of machine learning tools developed by Google. It was first introduced in 2018 with a tightly integrated with Firebase version. Nowadays, developers can easily incorporate machine learning features into both Android and iOS applications using ML Kit with a new standalone ML Kit SDK version. It supports the development process for both native and cross-platform applications. (Prins and Hu 2020.)

As illustrated below in figure 10, ML Kit offers a range of ready-to-use Application Programming Interfaces (APIs). ML Kit's APIs are divided into 2 segments: Vision and Natural Language as in figure 10. These APIs offers to solve common challenges such as image labelling, text recognition, face detection, barcode scanning, and many more. (Google 2024.)

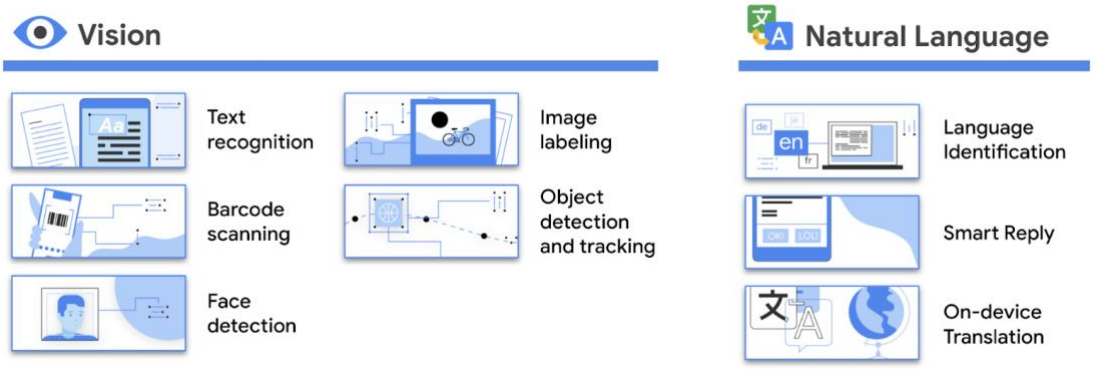


Figure 10. A Pictures of ML Kit APIs. Copied from Google Developers Blog (2020).

Google-trained ML models, which are called base models and exists in ML Kit by default, run entirely on device, which give users speed and privacy advantages. In addition, ML Kit's APIs also allow developer to replace the base models with customized models. (Google 2024.)

Supported mobile platforms:

- iOS
- Android

3.4 Android: AICore

Android AICore is the latest system service designed in Android 14 that enables running foundation models directly on-device. Foundation models such as Gemini were trained with diverse range of data to support multiple ML use cases. Gemini Nano is the default size for mobile devices which used in Android AICore. (Burke 2023)

According to Burke (2023), Android AICore is built on Android's Private Computer Core. It is isolated from network via open-source API. Android AICore handles model management, runtimes, safety features according to the architecture shown in figure 11.

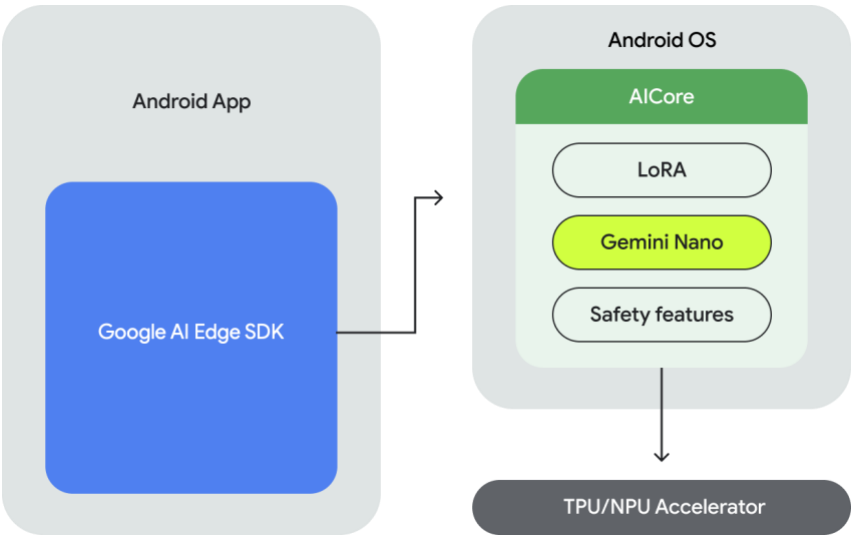


Figure 11. Android AICore Architecture. Copied from Google (2024).

Supported mobile platform:

- Android

3.5 Google: MediaPipe

MediaPipe is a cross-platform and open-sourced framework developed by Google for building pipelines to provide ML solutions. It is a combination of several existing tools from Google: MediaPipe Solutions, TensorFlow Lite Task Library, and TensorFlow Lite Model Maker. MediaPipe tasks can run on mobile devices, workstations, and servers, and support mobile GPU acceleration (Google 2023.)

Solution	Android	Web	Python	iOS	Customize model
Object detection	●	●	●	●	●
Image classification	●	●	●	●	●
Image segmentation	●	●	●		
Interactive segmentation	●	●	●		
Hand landmark detection	●	●	●	●	
Gesture recognition	●	●	●	●	●
Image embedding	●	●	●		
Face detection	●	●	●	●	
Face landmark detection	●	●	●		
Face stylization	●	●	●		●
Pose landmark detection	●	●	●		
Image generation	●				●
Text classification	●	●	●	●	●
Text embedding	●	●	●		
Language detector	●	●	●		
Audio classification	●	●	●		

Figure 12. MediaPipe available Solutions. Copied from Google (2024).

MediaPipe's available solutions support tasks that belong to three main domains: vision, text, and audio. Figure 12 lists all currently available task with their supported platforms. As MediaPipe Solutions is an evolving product, it may have

some limitations such as limited support, incompatible issues with earlier versions, and availability may change without notice. (Google 2024.)

According to Google (2023), each MediaPipe solution is equipped with the following libraries and resources to provide its core functionality:

- MediaPipe Task: A cross-platform API and library for deploying solutions.
- MediaPipe model: Pre-trained, ready-to-run model use with solution.
- MediaPipe Model Maker: A tool to customize model if needed.
- MediaPipe Studio: A tool to visualize, evaluate, and benchmark solutions in the browser.

Supported mobile platform:

- Android
- iOS (availability is shown in figure 12)

4 Applications' specification

4.1 Applications' concept

One goal of this thesis is to explore the utilization of Google's ML Kit in the development of Android applications. To familiarize with ML Kit's range of features, the thesis chooses to develop demo applications utilizing two distinct solutions offered by ML Kit: one from the Vision APIs (ML Kit Pose Detection) and another from the Natural Language APIs (ML Kit Translation). The outcome is three mobile applications that run only on Android devices and can perform the machine learning task associated with the selected ML Kit solutions.

4.2 Applications' functionalities

Two applications' main functionality is posture detection. With the utilization of Google ML Kit's Pose Detection, users would be able to detect a human's body pose either from a static image or a camera live feed.

In the first demo application, users should be able to choose a picture from their image gallery as the input and choose to process the image by click a button. It was developed with the base model provided by ML Kit's Pose Detection. The result is displayed to the users as a drawing of a human body's pose.

In the second demo application, input is a camera live feed. For the second demo application, a custom-trained model is integrated in the development environment. The application enables users to classify a yoga poses from a live camera feed. The result would be shown to the users in real time with the drawing of a human's body pose and a yoga pose's name.

The third demo application, the main function is to translate text from one language to another language. The application allows users to enter text and with the ML Kit's on-device translation API, it will display the required translated result.

4.3 Technical requirements

The target platform of both applications is Android. Operating system of the Android devices is required to be Android SDK 28 or above, which means that all devices running Android 9 and above can install the demo applications. The reason behind this specification is because of ML Kit framework.

4.4 Tools used in the project

4.4.1 Android Studio

Android Studio is Google's official Integrated Development Environment (IDE) for Android development application. It was built on IntelliJ IDEA, anther IDE from Jet Brains. Android Studio is easy to install on various operating systems such as Windows, Mac, Linux, and ChromeOS. Android Studio is a comprehensive tool that is more than just an intelligent code editor. It also includes an Android Virtual Device (AVD) Manager, which can create multiple emulators. This feature enables developers to test their applications without physical device. Additionally,

Android Studio provides a suite of tools to debug, write test and more. (Google 2024.)

4.4.2 ML Kit's Pose Detection

According to Google (2023), the ML Kit Pose Detection is a lightweight solution used for detecting human bodies pose in real time from a static image or continuous video. It produces a full-body 33 key skeletal landmark points which is depicted in figure 13.

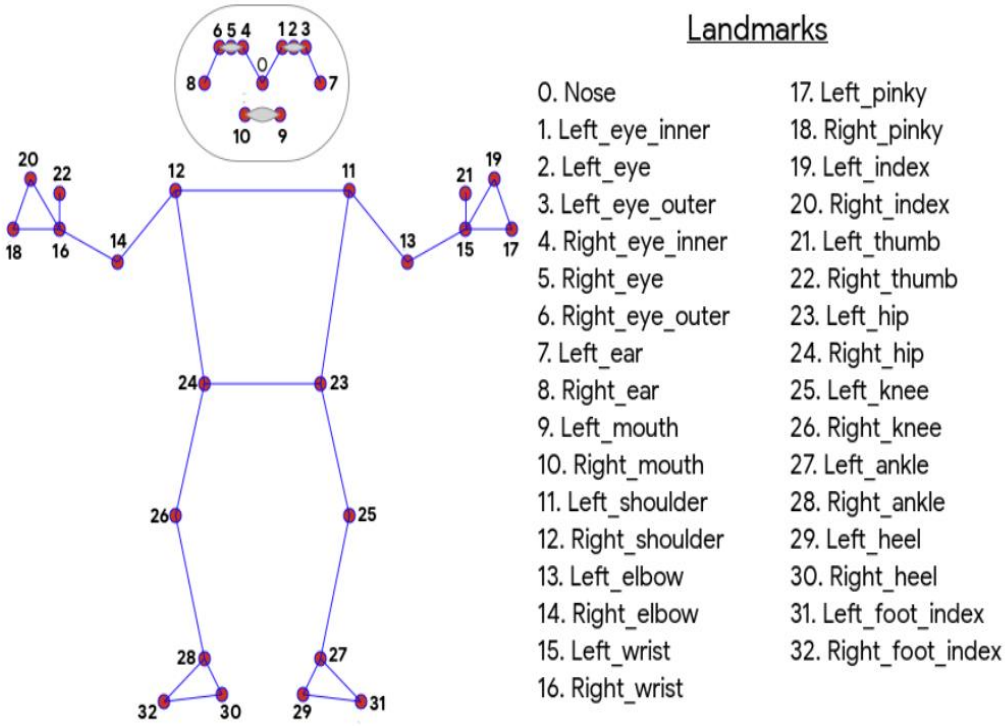


Figure 13. ML Kit Pose Detection Landmarks. Copied from Google (2023).

As shown in figure 13, the Pose landmark detection uses a series of models: pose detection model and pose landmarker model. The pose detection model is responsible for detecting the presence of human bodies within an image frame. The pose landmarker model locates landmarks on the bodies. (Google 2024.)

The model, developed by Google (2021), is designed for real-time performance on a diverse range of mobile devices. However, it is sensitive to factors such as

face position, scale, and orientation within the input image. In addition, the model can’t handle the following scenarios:

- There are multiple people present in the same image.
- The distance between the body and the camera is more than 14 feet/4 meters.
- The head is not within the camera’s frame.

There are two optimized SDKs for pose detection: pose-detection and pose-detection-accurate. They have the same implementation process. The only key differences are the app size and performance. Pose-detection-accurate will provides better performance at bigger app size. (Google 2024.)

4.4.3 ML Kit’s Translation

Google (2024) states that ML Kit’s Translation solution supports more than 50 languages. It uses the same models used by the Google’s translate application in offline mode. Since ML Kit’s Translation model run on device, the translate process is quick.

It is crucial to noted that the ML Kit Translation is “intended for casual and simple translations”. English is used as a translation intermediate since ML Kit Translation’s models are trained for translating to and from English. Thus, depending on the languages being translated, the quality of these translation varies. (Google 2024.)

4.4.4 Google Colaboratory

Google Colaboratory, often referred to as “Colab”, is a free cloud service based on Jupyter Notebooks. It provides users an interactive environment to write and execute Python in the browser. One of the most significant features of Colab is its zero-configuration requirement. It means that a user can start working with projects without any setup process. Additionally, it provides free access to GPUs. This feature is beneficial for projects that require high computational power.

Moreover, Collab notebooks are easy to share and collaborating on. (Google 2024)

4.5 Scope and limitation

The primary purpose of the demo applications is to explore the level of machine learning knowledge required by developers when utilizing Google ML Kit in Android application development. Thus, the focus lies on implementing the core machine learning functionality within the apps.

It's important to note that user experiences and user interfaces were not prioritized and thus were intentionally left minimal for these applications. It means that the applications may not provide an optimal user experience. Users might find the interfaces less intuitive or visually appealing.

Additionally, application architecture and optimization were not the primary concern during the development. Hence, the applications may not be efficient in terms of resources utilization. Scalability or maintainability might be compromised.

5 Implementation

This chapter describes the process of developing the demo applications in detail. The first part provides a detailed implementation of how to create the pose detect applications. The text translate application implementation is explained in the second part. The final part discusses the implementation process.

5.1 Pose detection Android applications

5.1.1 Creating initial application

The goal was to develop mobile applications for Android devices that could detect pose from static images or a camera live feed. To develop an Android

application, the project uses Android Studio Giraffe version 2022.3.1 Patch 1 which was built on August 2023. Kotlin is the main language used in the development process.

The initial application is created using **Empty Views Activity** template from the category **Phone and Tablet** in Android Studio. It has only one main activity which opens when the user launches the application. Figure 14 shows the **MainActivity** file which was auto generated in the template project.

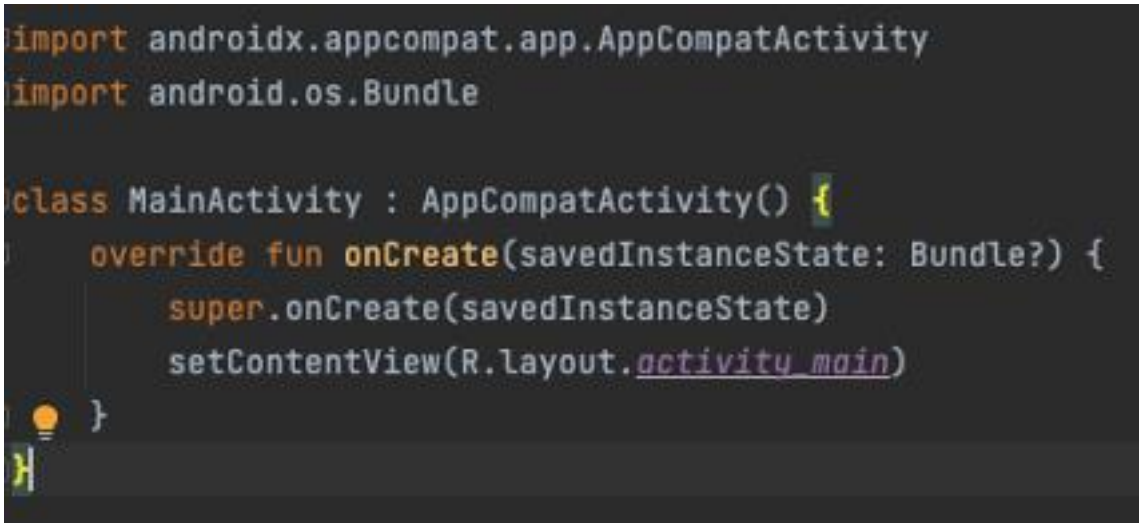


Figure 14. Screenshot of the MainActivity file of the template project.

In order to use Google ML Kit Pose detection, in the project-level **build.gradle** file, Google's Maven repository must be added in both **buildscript** and **allprojects** sections. Then, in the module's app-level gradle file, the dependencies for the ML Kit Android libraries can be added as in the figure 15. (Google 2024.)

```
// Pose detection
implementation ("com.google.mlkit:pose-detection:18.0.0-beta4")
implementation ("com.google.mlkit:pose-detection-accurate:18.0.0-beta4")
```

Figure 15. Screenshot of ML Kit Pose detection injection in the project's Gradle file.

To access the mobile devices' photo gallery or its camera, the application must request correct permissions from the user. To implement this, the necessary permissions must first be defined in the Android project's Manifest file as in figure 16, which can be found in the project under the name AndroidManifest.xml.

```
<!--    Needed permissions for this application-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Figure 16. Screenshot of permissions in AndroidManifest.xml file.

5.1.2 Create a custom trained model

To build a yoga pose classifier that recognized specific yoga pose is quite a challenging task for developers. Fortunately, ML Kit Framework provides the MediaPipe Colab (Appendix 1) with step-by-step instructions on how to create a custom pose classifier. (Google 2022.)

The process started with collecting image samples. The thesis uses a yoga poses' dataset found in Kaggle (Appendix 1). The dataset can be used to train five well-known yoga poses: tree, warrior two, downdog, goddess and plank. According to Google (2022), for the classification to work well, each pose class should have about 100 images. The dataset's file structure is illustrated in the figure below.

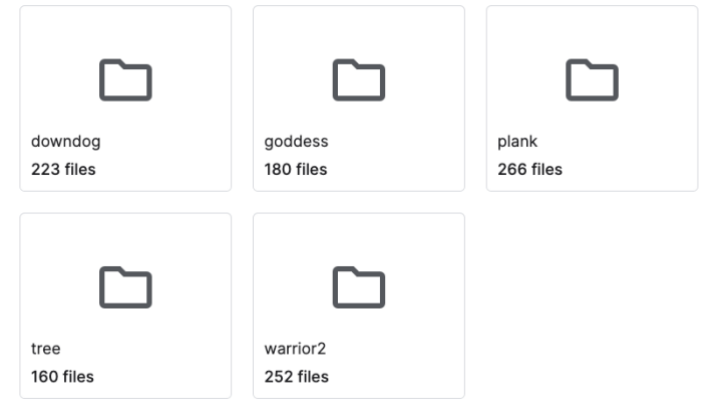


Figure 17. Folder structure of train images in dataset. Copied from Pandit (2020).

Developers need to upload the images from the dataset into Colab. The instruction for folder structure and naming convention can be found on **Step 1: Build classifier**. Figure 18 shows the code block in the Colab.

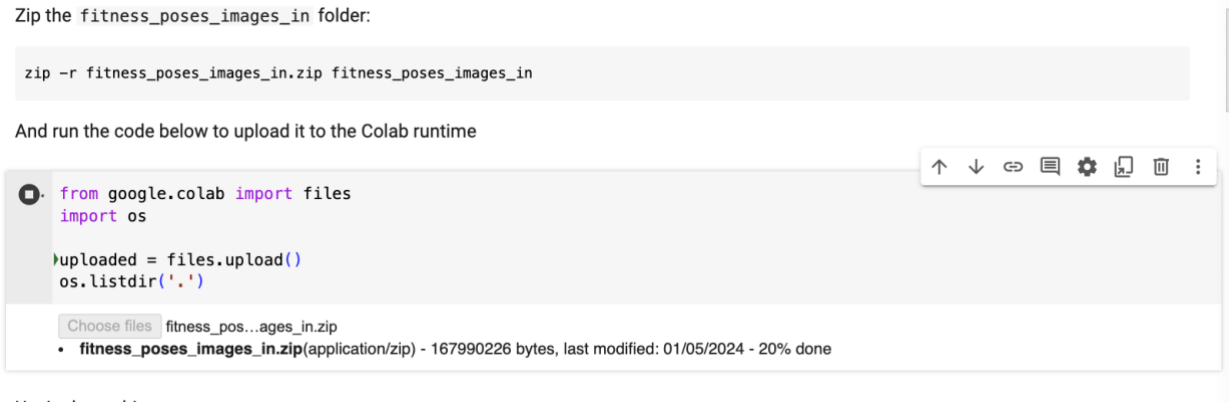


Figure 18. Screenshot for code block of uploading training images to Colab.

Next step is to run pose detection on the sample images. This produces a set of pose landmark that later will be used for training. After that, the MediaPipe Colab is used to access the code for classifier and train the model. The result is a Comma Separated Values (CSV) file that can be integrated with ML Kit Android application. Figure 16 shows the Colab code block that, when it is executed, a CSV file will be created and downloaded to the computer. (Google 2022.)

Dump for the App

Dump filtered poses to CSV and download it.

Please check this guide on how to use this CSV in the ML Kit sample app.

```
[36] import csv
     import os
     import numpy as np


     def dump_for_the_app():
       pose_samples_folder = 'yoga_poses_csvs_out'
       pose_samples_csv_path = 'fitness_poses_csvs_out.csv'
       file_extension = 'csv'
       file_separator = ','

       # Each file in the folder represents one pose class.
       file_names = [name for name in os.listdir(pose_samples_folder) if name.endswith(file_extension)]

       with open(pose_samples_csv_path, 'w') as csv_out:
         csv_out_writer = csv.writer(csv_out, delimiter=file_separator, quoting=csv.QUOTE_MINIMAL)
         for file_name in file_names:
           # Use file name as pose class name.
           class_name = file_name[:-(len(file_extension) + 1)]

           # One file line: `sample_00001,x1,y1,x2,y2,....`.
           with open(os.path.join(pose_samples_folder, file_name)) as csv_in:
             csv_in_reader = csv.reader(csv_in, delimiter=file_separator)
             for row in csv_in_reader:
               row.insert(1, class_name)
               csv_out_writer.writerow(row)

       files.download(pose_samples_csv_path)


     dump_for_the_app()
```

Figure 16. Screenshot for Colab code block used to generate and download a CSV file.

5.1.3 Implementation without classification

This demo application is designed to demonstrate how to incorporate Google ML Kit Pose Detection API into developing application. The application makes use of the Google-developed base model with its main function: identify a person's posture from a static image.

When the app is open, users can choose an image from the device's photo gallery. The selected image will be displayed on the screen. Then, users can choose to detect pose from the selected image or clear the selection by clicking the corresponded button.

Based on the process described above: the application has three different features that are implemented: pick an image, detect a human's posture from the image, and save the result to the local storage.

Pick an image

The first feature to implement is pick an image from the photo gallery. When the application is installed on a new device, users must be shown a window where the permissions to access the photo gallery can be either granted or denied as shown in figure 17.

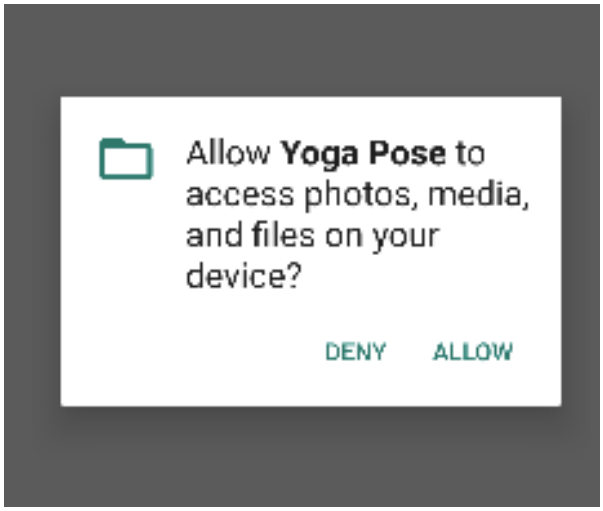


Figure 17. Screenshot of permission dialog.

In the demo application, this window will be launch when the user starts to choose an image by clicking a button as in figure 18. The permissions are checked every time the user tried to access the photo gallery. The permission checked is performed in the **onCreate** method of MainActivity class. If the permissions have been already granted, the permission request dialog in figure 17 does not need to be shown to the user again. When **checkSelfPermission** method is called, it will return a response based on which user can access the media or not. If the response is "**PERMISSION_GRANTED**", it indicates that the permissions have already granted, and the application can open the device's photo gallery. A response "**PERMISSION_DENIED**" invokes the "requestPermissions" method, which opens the permission request window as in figure 17. After the permission request is completed, the systems call the `**onRequestPermissionsResult**`

method, where it can be determined what happens if the permissions are granted or denied as in figure 19.

```
// Select image from gallery
buttonImage.setOnClickListener{ it: View!
    // Check for permissions
    if (permissionGranted()){
        selectImage()
    }else{
        // Ask for permissions
        ActivityCompat.requestPermissions( activity: this@MainActivity, REQUIRED_PERMISSIONS, REQUEST_CODE_PERMISSION)
    }
}
```

Figure 18. Select an image from devices' photo gallery.

```
// Check users' permission
private fun permissionGranted() =REQUIRED_PERMISSIONS.all{ it: String
    ContextCompat.checkSelfPermission(baseContext, it) == PackageManager.PERMISSION_GRANTED
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)

    if(requestCode == REQUEST_CODE_PERMISSION){
        Log.d( tag: "Permission",  msg: "Permissions granted")
        selectImage()
    }else {
        Toast.makeText( context: this,  text: "Permissions not granted!", Toast.LENGTH_SHORT).show()
    }
}


companion object{
    private  val REQUIRED_PERMISSIONS = arrayOf(
        Manifest.permission.READ_EXTERNAL_STORAGE,
        Manifest.permission.WRITE_EXTERNAL_STORAGE
    )
    private const val REQUEST_CODE_PERMISSION = 0
}
```

Figure 19. Permission request flow.

When users click the photo button and all permissions have been granted, the **selectImage** function is called. The function starts the process of display photo gallery for the user to select an image. The selected image's content Uniform Resource Identifier (URI) is then handled in the lambda expression within **registerForActivityResult**, which is displayed the selected image on the screen.

```
// Launch the photo picker and let the user choose only images.
private fun selectImage() = pickedImage.launch( input: "image/*")

// Registers a photo picker activity launcher in single-select mode.
private val pickedImage = registerForActivityResult(ActivityResultContracts.GetContent()){ uri ->
    uri?.let{ it: Uri
        Log.d( tag: "Select image",  msg: "Selected URI: $it")
        binding.imageViewPreviewPhoto.setImageURI(it)
    }
}
```

Figure 10. Image picker.

After the user selects an image, there are two options available: clear the selected image or start the posture detection process. The following figure illustrates the clear image process.

```
// Clear selected image
buttonDelete.setOnClickListener { it: View!
    if(imageViewPreviewPhoto.drawable != null){
        val builder: AlertDialog.Builder = AlertDialog.Builder( context: this@MainActivity)
        builder
            .setMessage("Are you sure?")
            .setTitle("Remove selected image?")
            .setPositiveButton( text: "Yes") { dialog, which ->
                imageViewPreviewPhoto.setImageResource(0)
                textViewStart.text = "Start by choosing an image"
            }
            .setNegativeButton( text: "No") { dialog, which ->
                Toast.makeText( context: this@MainActivity,  text: "Cancel the process", Toast.LENGTH_SHORT).show()
            }

        val dialog: AlertDialog = builder.create()
        dialog.show()
    }
}
```

Figure 11. Clear selection choice.

When the user clicks the button to detect the posture from the image, first the application will check if an image is selected. If there is no image, a toast message asking the user to choose an image is displayed. Otherwise, the image is processed with ML Kit's Pose Detection by calling the **detectPose** methon on a **poseViewModel** object. After a delay of 2 seconds, the **ResultActitivity** starts. The implementation is shown in figure 22.

```
// Start detect pose
buttonStart.setOnClickListener{ it: View!
    if(imageViewPreviewPhoto.drawable == null){
        Toast.makeText( context: this@MainActivity,  text: "Please choose an image!", Toast.LENGTH_SHORT).show()
    }else{
        Toast.makeText( context: this@MainActivity,  text: "Processing....", Toast.LENGTH_SHORT).show()
        poseViewModel.detectPose( context: this@MainActivity, imageViewPreviewPhoto.drawable.toBitmap())
        Handler(Looper.getMainLooper()).postDelayed({
            val resultIntent = Intent( packageContext: this@MainActivity, ResultActivity::class.java)
            startActivity(resultIntent)
            finish()
        },  delayMillis: 2000)
    }
}
```

Figure 12. Detect pose button.

When using ML Kit's Pose Detection, detection mode can be specified. The settings for Pose Detection mode are configured using an object of type `**AccuratePoseDetectorOptions**` as in figure 23. An instance of the PoseDetection class is created by calling the `**PoseDetection.getClient**` method.

```
// Specify the pose detector options
private var options = AccuratePoseDetectorOptions.Builder()
    .setDetectorMode(AccuratePoseDetectorOptions.SINGLE_IMAGE_MODE)
    .build()
// Create an instance of PoseDetector
private val poseDetector = PoseDetection.getClient(options)
```

Figure 23. ML Kit's Pose Detection mode.

The **PoseDetection** object can receive images for analysis by calling its method "process". This method accepts images of type **InputImage**, which can be created either from a Bitmap, media.Image, ByteBuffer, byte array, or a file on the device (Google 2024). In this demo application, the **InputImage** is created from a Bitmap object as shown in figure 24.

```
// Prepare the input image by creating an InputImage object from a Bitmap object
val image = InputImage.fromBitmap(bitmap,  rotationDegrees: 0)
```

Figure 24. Prepare input image for analysis.

The process method call returns an asynchronous task, to which listeners can be attached as in figure 25. Listeners are used in conjunction with the ML Kit API such as **onSuccessListener**, **OnFailureListener**. The code in figure 25 defined that if the process is successfully completely, a function named **processPose** will be executed. On the other hand, if the process is failed to complete, a pop up will be show.

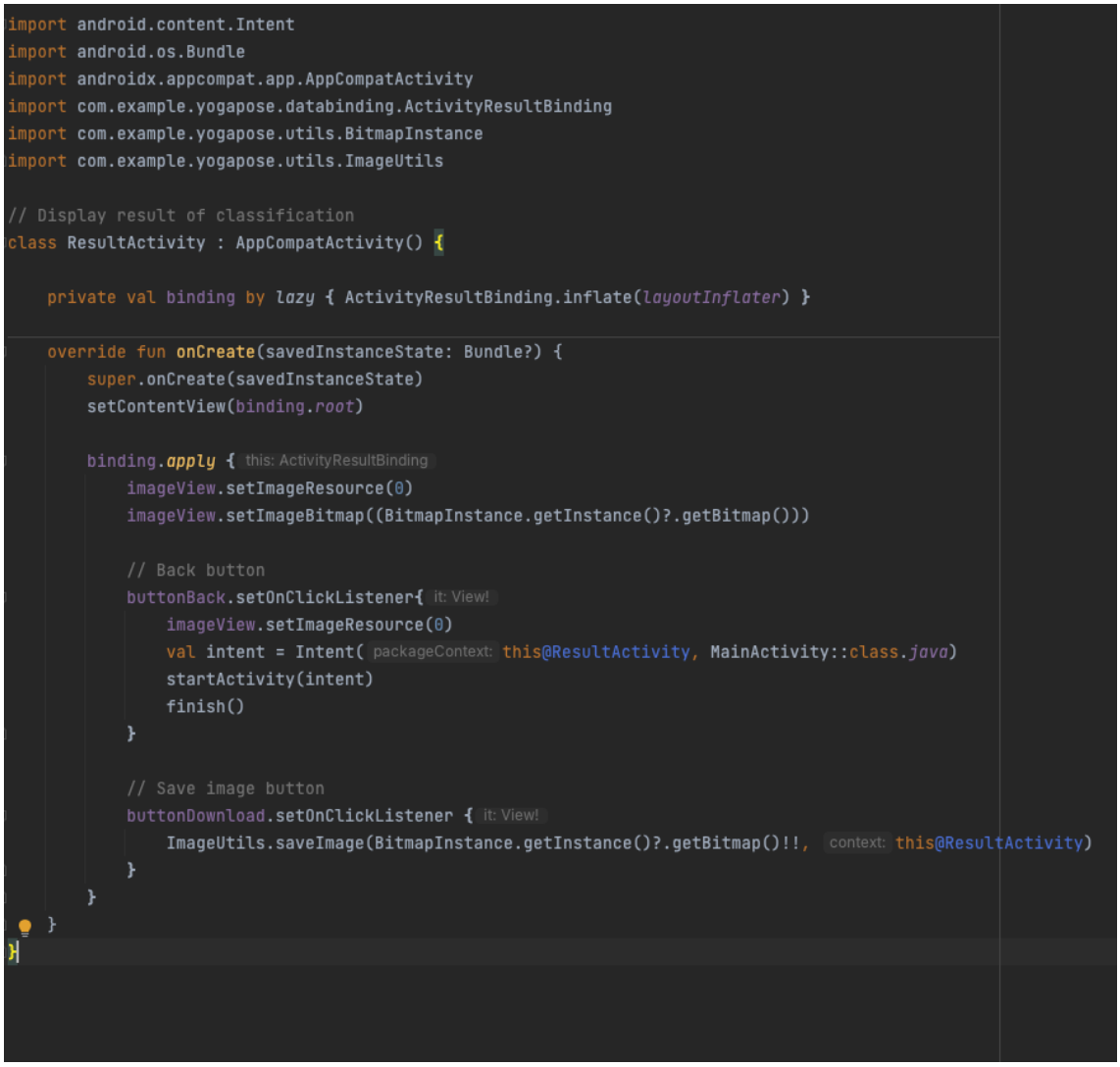
```
// Process the image
 poseDetector.process(image)
    // Task completed successfully
    .addOnSuccessListener { pose ->
        processPose(context, bitmap, pose)
    }
    // Task failed with an exception
    .addOnFailureListener { it: Exception
        Toast.makeText(context,  text: "Can't detect pose!", Toast.LENGTH_SHORT).show()
    }
```

Figure 25. Analyse the image for detecting pose.

If the task completes successfully, the result of the pose detection will be shown. The result is a drawing of the pose. With the ML Kit Pose Detection API, a full-body 33 landmarks were returned. Based on those landmarks, the demo application can draw a skeleton image from shoulder downward of the pose by connecting those landmarks as in figure 26.

Figure 26. Result of pose detection.

The display of pose detection result is handled in the **ResultActivity**. The code block in figure 27 ensures that the result image is displayed, can be saved, and allows the user to go back to the main activity.



```
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.example.yogapose.databinding.ActivityResultBinding
import com.example.yogapose.utils.BitmapInstance
import com.example.yogapose.utils.ImageUtils

// Display result of classification
class ResultActivity : AppCompatActivity() {

    private val binding by lazy { ActivityResultBinding.inflate(layoutInflater) }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(binding.root)

        binding.apply { this: ActivityResultBinding
            imageView.setImageResource(0)
            imageView.setImageBitmap((BitmapInstance.getInstance()?.getBitmap()))

            // Back button
            buttonBack.setOnClickListener{ it: View!
                imageView.setImageResource(0)
                val intent = Intent( packageContext: this@ResultActivity, MainActivity::class.java)
                startActivity(intent)
                finish()
            }

            // Save image button
            buttonDownload.setOnClickListener { it: View!
                ImageUtils.saveImage(BitmapInstance.getInstance()?.getBitmap()!!, context: this@ResultActivity)
            }
        }
    }
}
```

Figure 27. ResultActivity.

5.1.4 Implementation with classification

This demo application is designed to demonstrate how to incorporate Google ML Kit Pose Detection API into developing application with a custom trained model. The application makes use of the downloaded CSV file to implement its main function: detect a human posture and classify yoga poses from a live camera feed. The application has three different features that are implemented in order:

open the device's camera, detect human posture, classify the yoga pose and displayed the result of classification.

To integrate the custom trained model into the application, developers need to add the exported CSV file downloaded from Google Colab to the app's asset folder (figure 28).

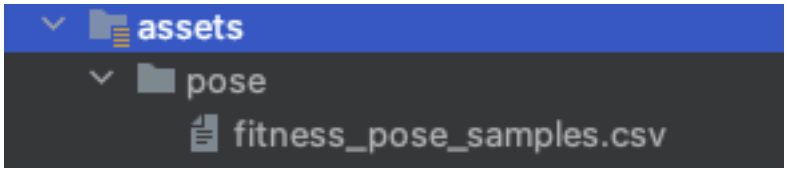


Figure 28. Assets folder of the application.

In addition to ML Kit's Pose Detection dependencies, the application needs to add CameraX depencies since it uses the camera live feed as the input for the classification as shown in figure 29. Not only that, but it also needs to update the **AndroidManifest** as illustrated in figure 30.

```
// Camera X
implementation ("androidx.camera:camera-core:1.4.0-alpha04")
implementation ("androidx.camera:camera-camera2:1.4.0-alpha04")
implementation ("androidx.camera:camera-lifecycle:1.4.0-alpha04")
implementation ("androidx.camera:camera-video:1.4.0-alpha04")
implementation ("androidx.camera:camera-view:1.4.0-alpha04")
implementation ("androidx.camera:camera-extensions:1.4.0-alpha04")
implementation("androidx.camera:camera-mlkit-vision:1.4.0-alpha04")
```

Figure 29. CameraX dependencies.

```
<uses-feature
    android:name="android.hardware.camera"
    android:required="false" />
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.CAMERA"/>
```

Figure 30. Users' permissions from AndroidManifest file.

The demo application has been developed by leveraging the foundational code from ML Kit's quickstart app project available on GitHub (Appendix 1). Developers can extract relevant files from the project and incorporate them into their own app development process. In this project, all files in folder **posedetector, utils**, **graphics** are copied from the sample app. Figure 31 shows the structure of the application.

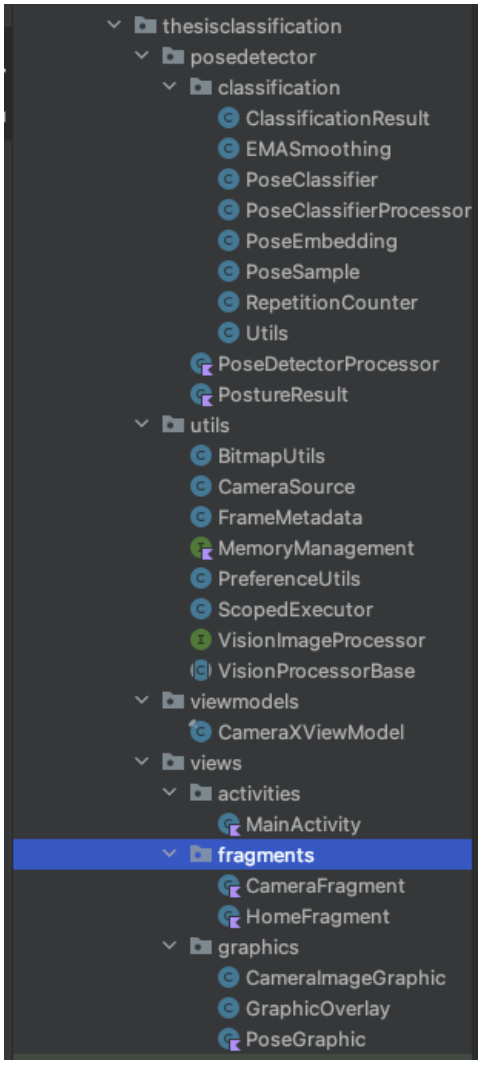


Figure 31. Application structure.

In the **PoseClassifierProcessor** file, developers need to update **POSE_SAMPLES_FILES** and **POSE_CLASSES** variables to match the CSV file and pose samples. In this demo application, it means five yoga poses: warrior, tree, downdog, goddess and plank as in figure 32. After updating the variables in

this file, the ML Kit Pose Detection is now able to classify the mentioned yoga poses.

```
// CSV file
1 usage
private static final String POSE_SAMPLES_FILE = "pose/fitness_pose_samples.csv";

// The class name for all the exercise
1 usage
public static final String WARRIOR_CLASS = "warrior2";
1 usage
public static final String TREE_CLASS = "tree";
1 usage
public static final String PLANK_CLASS = "plank";
1 usage
public static final String DOWNDOG_CLASS = "downdog";
1 usage
public static final String GODDESS_CLASS = "goddess";
1 usage
public static final String[] POSE_CLASSES = {
        WARRIOR_CLASS, TREE_CLASS, PLANK_CLASS, DOWNDOG_CLASS, GODDESS_CLASS
};
```

Figure 32. Update variables in PoseClassifierProcessor.

The demo application has only one activity **MainActivity** and 2 fragments. The main activity inflates the layout and initializes the navigation components. The home fragment welcomes users and provides instructions on how to start using the application. The camera fragment launches the device's camera. Most of function in the camera fragment can be found on the quickstart app. Figures 33 show the navigation graph of the application.

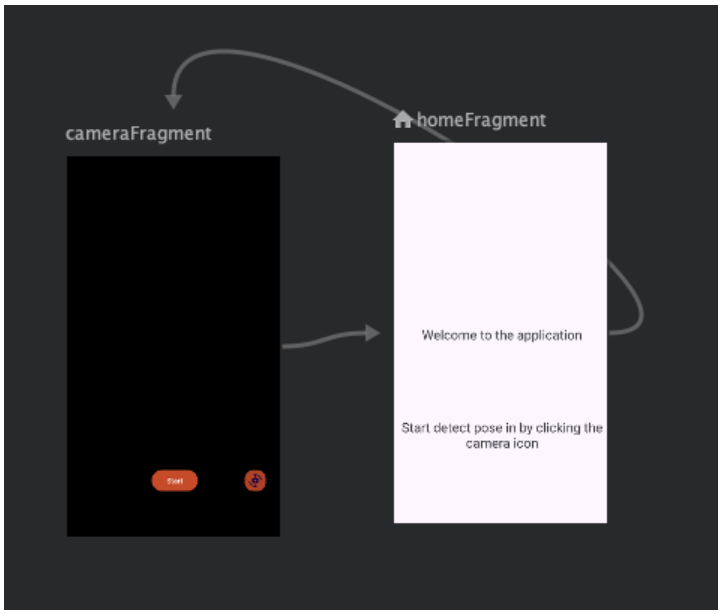


Figure 33. Application navigation graph.

When users move to **CameraFragment**, the application checks for the necessary permissions as the fragment require access to the device's camera. If users allow permission for the camera, ML Kit's Pose Detection starts detecting and tracking body posture once the camera is working as shown in figure 34.

Figure 34. CameraFragment.

The yoga classification won't be triggered unless users click the start button. Once the classification process starts, start button and camera flip button disappear from screen and a stop button appears. The process logic can be found in figure 35. The result of the classification can be found in figure 36.

```
// Start button
startButton.setOnClickListener { it: View!
    // start triggering classification process
    cameraViewModel.triggerClassification.value = true
    // showing loading AI Pose detection model
    loadingTV.visibility = View.GONE
    loadProgress.visibility = View.GONE
    cameraFlipFAB.visibility = View.GONE
    buttonStop.visibility = View.VISIBLE
    startButton.visibility = View.GONE
}
```

Figure 35. Start button on CameraFragment.

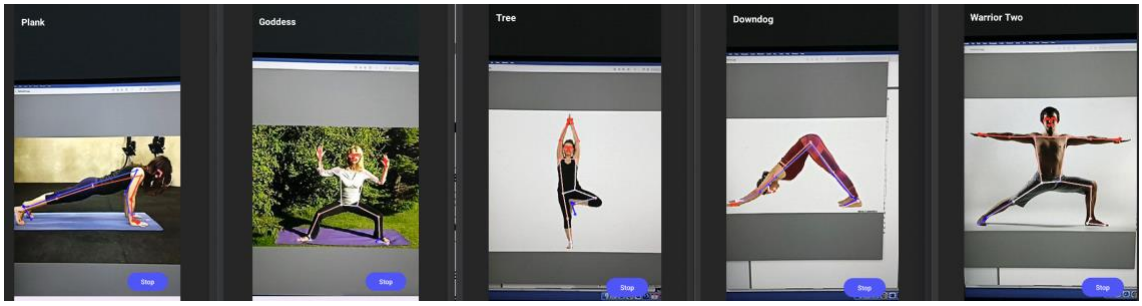


Figure 136. Result of classification.

5.2 Text translation application

This demo application is created following the instruction of ML Kit on Android. The application allows users to input English text and receive the corresponding Finnish translation. First, to integrated ML Kit Translate into developing application, the dependency for ML Kit must be added as shown in figure 37.

```
// ML Kit Translate
implementation ("com.google.mlkit:translate:17.0.2")
```

Figure 37. ML Kit Translate dependency.

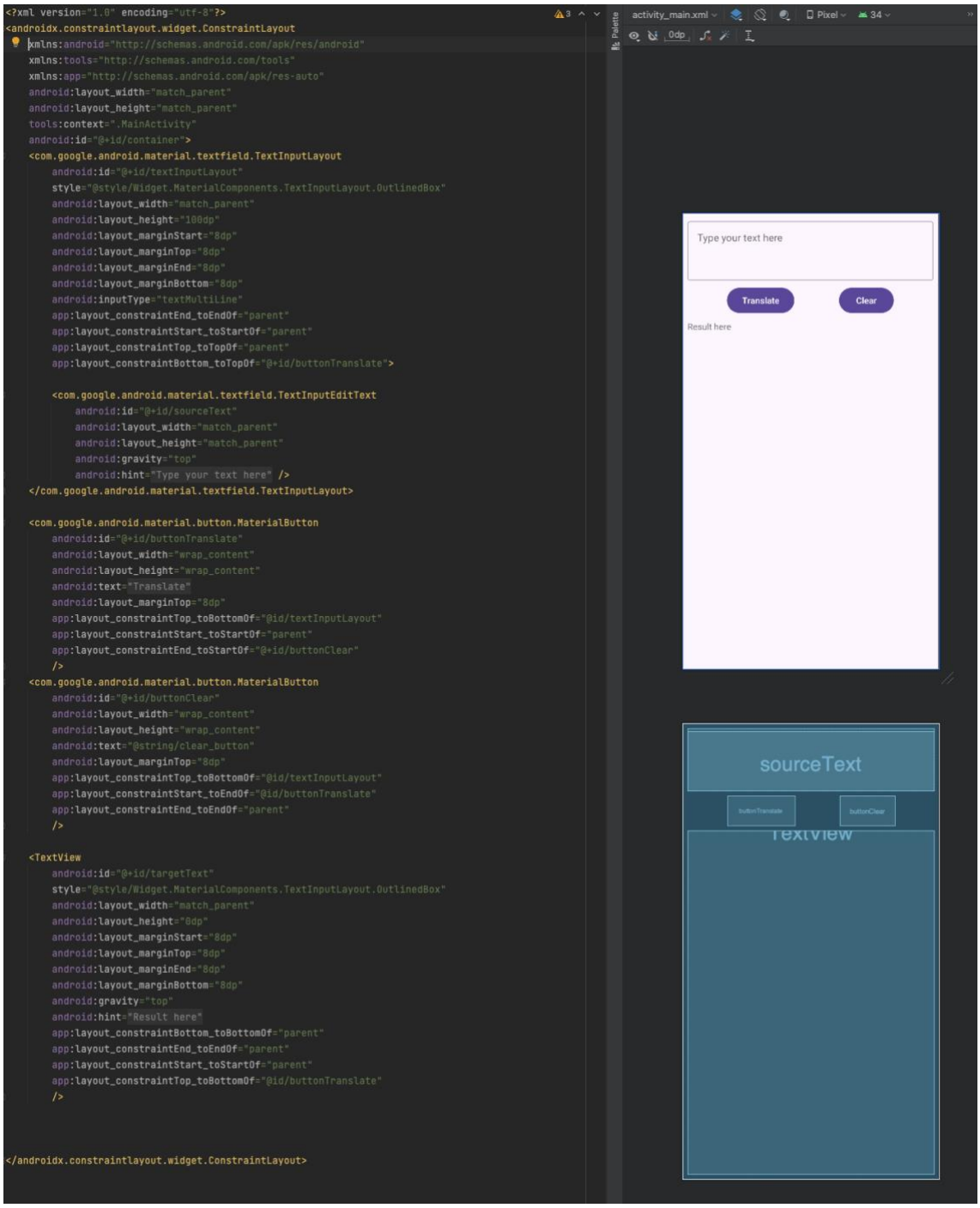


Figure 38. Layout of the application.

Figure 38 demonstrates the **MainActivity** component and styling for screen. There is one box for users to type in text, one box to show the translated text, and two buttons for users to interact with the application. The **clear** button is to delete users' input text as well as its translated result. The **translate** button lets users translate their text.

To translate a string between two languages, the first step is to create a **Translator** object and configure it with the source and target languages (English and Finnish in order). When the users click the **Translate** button, the application checks the input text. If the input text is empty, the application displays a brief message prompting users to enter some text. If the input text is not empty, the following steps occur:

- The application checks whether the required translation model has been downloaded to the device.
- If the model is available, it proceeds to translate the text and display the result on screen.
- If the model is not downloaded, an error is logged to the console.

Figure 39 shows the process described above and its result.

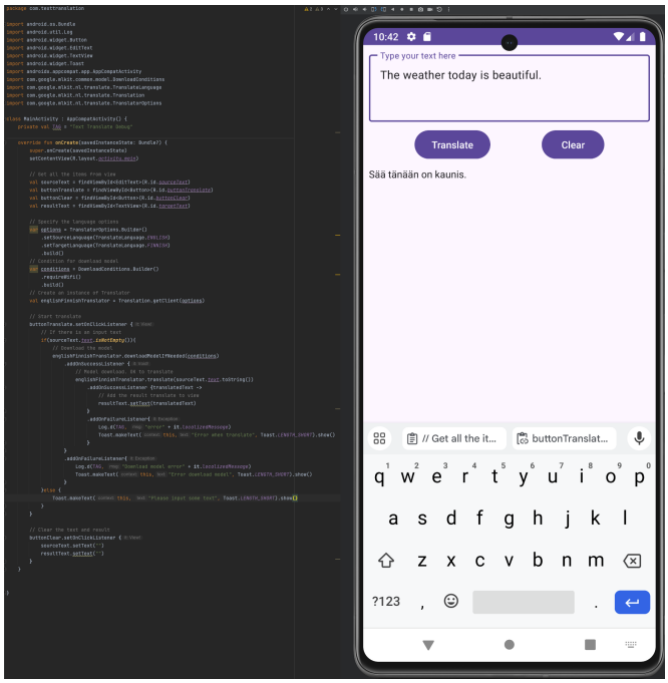


Figure 39. MainActivity file and the application at work.

5.3 Discussion

Overall, all demo applications are completed and functional. The implementation was straightforward due to the thoroughly instructions as well as the example app provided by ML Kit's framework.

ML Kit's Pose Detection model and MediaPipe Colab provide developers with powerful tools for integrating human's body posture detection and tracking into their application. One application can detect a pose from an image, while another one can detect, track, and classify a yoga pose from a live camera feed. One drawback of ML Kit's Pose Detection is that the model can only detect one person in a frame. When multiple people are present, the model assigns landmarks to the person it detected with the highest confidence. In addition, it works best when the subject's entire body is visible in the frame. Although the model can detect partial body poses, but in this case, the landmarks that are not recognized are assigned coordinates outside of the image. This affects the result of classification. It is important that developers include user instructions for users in their application. (Google 2024.)

The most challenging part in the process is to create a custom-trained model when implementing pose detection applications. Developers might encounter errors when executing the code blocks from the provided Colab. Without a good understanding of Python, it can be time-consuming to troubleshoot and resolves these issues.

However, there is room for further development in the demo applications. A significant improvement to the demo applications would be database integration. With database, results can be saved and reviewed later. Another function for the pose detection applications is to calculate angles between joints, for example between hip and knee, and the application can give feedback to users if their postures are correct or not. For translation application, the application can let users choose what languages they want to use in translation since the ML Kit

Translate API supports dynamic model downloads. Additionally, implementing tests would be beneficial to ensure the functionality of applications.

6 Conclusion

The purpose of the study was to explore the utilization of Google ML Kit in Android application development by building demo applications. Through the demo applications, it is proven that creating a ML based application using Google MK Kit framework is not complicated for developers. With basic knowledge of software development, developers can build applications by following the guidelines and reviewing example applications provided by Google ML Kit.

When integrating ML into their applications using Google ML Kit, developers can follow a straightforward process. First, developers should determine the specific machine learning tasks they need for the application. Then, based on the identified task, developers can select the appropriate ML Kit API. Finally, developers can integrate the chosen API into their application following ML Kit's documentation and instructions.

Based on this study it seems that knowing about the inner-working of machine learning did not seem beneficial when developing application utilizing Google ML Kit. The reason for it is because Google ML Kit's APIs make use of Google trained machine learning models. Developers do not need to create machine learning models from scratch, so no extended knowledge is required.

More often, the default model in ML Kit framework might not suit developers' exact use cases and there is a need for a different model. Fortunately, some ML Kit APIs provide tools to create custom models such as MediaPipe Colab, which simplify the process of fine-tuning models and allow developers to retrain models without delving deep into ML theory.

Machine learning is a complex topic. To delve deeper in machine learning field would require more advanced knowledge of mathematics than a typical

software developer possesses. Nonetheless, the thesis demonstrates how Google’s ML Kit framework is making ML more approachable for developers.

References

Alpaydin, Ethem. 2016. Machine Learning: the New AI. Cambridge, MA: MIT Press.

Apple Inc. 2024. Core ML. Online. Apple Developer. <https://developer.apple.com/machine-learning/core-ml/>. Accessed: 20 March 2024.

Banoula, Mayank. 2023. Supervised Machine Leaning: All You Need to Know. Online. Simplilearn. < https://www.simplilearn.com/tutorials/machine-learning-tutorial/supervised-machine-learning>. Accessed: 6 March 2024.

Brown, Sara. 2021. Machine Learning, Explained. Online. MIT Sloan School of Management. <https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained>. Accessed: 6 March 2024.

Burke, Dave. 2023. A New Foundation for AI on Android. Android Developers Blog. <https://android-developers.googleblog.com/2023/12/a-new-foundation-for-ai-on-android.html>. Accessed: 21 March 2024.

Dutt, Saikat; Chandramouli, Subramanian & Das, Kumar Amit. 2018. Machine Learning. India: Pearson.

Chollet, Francois. 2021. Deep Learning with Python. Shelter Island: Manning Publications Co.

GeeksforGeeks. 2023. What is Transfer Learning. Online. GeeksforGeeks. <https://www.geeksforgeeks.org/ml-introduction-to-transfer-learning/>. Accessed: 2 April 2024.

Google. 2023. About the Preview. Online. Google for Developers. <https://developers.google.com/mediapipe/solutions/guide>. Accessed: 23 March 2024.

Google. 2024. Android AICore. Online. Google for Developers. <https://developer.android.com/ml/aicore>. Accessed: 21 March 2024.

Google. 2024. Colaboratory. Online. Google. <https://research.google.com/colaboratory/faq.html>. Accessed: 25 April 2024.

Google. 2024. Custom models with ML Kit. Online. Google for Developers. <https://developers.google.com/ml-kit/custom-models>. Accessed: 20 March 2024.

Google. 2024. Detect poses with ML Kit on Android. Online. ML Kit. <https://developers.google.com/ml-kit/vision/pose-detection/android>. Accessed: 5 March 2024.

Google. 2024. Detect poses with ML Kit on iOS. Online. ML Kit. < https://developers.google.com/ml-kit/vision/pose-detection/ios>. Accessed: 5 March 2024.

Google. 2024. Grant partial access to photos and videos. Online. Google for Developers. <https://developer.android.com/about/versions/14/changes/partial-photo-video-access>. Accessed: 24 March 2024.

Google. 2021. Model Card. Online. Google for Developers. <https://storage.googleapis.com/mediapipe-assets/Model%20Card%20BlazePose%20GHUM%203D.pdf >. Accessed: 5 April 2024.

Google. 2023. MediaPipe Model Maker. Online. MediaPipe. <https://developers.google.com/mediapipe/solutions/model_maker>. Accessed: 5 April 2024.

Google. 2023. MediaPipe Solutions Guide. Online. Google for Developers. <https://developers.google.com/mediapipe/solutions/guide>. Accessed: 23 March 2024.

Google. 2024. ML Kit. Online. Google for Developers. < https://developers.google.com/ml-kit>. Accessed: 20 March 2024.
Google. 2024. Overview of Colaboratory. Online. Google Colab. <https://colab.research.google.com/notebooks/basic_features_overview.ipynb>. Accessed: 25 March 2024.

Google. 2022. Pose classification options. Online. ML Kit. <https://developers.google.com/ml-kit/vision/pose-detection/classifying-poses>. Accessed: 15 March 2024.

Google. 2024. Pose Landmark Detection Guide. Online. Google for Developers. <https://developers.google.com/mediapipe/solutions/vision/pose_landmarker/index#pose_landmarker_model>. Accessed: 23 March 2024.

Google. 2024. Translation. Online. ML Kit. <https://developers.google.com/ml-kit/language/translation>. Accessed: 2 May 2024.

Google. 2024. Translate text with ML Kit on Android. <https://developers.google.com/ml-kit/language/translation/android>. Accessed: 2 May 2024.

Google. 2024. What is Supervised Learning. Online. Google Cloud. <https://cloud.google.com/discover/what-is-supervised-learning>. Accessed 6 March 2024.

Google. 2022. What is Clustering. Online. Machine Learning. <https://developers.google.com/machine-learning/clustering/overview>. Accessed: 23 March 2024.

Google. 2024. Why On-Device Machine Learning. Online. Google for Developers. <https://developers.google.com/learn/topics/on-device-ml/learn-more>. Accessed: 20 March 2024.

Gopalakrishnan, Revathi & Venkateswarlu, Avinash. 2018. Machine Learning for Mobile. Birmingham: Packt Publishing.

IBM. 2024. What is Unsupervised Learning. Online. <https://www.ibm.com/topics/unsupervised-learning>. Accessed: 26 April 2024.

The MathWorks Inc. 2020. Reinforcement Learning with MATLAB. Online. <https://se.mathworks.com/content/dam/mathworks/ebook/gated/reinforcement-learning-ebook-all-chapters.pdf>. Accessed: 23 March 2024.

O'Donnellan, Ruairi. 2024. Machine Learning by The Numbers: Its Impact on Business. Online. Intuition. <https://www.intuition.com/machine-learning-by-the-numbers-its-impact-on-business/>. Accessed 4 March 2024.

Patel, A. Ankur. 2019. Hands-On Unsupervised Learning Using Python. Sebastopol, CA: O’Reilly Media Inc.

Prins, Christiaan & Hu, Shiyu. 2020. On-device Machine Learning Solutions with ML Kit, Now Even Easier to Use. Online. Google Developers Blog. <https://android-developers.googleblog.com/2020/06/mlkit-on-device-machine-learning-solutions.html >. Accessed: 20 March 2024.

Ng, Karthikeyan. 2018. Machine Learning Projects for Mobile Applications. Birmingham: Packt Publishing.

Singh, Anubhav; Bhadani, Rimjhim. 2020. Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter. Birmingham: Packt Publishing.

Sutton, S. Richard; Barto, G. Andrew. 2018. Reinforcement Learning: An Introduction. London: The MIT Press.

TensorFlow. 2024. TensorFlow Lite. Online. <https://www.tensorflow.org/lite/guide>. Accessed: 20 March 2024.

TensorFlow. 2024. TensorFlow Lite Delegates. Online. <https://www.tensorflow.org/lite/performance/delegates>. Accessed: 20 March 2024.

Zaheer, Asim. 2023. The Future of AI in Mobile App Development. Online. Spiceworks. <https://www.spiceworks.com/tech/artificial-intelligence/guest-article/role-of-ai-in-mobile-app-development/>. Accessed 4 March 2024.

Wang, Jindong; Chen, Yiqiang. 2023. Introduction to Transfer Learning. Singapore: Springer Nature Singapore Pte Ltd.

Links to Resources Used

ML Kit Vision Quickstart Sample App

https://github.com/googlesamples/mlkit/tree/master/android/vision-quickstart

Link to MediaPipe Colab to build a custom pose classifier:

https://colab.research.google.com/drive/19txHpN8exWhstO6WVkfmYYVC6uug_oVR

Link to yoga pose dataset:

https://www.kaggle.com/datasets/niharika41298/yoga-poses-dataset