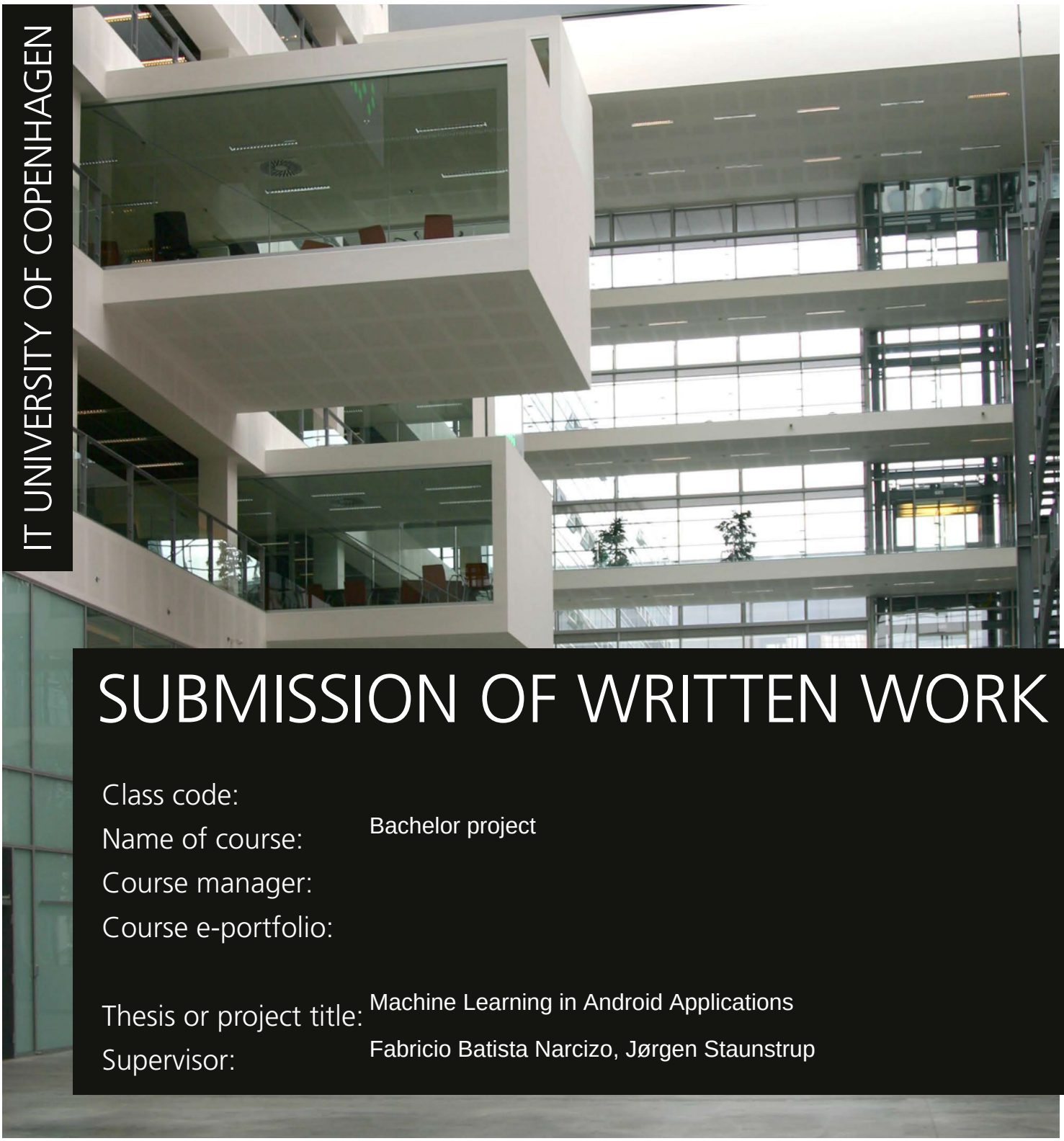
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code:

Name of course: Bachelor project

Course manager:

Course e-portfolio:

Thesis or project title: Machine Learning in Android Applications

Supervisor: Fabricio Batista Narcizo, Jørgen Staunstrup

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Michelle Lykke Blomgaard Karlsson	08/12-1997	milk@itu.dk
2.			@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk
8.			@itu.dk

Machine Learning in Android Applications

Michelle Lykke Blomgaard Karlsson

Spring 2020

1 Abstract

This project is motivated by the ever-increasing popularity of machine learning techniques for solving repetitive everyday tasks, as well as the availability and importance of smartphones in today's society. The combination of the two creates an environment in which the use of machine learning for simplifying mundane tasks in mobile applications may be experimented with. This project is a study in the use of machine learning in the context of such a mobile application, and specifically uses the Firebase ML Kit mobile SDK in that pursuit. The project includes the development of an application that allows users to generate descriptions of electronic devices they wish to post for sale on online marketplaces. The application utilizes machine learning and natural language generation to present the user with a textual description of the image submitted to the application. This project gives an overview of central machine learning principles, and goes into detail about the concepts relevant to solving the problem in question, namely classification, and neural networks. It also describes the process of implementing the application and how Firebase ML Kit provides machine learning capabilities, as well as how SimpleNLG provides natural language generation functionality to the application. The project further reflects on the application created and the use of ML Kit therein.

1.1 Github

The application developed as part of this project can be found at:

https://github.itu.dk/milk/Machine-Learning-in-Android-Applications

Contents

2 Introduction

Machine learning continues to grow in popularity, as demonstrated by the plethora of easily accessible online services aiming to educate or aid developers in introducing machine learning functionality into their software products. An example of such an online service is Firebase ML Kit, which allows developers of every skill-level to add such functionality to their applications.

Further motivation can be found in the fact that smartphones are an important part of everyday life in 2020. People use them in many contexts and for many tasks, including making posts for online marketplaces such as Facebook Marketplace. Many items are posted on Facebook Marketplace every day, so the automation of the process of creating the required description for each item is desirable.

The problem this study focuses on is that of automatically generating a description of an item in a picture with as little human interference as possible.

Section 3 gives an overview of central machine learning concepts. Section 3.1.1 describes what characterizes a supervised learning problem, and Section 3.1.2 describes unsupervised learning problems. Section 3.2 and Section 3.3 go into depth about two concepts of special interest when it comes to solving the stated problem, namely neural networks and classification problems.

Section 4 provides a description of the development process of the application. Section 4.1 goes into depth with all the details concerning the use of Firebase ML Kit, such as the training of the model and the implementation of the functionality in the application. Section 4.2 gives a brief overview of how the SimpleNLG library allows for the generation of the final description, which is output to the user. Section 4.3 gives an overview of the activities and fragments in the application and their individual roles.

Finally, Section 6 reflects on the use of Firebase ML Kit and the implementational details of the application, and Section 7 presents the results of the project.

3 Machine Learning Concepts

This chapter gives a brief overview of fundamental machine learning concepts, followed by a more detailed description of two of the central concepts for solving the problem at hand, namely neural networks and classification.

3.1 Machine Learning

Machine learning is the concept of an algorithm learning from data provided. The term was initially defined by Tom Mitchell in 1997, as the following.

"A computer program is said to **learn** from experience E with respect to some class of task T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [7]

Machine learning problems divide into one of two sub-categories: supervised and unsupervised learning, respectively.

3.1.1 Supervised learning

In supervised learning, the training data contains the "right answer" along with the individual data items.
Supervised learning algorithms build on the problem of function approximation: the data contains an input parameter x and an output parameter y, and the algorithm is approximating the mapping function from x to y. The algorithm will then be able to predict the output y for a new x.
Supervised learning gets its name from the fact that the dataset provides the correct answer, and the training of the algorithm is essentially the task of supervising that the algorithm learns the correct answer and is corrected otherwise.

Supervised learning problems divide into one of the following two categories:

- **Regression** - Regression problems seek to predict the output as a continuous value. An example of a common regression problem could be predicting the house price based on parameters like how many rooms it has, or the size of the house in square meters [8]. The mean squared error is the primary performance measure for regression models.
- **Classification** - Classification problems seek to predict the output as a discrete value. Classification algorithms want to sort the input into one of a few possible categories. Examples of classification problems could be predicting whether an email is either spam or not spam [9], or examining a photo of an animal to determine what species it is (cat, dog, among others) The primary performance metric for classification algorithms is the precision, a relatively simple computation of the number of correct predictions made out of the total amount of predictions made.

This project aims to address the problem of determining what is in an image, from a few possible categories. Therefore, a more detailed description of classification problems will follow in Section 3.3.

3.1.2 Unsupervised Learning

In contrast to supervised learning, unsupervised learning trains on data that does not contain "right answers", meaning the problem is not that of function approximation, but rather that of finding structure in the data itself.
In unsupervised learning, there is no information given prior to the training,

and thus the algorithm must teach itself how to make sense of the data.

Unsupervised learning problems divide into one of the following two subcategories:

- **Clustering** - Clustering algorithms deal with grouping the data into one of several categories based on the similarity (proximity) to other data points.
- **Non-clustering** - Non-clustering algorithms deal with finding rules that apply to large portions of the data. For instance, anomaly detection algorithms examine a dataset and find the data points that are unlike the rest.

3.2 Neural Networks

(Artificial) Neural Networks are a simulation of how networks of neurons work in the human brain and provide a useful way of representing a hypothesis [10].

A neural network consists of a set of neurons connected by edges, split into layers: An input layer, a variable number of hidden layers, and an output layer. The activation function computes a value for each neuron and passes it on to the next layer. There are many options for activation functions, but one commonly used is the sigmoid function (which forces the values into a 0-1 range). The activation function computes the output for a neuron based on the input from the neurons in the previous layer along with the parameter given by the connections between them, often called "weights". The output layer computes the final hypothesis.

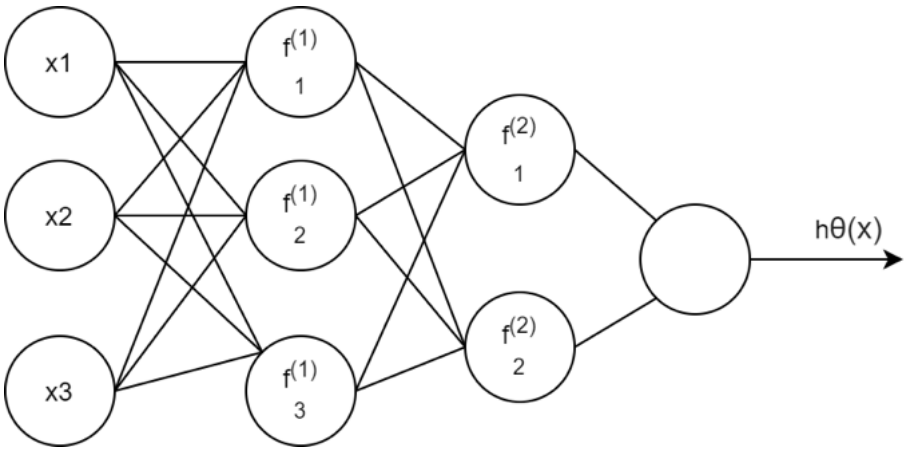


Figure 1: Example neural network with one input layer, two hidden layers and one output layer.

Figure 1 shows an example of a neural network. The neural network has three input units, $x1$, $x2$, $x3$, followed by two hidden layers in which the notation $f_y^{(x)}$ is used to reference each neuron, where x is the layer and y is the neuron within the layer. The output layer outputs the hypothesis, denoted $h\theta(x)$.

3.3 Classification

In classification problems the value of the output y is a discrete value, meaning there is a finite set of possible results.

If for some input x, we want to solve a classification problem by sorting x into one of several categories (y) (A multi-class classification problem), one could use an algorithm like One-vs-All. The general idea is that one uses logistic regression to create a classifier for each class i (separating the class from everything else), and on any new input x, running all the classifiers and selecting the one that outputs the highest number (the highest probability of the input x belonging to that class). This approach is reasonable when there is a small number of output classes, as it is necessary to run all of them on each new input x.

However, when the problem is much larger, like detecting an object in an image, there could be hundreds or thousands of output values we would want to be able to recognize. When this is the case, a better algorithm would be a neural network. The neural network can be designed with as many output nodes as needed to accomplish this. The output layer gives a vector in which the position of the value 1 denotes which class the image belongs to and all other classes have the value 0.

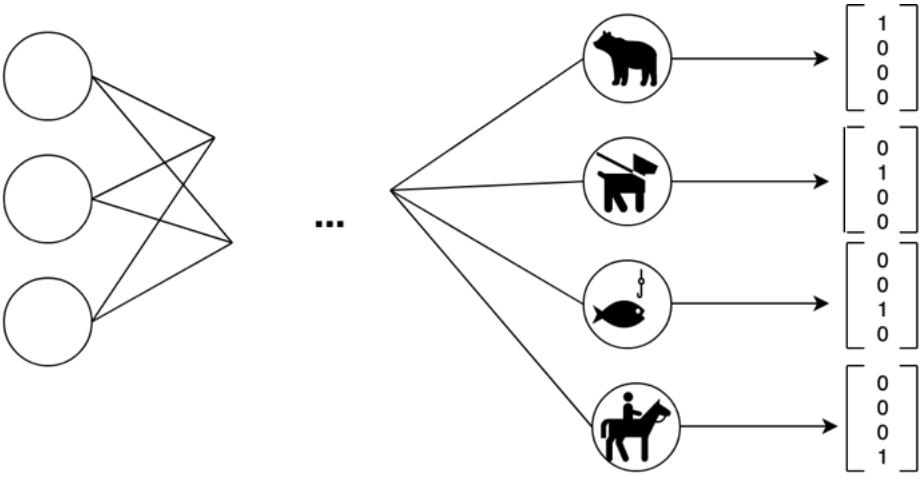


Figure 2: Classification of animal pictures in a neural network, hidden layers abstracted away

As an example, consider the task of classifying images of animals. Suppose having four categories, being *bear*, *dog*, $fish$, *horse*. The neural network has an input layer, some hidden layers, and an output layer. The output layer needs to have four nodes, one for each possible category the image could belong to. Figure 2 shows the output is a vector in which the number 1 denotes which class the image belongs to, and all other classes have the value 0.

4 Implementation

This chapter aims to give an overview of the implementational details of the application, especially with regards to the various steps of implementation of an ML Kit AutoML Vision Edge model for image classification.

4.1 ML Kit

This section aims to explore some of the ways of using ML Kit for machine learning features in an Android application, for the specific use case.

First and foremost, ML Kit provides a few ready-made models for some commonly found use cases that seamlessly integrates into an application. These models have both compact local versions, downloadable to the device, for use in situations where there might not be any Internet connection, and Firebase-hosted versions that provide a higher precision at the cost of needing internet access. The primary model of interest for this application would be the Image Labeling model. This model can recognize a wide range of objects. However, it also tends to return many labels that are *related* to the image (adjectives), making it excellent for uses where one wants to add several tags to images to sort them into categories, but less ideal for purposes where one wants to find an object within the image and create a description of it [5]. For this reason, using the Image Labelling model is inappropriate for this project.

Another option is that of tailoring a model to a specific use case through the use of the Custom Model feature. This allows the developer to upload a TensorFlow Lite model to Firebase, which provides for both particular and comprehensive models depending on what the application needs. TensorFlow provides a small library of a few pre-trained models that are in the required *lite* format and are ready to be used with ML Kit for Android or iOS development [15]. The option of using TensorFlow models lends itself well to situations where it may be desirable to be able to transfer a pre-trained model for use in a mobile application context.

ML Kit also provides the ability to host a model and train it through Firebase. This comes with the AutoML Vision Edge package. The user may upload a training set, and Firebase will automatically train and host a model based on it [2]. This is especially useful for cases where one might only want to recognize

a few select categories of things and tailor the output to the application's needs. This allows one to associate more information with an image, to whichever degree necessary, and generally customize the model a bit more. It comes with the restriction that datasets can only be up to 1000 training examples on the free plan of Firebase.

For this application, it seems evident that AutoML Vision Edge is best suited. Due to the full control of the training dataset and the output labels, it is possible to create a model for the specific use of determining an object and further being able to differentiate between different brands of a category of objects.

4.1.1 Training the models

AutoML Vision Edge needs a dataset for the model to be trained with. Firebase allows up to three hours [4] of free training which is more than enough training time for small applications.

Since the training data was limited to 1000 images, it was necessary to reduce the scope of the objects recognized to a few specific categories. Therefore, the application should be able to differentiate between the following subcategories:

- Smartphones
  - Android phones
  - Apple phones
- Laptops
  - Acer laptops
  - Dell laptops
  - Lenovo laptops
  - Apple laptops

The reason for this division of categories is simple: Using Google Image search, with the safe search filter for no copyright images on, to collect the dataset, it was possible to gather enough images of the categories at this level of specificity. Apple products tend to have many images of every kind of product in every color, but when searching for Android, specifying a single brand would return only a few images. As for laptops, specifying the color made it impossible to gather enough images, and for that reason, the search parameter became limited to just the brand. The initial result was a dataset consisting of 800 images, 100 of each subcategory.

However, the application needed a second model, due to the device model having difficulty differentiating between colors, and to ensure the ability to detect the color of laptops. The color model can distinguish between three of the most common colors for electronic devices: black, silver and white.

After training a model, the developer receives the evaluation details and a confusion matrix. This information aids the developer in making decisions about the threshold that will work best for the model, and which categories the model has difficulty differentiating between.

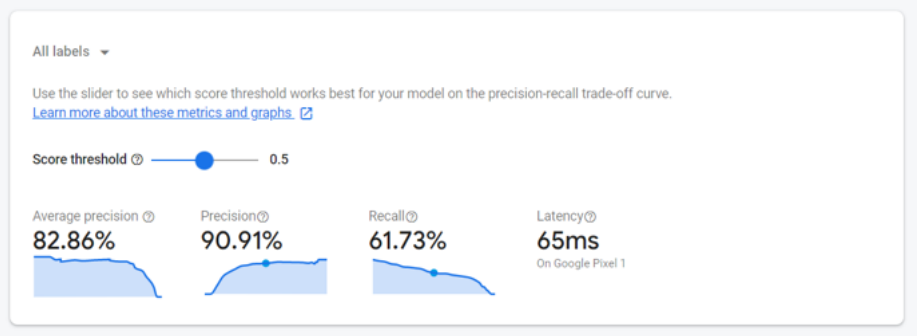


Figure 3: Evaluation of the device model, with a sliding bar for choosing the threshold

The factors affecting the confidence threshold are precision and recall, as shown in Figure 3. Precision is a metric for the model's ratio of correctly labeled images to the number of images labeled with the given threshold, selected by the sliding bar. The higher the precision, the fewer false positives (incorrectly assigned labels). Recall is a metric for the ratio of the number of images (correctly) labeled to the number of images that the model should be able to label. A high recall means the model fails to assign a label less often, which means fewer false negatives (not assigning a label for an image the model should have been able to recognize) [3]. The developer can then decide for themselves, depending on their use-case, which of the two is essential for them. For this application, the threshold is 0.5. Testing a few images of smartphones not found in the training set proved what the confusion matrix in Figure 4 also shows: the model is not very good at telling the difference between Android phones and Apple phones. The threshold is reasonably low, but for this application, it was better to be a bit more confident in the label where possible (prioritizing the precision).

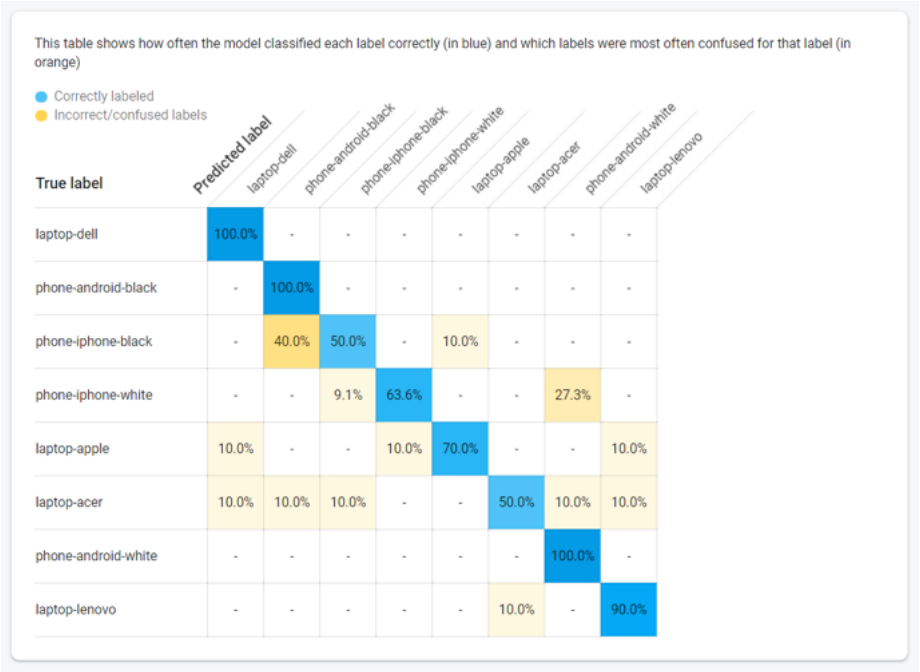


Figure 4: Confusion matrix for the device model

Figure 4 shows the confusion matrix for the created device model. The main issue it highlights is the model's difficulty in differentiating between iPhones and Android phones. The model further has some issues separating Acer laptops from other brands. It is important to note that the model still contains some remnants of the color detection it initially performed, as shown by the class names. The color model performs this detection now, and thus the color information from the device model is disregarded. These two factors, the accuracy problem, and the poorly named classes, necessitated an update to the training data and the model, which Section 6 describes.

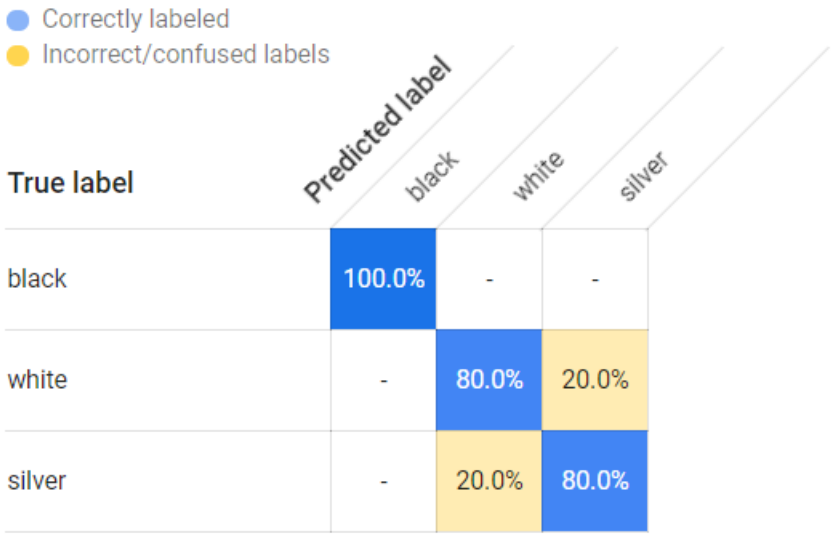


Figure 5: Confusion matrix for the color model

As Figure 5 shows, the color model mainly has problems differentiating between silver and white.

Section 6.1 discusses the problems the two models experience, and how it is mitigated in the final version of the application.

4.1.2 Model implementation

The addition of a model to the application is an efficient process thanks to a guide provided by Firebase [6]. All of the code for the labeling of images belongs to the SummaryActivity in the application. The following summarizes the flow of machine learning activities in SummaryActivity:

- Initialization of the remote and local models is the first step, and upon the creation of the two labelers (one for each model), the criteria for download of the remote models decide whether or not to proceed with the download. The remote models are either downloaded and used in the initialization of the labelers, or the local models are used instead.
- The user provides a photo, which is transformed into a FirebaseVisionImage, and the labelers process this one at a time. The output of this operation is a list of FirebaseVisionImageLabels, which is what the labeler detects in the image - Objects that have a String (being the label itself) and a Float representing the confidence.

- The application now has information about the device detected and the color thereof and uses this to create the final description.

4.2 Natural Language Generation

SimpleNLG [13] is the library chosen for handling the natural language generation after the gathering of facts from the picture. Formally, SimpleNLG is a realizer that allows the construction of meaningful sentences from some non-linguistic representation of information.

SimpleNLG and Natural Language Generation

The explanation of SimpleNLG requires a summary of some natural language generation concepts.

Natural language generation is a complex task, and as a result, NLG systems often quickly grow to be pretty complicated. In many cases it is beneficial to divide them into a few modules that each handle a specific task, acting as a pipeline that takes some non-linguistic input and produces an output in the form of some piece of text that makes sense and follows the grammatical and structural rules of the language it is in.

There are some distinct main categories of problems in NLG, namely the following [14].

- **Content Determination** - Problems concerning deciding on which information to include in the text.
- **Document Structuring** - Problems concerning deciding on the text structure and organization.
- **Lexicalisation** - Problems concerning deciding on specific words or phrases required to communicate the specified information.
- **Aggregation** - Problems concerning deciding on how to compose information into sentences of reasonable length.
- **Referring Expression Generation** - Problems concerning deciding on what properties of an entity to use when referring to the entity.
- **Surface Realisation** - Problems concerning deciding on how to create the sequence of grammatically correct sentences from the underlying context.

SimpleNLG is a surface realizer, which does not perform the other five tasks, so for this project, they are less critical.

SimpleNLG defines surface realization as consisting of two concepts: linguistic realization and structure realization [13]. The former is related to making

actual sentences from the abstract text, and the latter concerns converting abstract structures into mark-up symbols to be realized. In other words, the primary responsibility of the surface realizer is handling things like morphology (forming the words) and syntax (forming the sentences). Since sentences are not just random sequences of words strung together but have to come in a specific order and follow some rules for how a well-formed sentence is supposed to be, the surface realizer is an essential subsystem in any NLG system. In this application, the job of the realizer is relatively simple: it has some keywords and has to create three simple sentences and string them together. All it has to do is ensure the words of the sentences are in the correct form/tense, capitalizing the first word of the first sentence and handle punctuation.

A 4-tuple, `(k,c,u,d)`, represents the input to NLG systems, in which `k` is the knowledge source, `c` is the communicative goal, `u` is the user model, and `d` is the discourse history. In this application, `k` is a select few words, which have been found through machine learning and user input (alternatively, one could say the "true" `k` is the image input by the user), `c` could be said to simply be "express the information given by `k`", `u` represents the characterization of the user that the text output is intended for, in this instance it is of little interest. `d` is the model of previous interactions between the user and the system. Again, of little importance here, as the user only gets a single message from the system, and there is no real discourse [14](pg.39-41).

The output is, in this case, a short text about the input given. This is an instance of template-based NLG, wherein the non-linguistic input maps directly to surface output [1]. In this instance, the desired sentence is: "This is a (color) (brand) (model) (type), It is in (condition) condition, and it costs (price) (currency)" The blanks are filled in by the input in a way that makes sense.

This process is the final step in creating the description of the image for the user, and happens after the various activities and fragments in the application have collected all the relevant information. The factory, lexicon, and realizer provide the functionality of the SimpleNLG system. The factory initializes the sentence components needed, the lookup of known words happens in the lexicon, and the realizer creates the final sentence, by taking the objects created by the factory and processing them into sentences.

SimpleNLG ships with a default lexicon but also provides options for the developer to either import their lexicon or extend the one given. Since this application only needed a few words not already included in the lexicon, an extension of the default with these words was sufficient (the brand names, "Apple" as an adjective and not just a noun, among others).

4.3 Application

This section provides brief descriptions of the individual activities and fragments the application consists of.

4.3.1 MainActivity

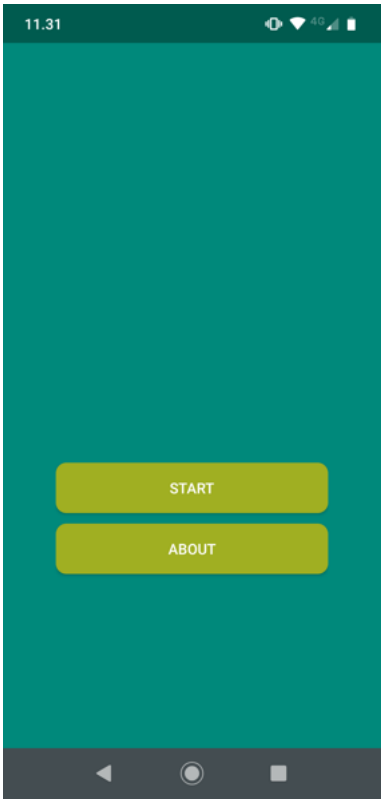


Figure 6: MainActivity screen in the application

The MainActivity is the first activity the user sees upon opening the application, containing two buttons that lead to the AboutFragment and the SummaryActivity, respectively. The AboutFragment is of little interest, its only job is to display the version number, hence why it is not included in this list by itself.

4.3.2 SummaryActivity

The SummaryActivity is the activity that has the responsibility for the bulk of the work of the application. The user interface consists of an ImageView, an

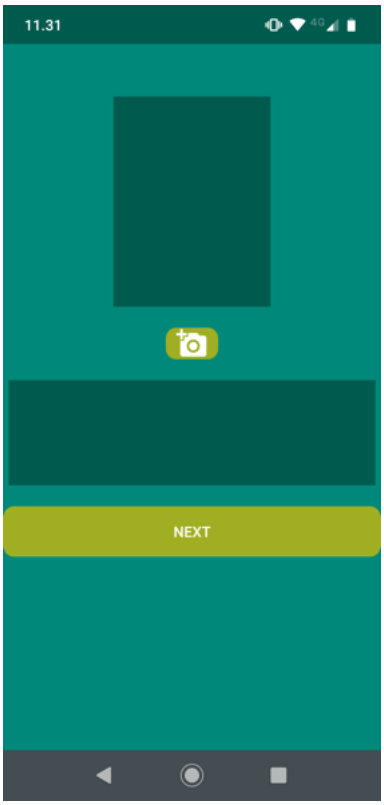


Figure 7: SummaryActivity screen in the application

ImageButton, a TextView, and a Button. This activity is the heart of the application, where the user can take a picture as well as view the final description of the image.

Beyond what is visible to the user, this activity is also responsible for running the ML models on the image, as well as gathering up all the extra information about the object in the image, and finally creating the descriptive sentence to be output to the user through SimpleNLG. The easiest way to describe all the functionality is through a description of the flow:

When the activity starts, all the UI elements bind to local variables. The currency code is also run at this point, as it requires location services to determine the geographical location of the device, and that is not desirable to perform in the middle of the construction of the sentence later on.

Following this, the creation and initialization of the AutoML models and labelers happen as described in Section 4.1.2.

The user may now take a picture, which appears in the ImageView, and the labelers run on this bitmap image. The labelers each return a list of FirebaseVisionImageLabels, which becomes input for the method responsible for creating the initial description. The initial description acts as a confirmation dialog for the user, as Figure 8 shows. The TextView prompts the user with the label and asks the user to either take a new picture if the labeler was wrong or press the Button labeled 'Next' to move on if it is correct, which inflates the ExtraInfoFragment. The ExtraInfoFragment collects more information about the item (that could not be found automatically in the image), and transfers control back to the SummaryActivity.

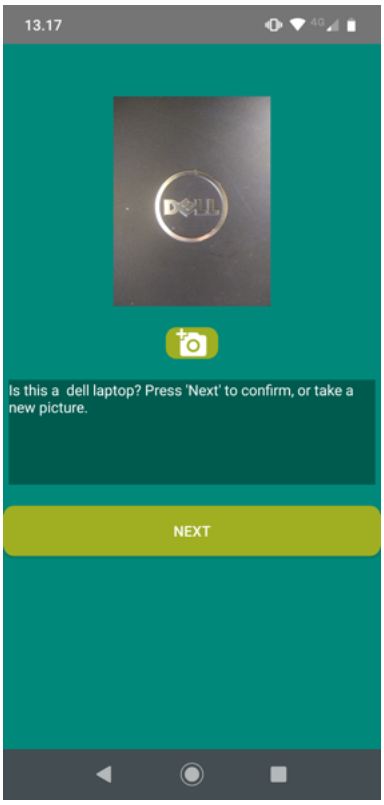


Figure 8: The initial description is output in the TextView of the SummaryActivity.

Next is the generation of the description of the item in the image: The SummaryActivity now has the currency (based on the location of the device) as well as the condition, price and model (input by user) along with the device-label (describing item and brand), and the color-label. Using this information, Sum-

maryActivity requests a description from SimpleNLG, which it then displays in the TextView.

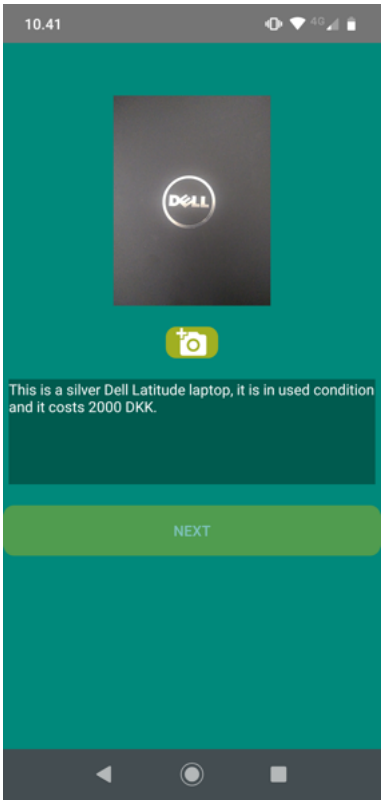


Figure 9: The final description is output in the TextView of the SummaryActivity.

Finally, Figure 9 shows the created string, displayed for the user in the TextView.

4.3.3 ExtraInfoFragment

This fragment is responsible for gathering more information for the construction of a descriptive sentence for the image. The fragment consists of two Spinners. In the first the user selects from a few pre-determined possible conditions for the item. In the second Spinner, the user selects from a few possible series/models for the detected brand or "None", if the model of the device in the picture does not match any of the listed models (If the detected device brand is Android, the user selects from several popular brands of Android phones rather than selecting models from a single brand, as "Android" is not a brand and the desired

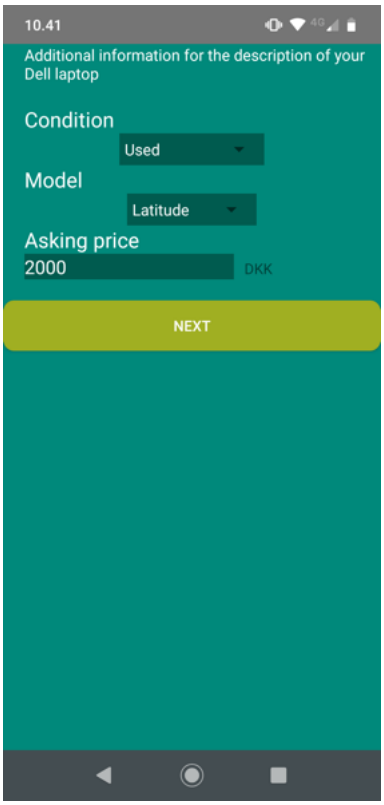


Figure 10: The ExtraInfoFragment screen in the application.

outcome is a more fine-grained description, compared to the starting point of "Android"). There is also a TextEdit in which the user may input a numerical value as the asking price for the item. The application detects the appropriate currency based on device location and displays that for the user. SummaryActivity receives all this information when the 'Next' button is pressed.

4.3.4 Picture

The Picture class is a simple class used primarily to generate unique filenames for the pictures taken by the application.

4.3.5 PictureUtils

PictureUtils provides a few methods that are handy for resizing the picture when it is fetched from internal storage and put into the ImageView in SummaryActivity [12].

4.3.6 SimpleNLG

The SimpleNLG class is responsible for handling both the initialization of the lexicon, factory, and realizer, as well as using them to create the image description for the SummaryActivity. Section 4.2 provides a detailed description of the NLG process.

5 Testing

This section describes the testing of the application.

5.1 Testing Options

Several testing methods are available for testing of Android applications, including traditional Android unit testing and instrumentation testing, Firebase Test Lab, and usability testing.

Traditional Android unit testing using JUnit and Espresso has many advantages, the primary being that it is the recommended way of testing Android applications, and as such has plenty of documentation and examples of tests.

Firebase provides their own test lab, which allows the developer to record and run UI crawler tests on a wide variety of physical devices, as well as run Espresso tests through Firebase Test Lab.

Usability testing allows the collection of ideas and issues that often aren't obvious to the developer who knows the application well. It is a test of the application as a whole by a user who has no pre-existing knowledge of the application and thus not a test of the individual modules.

5.2 Testing

Experimentation with Firebase Test Lab showed issues identifying UI elements with Robo Test, the automated UI crawler. Using Robo Test would require major changes to the UI throughout the entire application. Further, this application is relatively small and doesn't have many test cases, so there was no advantage in exporting and running the tests remotely, as Android Studio is sufficient for this.

The JUnit, Espresso, and Robolectric frameworks were used in testing the application, as well as manual navigation and verification by user testing.

The test suite for the application consists of a mix of traditional unit tests for the modules that do not need to call the Android SDK, being primarily that of the SimpleNLG class. MainActivity was tested using the Robolectric framework, which runs tests directly on the JVM rather than through an emulator or device.

The instrumentation tests utilize the JUnit and Espresso frameworks. They test the navigation between activities as well as the correct inflation of fragments on button presses. Conflicts with the Firebase-based functionality as well as conflicts with permissions meant that recording a test using Espresso was not possible.

Usability tests performed with a small user group ensured the application's flow made sense. The users had no prior knowledge of the application's intended functionality and/or appearance and thus no bias, which made it possible to collect thoughts about possible UI improvements from them.

Three users tested the application, the key points were:

- 2/3 users tried to input the currency along with the asking price. The ExtraInfoFragment needs to display the detected currency.
- 2/3 users did not read the Toast saying nothing was detected in the image, it needs to be adjusted to a longer duration.
- 3/3 users pressed "Next" after seeing the final description output. The app should make the user aware that they are done at this point.

Based on these points the application was modified to be more intuitive to users, by adding the currency detected to the ExtraInfoFragment, giving the error Toast a longer duration, and finally disabling and changing the opacity of the "Next" button after the final description has been provided.

6 Reflection

This chapter reflects on the project, with regards to the model precision problems experienced, as well as the use of Firebase ML Kit in general.

6.1 Model precision

As Section 4.1.1 described, both the device model and the color model are often confused in their results. The device model shows significant problems with telling Android and Apple smartphones apart because they are often very similar in appearance. Another issue is the accurate detection of Acer laptops. The model often confuses this brand for others. For the color model, the issue is

mostly detecting the difference between silver and white.

However, testing with new data proved that the datasets for both models were inadequate. For instance, the color model confidently determines a black phone or laptop as being silver, and likewise the device model often confidently determines a Dell laptop as being a Lenovo. This is an error stemming from the fact that things like different lighting conditions and many different angles of each type of device are not accurately represented in the datasets the models were trained on. The color model thinks black devices are pitch-black, and often they appear dark grey in pictures taken in broad daylight, leading the model to conclude that they are silver. In other words, the models are bad at generalizing to new input images.

The imprecision of the device model is mitigated somewhat by the confirmation screen that prompts the user to either accept the detected brand or take a new picture and try again.

The addition of more training data might solve the problems. 100 training examples are not enough, and increasing the amount of training examples would likely lead to better results. The color model needs more shades of each color, however, this might lead to lower confidence, as it blurs the lines between each distinct color. Similarly, more images would improve the device model's ability to generalize for new input data. An option for generating more training data, and at the same time training the neural network to be less affected by distortions in the new images, is using augmentation and applying geometric transformations to the training data, which creates more variants of an image.

Figure 11: Unedited, affine transform and scaled versions of the same image

Figure 11 shows some of the possible ways one can transform a single image to make a model more robust to variations in the input.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import glob
import os

directory = "C:/Users/miche/Desktop/dataset"

for f in os.listdir(directory):

    version = 1

    img = cv2.imread(os.path.join(directory, f))
    rows, cols, ch = img.shape

    pts1 = np.float32([[50,50],[200,50],[50,200]])
    pts2 = np.float32([[10,90],[150,50],[100,300]])

    M = cv2.getAffineTransform(pts1, pts2)

    dst = cv2.warpAffine(img, M, (cols, rows))

    filename, fileext = os.path.splitext(f)

    if not cv2.imwrite(os.path.join(directory, filename + str(version) + fileext), dst):
        raise Exception("could not write image " +  os.path.join(directory, f, str(version)))

    version = 2

    M2 = np.float32([[2,0,0],[0,2,0]])
    res = cv2.warpAffine(img, M2, (cols, rows))

    if not cv2.imwrite(os.path.join(directory, filename + str(version)+fileext), res):
            raise Exception("could not write image " + os.path.join(directory, f, str(version)))
```

```
~/Desktop/python-trial/aft.py [unix] (14:30 22/04/2020)
"aft.py" [unix] 38L, 992C written
```

Figure 12: Script for applying two kinds of geometric transformations to an image, and storing them in the same directory

Using OpenCV for Python, writing a small script for generating two additional variations of every image in a given directory is a relatively easy task. Figure 12 shows a script that creates an 'affine transformation' version (which alters the image along three points while preserving parallel lines in the image) as well as a 'scaled' version (which 'zooms' in on an area of the image) of each image [11]. Applied to the original dataset of 800 images, the new dataset (which also had the black and white versions of both Android and Apple classes merged into one, as a separate model responsible for detecting color now exists) contains 1700 images.

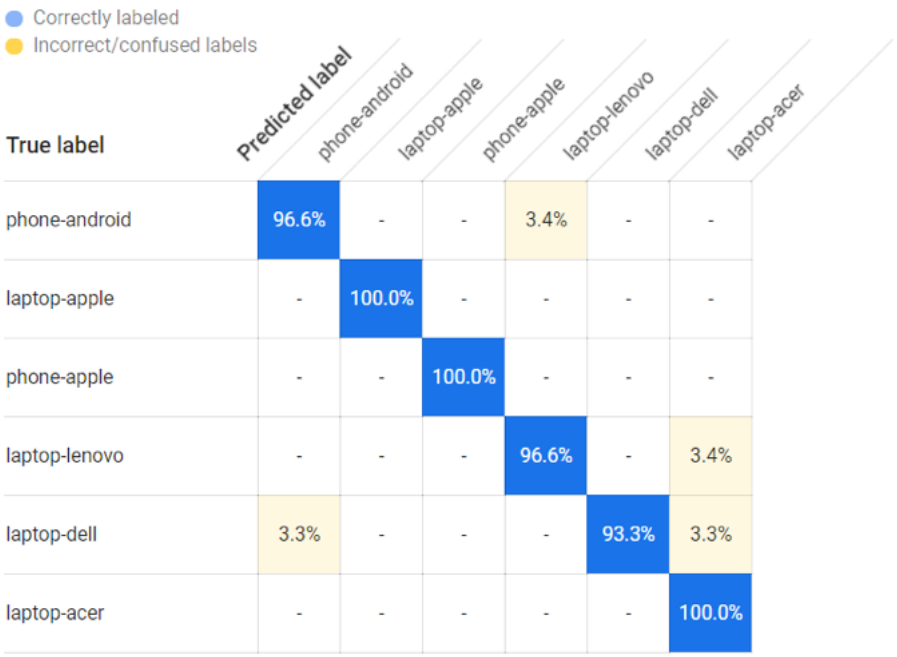


Figure 13: The new confusion matrix for the model

Figure 13 shows the new confusion matrix for the device model. Comparing this to Figure 4, it is evident that the model performance has improved. Using the application on devices the model previously misidentified also showed the improvement in accuracy.

6.2 ML Kit

Firebase ML Kit provides several machine learning models for use in Android and iOS applications. As this project shows, even if the software project needs a model that ML Kit doesn't directly provide, it is possible to tailor it to most needs. The process of creating custom models with ML Kit is easy, simply supply a dataset and the desired classes, and ML Kit trains the model. Even easier is the process of using the ready-made models provided by ML Kit, which is in most cases a matter of a few lines of code added to the project. Thus, creating Android or iOS applications with machine learning functionality is simple, even for developers new to the field of machine learning. An additional bonus is the ability to host and run test suites through Firebase Test Lab, which wasn't relevant for this application, but might be for larger applications.

Some of the downsides of working with Firebase ML Kit include the fairly limited free plan of the service. Even a simple application quickly grows to need

more training data or more training time than given, which may discourage some from experimenting with it. ML Kit is also, at the time of writing this report, still in beta, with some of its tutorials and example code being slightly outdated or lacking in general. Further, ML Kit makes no promises about backward compatibility, which may discourage creating production applications based on its features at present.

7 Conclusion

The aim of this project has been to get acquainted with machine learning principles and use them to solve the problem of image classification.

The first part of the project examines machine learning from a top-level perspective and accounts for supervised learning problems and the subcategories of regression problems and classification problems, followed by an account of unsupervised learning problems with the subcategories of clustering problems and association problems, and the differences between them. Following this, the project goes into depth with the concepts relevant to solving the problem of identifying an item in a picture, namely classification and, neural networks.

The project then describes the process of implementing an Android application with machine learning and natural language generation functionality, provided by Firebase ML Kit and SimpleNLG, respectively. It includes an evaluation of different ways of utilizing Firebase ML Kit for solving the stated problem and explains the process of using AutoML Vision Edge for creating a custom model, with classes tailored to the application's needs. It further describes all the fragments and activities of the application and what functionality they provide.

The final part of the project evaluates the process, with regards to the model trained and implemented, and provides some insight into how the initial model was improved upon as well as reflects on the use of Firebase ML Kit in general.

References

[1] K. van Deemter, M. Theune, and E. Krahmer. *Real vs. Template-based natural language generation: a false opposition?* URL: https://wwwhome.ewi.utwente.nl/~theune/PUBS/templates-squib.pdf. (accessed 19.03.2020).

[2] Firebase ML Kit. *AutoML Vision Edge*. URL: https://firebase.google.com/docs/ml-kit/automl-image-labeling. (accessed: 20.02.2020).

[3] Firebase ML Kit. *Evaluate your model*. URL: https://firebase.google.com/docs/ml-kit/train-image-labeler?authuser=0#evaluate_the_model. (accessed: 27.02.2020).

[4] Firebase ML Kit. *Firebase pricing plans*. URL: https://firebase.google.com/pricing. (accessed 19.03.2020).

[5] Firebase ML Kit. *Image Labeling Docs*. URL: https://firebase.google.com/docs/ml-kit/label-images. (accessed: 20.02.2020).

[6] Firebase ML Kit. *Label images with an AutoML-trained model*. URL: https://firebase.google.com/docs/ml-kit/android/label-images-with-automl. (accessed: 17.03.2020).

[7] Tom M. Mitchell. "Machine Learning". In: McGraw-Hill Science/Engineering/Math, 1997. Chap. 1.1. ISBN: 0070428077.

[8] Andrew Ng. *Machine Learning (Coursera course)*. URL: https://www.coursera.org/learn/machine-learning/lecture/1VkCb/supervised-learning. (accessed: 13.02.2020).

[9] Andrew Ng. *Machine Learning (Coursera course)*. URL: https://www.coursera.org/learn/machine-learning/lecture/wlPeP/classification. (accessed: 13.02.2020).

[10] Andrew Ng. *Machine Learning (Coursera course) Neural Networks - Model Representation I*. URL: https://www.coursera.org/learn/machine-learning/lecture/ka3jK/model-representation-i. (accessed: 02.04.2020).

[11] OpenCV. *Geometric Transformations of Images*. URL: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html. (accessed 22.04.2020).

[12] B. Phillips, C. Stewart, and K. Marsicano. *Android Programming: The Big Nerd Ranch Guide*. Big Nerd Ranch, 2017, p. 314. ISBN: 978-0134706092.

[13] E. Reiter et al. *SimpleNLG github repository*. URL: https://github.com/simplenlg/simplenlg. (accessed 19.03.2020).

[14] E. Reiter and R. Dale. *Building Natural Language Generation Systems*. Cambridge University Press, 2000. Chap. 3. ISBN: 0-521-62036-8.

[15] TensorFlow. *TensorFlow Lite models*. URL: https://www.tensorflow.org/lite/models. (accessed: 20.02.2020).