Automated Water Meter Reading Through Image Recognition

Undergraduate Student Project

by

Francis Serafin A. Bernardo
Electronics Engineering

Mith Lewis W. Concio
Computer Engineering

Justine Marcus A. Opulencia
Electronics Engineering

Adviser:

Jhoanna Rhodette I. Pedrasa, PhD
Gregorio L. Ortiz III

University of the Philippines, Diliman

July 2022

Acknowledgment

We would like to express our special thanks of gratitude to our project advisers Jhoanna Rhodette I. Pedrasa and Gregorio L. Ortiz III for their help and advice on the project, without which the project wouldn't have been possible. We would also like to thank the UP Diliman Electrical and Electronics Engineering Institute for giving us this opportunity to work on the project, and for the knowledge and experience it has granted us over the past four years.

Finally, we would also like to thank our parents and friends who helped us throughout this project, with the much-needed emotional support, as well as with the collection of local meter images.

Mith Lewis W. Concio
Computer Engineering

Francis Serafin A. Bernardo
Electronics Engineering

Justine Marcus A. Opulencia
Electronics Engineering

Abstract

Automated Water Meter Reading Through Image Recognition

Water meter readings within the Philippine setting are primarily done manually, which is error-prone, work-intensive, and lacks Internet of Things (IoT) integration. While effective, existing smart solutions for the problem, such as smart water meters, are costly and impractical to implement, as they would require overhauling the existing metering infrastructure. A popular alternative approach to this problem is to perform image recognition on the existing water meters using deep learning methods to streamline the meter reading process while minimizing cost. Utilizing this concept, this project has developed an end-to-end system for an Image-Based Automatic Meter Reader (AMR) as a proof-of-concept solution for local water utilities. The system includes an admin-side mobile application for taking and verifying images of the water meters to be read, a consumer-side mobile application used for viewing consumption data and contest erroneous readings, a centralized cloud database for collecting and cataloging all the data received, a website dashboard to view and edit the database, an image recognition pipeline for water meters, and an automated process that retrieves, reads and uploads the result to the database. The system primarily used openly available resources for its implementation, especially for the mobile application, database and dashboard development. On the other hand, the image recognition pipeline was constructed based on an already established architecture for image-based water meter reading and only used existing deep learning models and datasets available online. The resulting completed pipeline was found to perform well when used on the foreign meters it was trained with but performed comparatively poorly on local water meters that better reflect its actual use case. Ultimately, as a proof of concept, the constructed system showed that such a system was indeed feasible, and could potentially serve as a viable and cost-effective alternative for smart metering in the Philippine setting, with the only caveat being that the deep learning models utilized have to be trained using images of local meters rather than the foreign meters used in the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Within the past decade, the worldwide water demand has been steadily increasing due to factors such as the rapid population growth and widespread industrial development, which affect the sustainability of accessible clean water [1]. In response, various studies have been conducted to improve the existing water distribution process, with the most popular solution being the concept of smart water metering [2]. In the Philippines context, local water utilities still heavily rely on manual meter reading, wherein a designated meter reader must travel to the actual meter location and transcribe the reading by hand. This method is often considered inefficient due to the amount of manpower required to check each meter individually and also leaves room for human errors during the transcription process that may result in complaints from the end-users [3]. To combat these issues, smart metering infrastructures have been proposed as an efficient alternative that completely automates the meter reading process. One example was put forward by the Philippine-based company Hiraya Water, which offers a smart water meter that automatically collects data wirelessly and tracks the end-users' real-time water usage for leak detection, enabling a seamless billing system through an online platform [4]. Though convenient, these approaches can quickly become impractical to implement on a broader scale, considering the cost and manpower it would take to replace the current metering infrastructure, which would not be feasible in the Philippine setting.

In order to solve the issue with the limited resources available, the solution should ideally be compatible with the existing infrastructure and be comparatively cheaper to implement on a large scale. Image-based deep learning methods fulfill these exact criteria, becoming a cost-effective alternative to smart metering due to the increased accessibility of computing hardware capable of running them in recent years [5]. These methods are relatively inexpensive to develop and deploy and are usually fully compatible with the existing infrastructure. Utilizing this idea, this project proposes a proof-of-concept solution in the form of an end-

to-end system for automatic water reading, which includes a mobile application for data collection, a cloud database to store and retrieve data from, an image recognition pipeline for water meters that retrieves, and an automated process that retrieves, reads and uploads the result to the database.

Chapter 2

Related Work

2.1 Image-Based Automatic Meter Reading

The concept of Automatic Meter Reading (AMR) in water utilities has been thoroughly explored in recent years, especially with the emergence of smart water meters streamlining the process immensely. Though convenient, the use of smart water meter systems comes with the downside of overhauling and replacing the current metering infrastructure, which can quickly become impractical considering the cost, manpower, and lack of scalability when it comes to implementing it [6]. Thus, an alternative solution emerged in the form of Image-Based AMR, the key advantage of this approach is that it is less costly to implement compared to the former, as it can be utilized with the existing infrastructure. Due to its nature as an appealing cost-effective alternative to smart metering, Image-Based AMRs have been proven to be a tried-and-true approach and are fast improving, especially with the rapid improvement of computer vision and deep learning technology [7], [8], [5]. However, despite rapid advancements in the field, the approach used to implement these are far from decisive, with two key design aspects that commonly vary among implementations, and are the utilization and the architecture of the said Image-Based AMR, each with their unique sets advantages and disadvantages.

The method the Image-Based AMR concept is utilized can be generalized into two categories: via a mounted camera or a smartphone app. The mounted camera implementation, such as the one used by Naim et al. [8], involves installing a camera onto a water meter and which sends an image of the meter to the server to be read. This allows for continuous meter readings and eliminates the need for a manual water meter reading, just as with the smart water meter, and can be integrated to any existing water meter without issue. However, similar to the smart water meter, the issue of scalability emerges when it comes

to implementing it on a wider scale as every meter would require its own camera. Another concern is that the camera is prone to theft or tampering, as it is merely mounted on the meter and is therefore vulnerable to these types of attacks. On the other hand, is the smartphone app implementation, such as the one done by Hong et al. [7], which utilizes a smartphone to capture an image of the water meter and use the app to send the image to the server. It was found that water meter readings done this way have higher efficiency, reduce the overall labor required and eliminate human error compared to manual meter reading. However, unlike the previous approach discussed, this method still requires a meter reader to be present and cannot continuously monitor the meter readings.

Similarly, the architecture used for Image-Based AMRs varies among implementations, with many factors being taken into consideration, with the biggest being the application of the AMR. Before the widespread adoption of deep learning in the field of computer vision, Image-Based AMRs explored the use of image enhancement techniques and handcrafted features consisting of three stages: counter detection, digit segmentation, and digit recognition [9]. These methods were usually prone to errors due to noise and were generally not robust enough to handle images taken in uncontrolled real-life situations [10]. Thus, with the advancement in computer hardware and widespread use of graphics processing units (GPU), the use of deep learning methods was quickly favored over these methods as it has proven to be a lot more robust approach, and can handle test cases from a wide range of conditions. Compared to the former, deep learning-based models are a lot flexible in their execution and scope, as it is capable of predicting all the digits of the counter simultaneously [11], rather than being restricted to segmenting and then recognizing each digit individually, and also meter and counter recognition with the use of object detection [7]. Because of this, there is a lot of variation within Image-Based AMRs that utilize deep learning methods, from different techniques and architectures used, to the scope they are designed to operate, which will be discussed in the following section. Image-Based AMRs for dial meters have also been explored in recent years [12], though the main focus of the proposed study will solely be on digit-based meters.

Generally, the most widely used approach to building Image-based AMR is to split the problem into two stages: counter detection, which involves detecting the location of the counter within a given image, and counter recognition, which involves recognizing the digits within the counter. Implementations that take on this approach have been shown to be more robust and accurate compared other methods, thus its wide adoption within literature [5]. For counter detection, the common approach is to utilize object detection algorithms: Laroca et al. utilized Fast-YOLOv2 [13], Hong et al. utilized YOLOv3 [7], and Koščević and Subašić utilized Faster R-CNN [14] to detect counters on images of water meters, more recently, Laroca et al. utilized Fast-YOLOv4 to achieve the same [5], whereas Tsai et al. used a modified Single Shot Multi-Box Detector (SSD) for electrical meters [15]. Some approaches didn't explore object detectors, Calefati

et al.used a Fully Convolutional Network (FCN) for semantic segmentation to handle the counter detection [16]. As for counter recognition, Laroca et al. evaluated three CNN-based approaches, where it was found that the CR-NET model [13] outperformed the two segmentation-free models, Yang et al. combined an FCN and Connectionist Temporal Classification (CTC) in counter recognition [17], and Marques et al. used object detectors such as the Faster R-CNN and RetinaNet for counter recognition [18]. There was also research that delved into higher efficiency models for counter recognition, Li et al. [3] proposed a lightweight CNN which includes backbone network, spliced convolution, and disordered combination splicing, despite the reduction of network parameters, it was shown that this approach did not lower the recognition rate. Some implementations opted to skip the counter detection stage altogether, and the meter reading was done directly in the input images [19]. Though these approaches were proven to be capable of returning the right meter reading, they were not as robust as approaches that implemented the counter detection and were prone to errors due to perspective distortions. More recently, Laroca et al. [5] added a new stage after the counter detection stage in their approach, called corner detection and counter classification, which has been shown to improve the current two-stage approach by a significant amount, by improving its robustness and accuracy, especially in unconstrained situations.

2.2 Flutter

Flutter is a free and open-source software development kit (SDK) from Google which is used mainly for developing applications for multiple platforms using only one codebase [20]. This is possible through widgets, which are the building blocks of Flutter's architecture. Flutter-based apps are shipped with their own rendering engine called Skia [21]. Essentially, building the user interface (UI) components is independent of the device architecture, thus enabling a single application to run on both iOS and Android for example. The tradeoff for this flexibility in coding is performance, which becomes evident when the application involves a more complicated UI as compared to a natively built one. However, for lightweight applications, both implementations yield the same performance along with the ease of development [22], [23]. Flutter also integrates quite nicely with Firebase as the backend. It also has a lot of resources available for referencing as compared to using a different SDK such as React Native or Xamarin, given that both platforms are developed by Google [24].

2.3 Firebase

Firebase is an online platform that provides tools that are needed for a streamlined application development process for both mobile and web applications [25], [26]. Initially released in 2012, Firebase has been providing a backend-as-a-service (BAAS) tool for application developers and has been purchased by Google in 2014. Among many of its services are Firebase Authentication, Firebase Database, Machine Learning, Firebase Cloud Functions and many more [25], [27].

Firebase Authentication provides the backend service of securing the application by verification through user logins [25], [28]. This service can be applied either through the built-in User Interface (UI) library or through the Software Development Kit (SDK). Users can be authenticated through various methods such as:

- Email and password
- Federated identity providers (Google, Facebook, Twitter, GitHub, etc.)
- Phone and number
- Anonymous authentication
- Custom authentication system

Firebase Authentication allows multiple sign-in methods to be linked with one another such that one user can sign-in to the same account with Facebook, Google, or their phone number.

The Firebase platform also provides a cloud-hosted database through the Firebase Realtime Database. This database follows a NoSQL setup and stores and displays data in JSON format. uses data synchronization such that changes in the database are reflected across all connected users. It also features offline access for the application as the data from the Realtime Database is saved locally and is updated once the application reestablishes online connection [25], [28].

Finally, Firebase Hosting provides the hosting services needed to deploy a web application. With Firebase Hosting, content can be delivered from the local directories within a computer to Firebase's hosting servers. It allows both static and dynamic content to be hosted. Domain names are also freely provided and it allows a secure hosting service by having a built-in SSL configuration for all domains.

While other existing BAAS platforms are available such as Amazon Web Services, Back4App, and 8Base, the proposed system will be using Firebase for its realtime updating of the database, its ease of use, and its

free to use services. As such, the proposed system will be using Firebase Authentication and Firebase Real-time Database for the mobile application while the web application will be using Firebase Authentication, Firebase Realtime Database, and Firebase Hosting services.

Chapter 3

Problem Statement and Objectives

3.1 Problem Statement

Water meter readings within the Philippine setting are primarily done manually, which is error-prone [3], work-intensive [6], and lack IoT integration. While effective, existing smart solutions for the problem, such as smart water meters, are costly and impractical to implement, as they would require overhauling the existing metering infrastructure [29]. A popular alternative approach to this problem is to automate the act of meter reading by performing image recognition on existing water meters using deep learning methods, which is much more cost-effective. Utilizing this concept, the proposed solution for this project is to develop a system that streamlines the entire meter reading process by reducing the original multi-step operation of collection and manual transcription into taking a photo of the meter with a mobile application, and the system will handle the actual reading and the logging of data to a centralized database, which in turn also eliminates the possibility of human error.

3.2 Objectives of the Project

To alleviate the above mentioned problem, this project aims to develop and test an end-to-end system for an Image-Based AMR as a proof-of-concept solution for local water utilities. The proposed system will consist of the following:

- An admin mobile application that can be used to capture the image of the meter, send that image and other necessary data, such as the meter ID, date, time, and location to the centralized database, and view the all uploaded content within the database

- A consumer mobile application that can be used to view consumption data, contact customer service, and upload pictures for contesting erroneous readings
- A database that collects and catalogs the all the data received and automatically attributes all data collected to the correct end-user so that only they can view their data and not other end-users
- An automated Image-Based AMR pipeline that can pull image data from the database, extract the meter reading value from that image and forward the extracted data to the database
- A dashboard website for the database to display and manage all the data within it, to be used by the water utilities admin

3.3 Scopes and Limitations

Due to time constraints, the following scope and limitations will apply in developing the project's Image-Based AMR pipeline:

- The project will only focus on the implementation of existing Image-Based AMR architectures, which means improving upon these known methods or formulating new ones are outside the scope of the project
- Only openly available pre-built deep learning models will be utilized in building the pipeline, rather than recreating a known model or creating a brand-new model from scratch
- The deep learning models used within the pipeline will only be trained using open-source datasets that are available online, which may not reflect on the type of water meters used in the Philippines.

Additionally, the mobile application and database development will only use methodologies and functionalities that are available for free on the chosen platform.

Chapter 4

Methodology

4.1 System Architecture

The system architecture for this project can be seen in Fig. 4.1. The consumers, water meter operators, and water utility administrators will have access to a unique mobile application using an account assigned for each of them. Each application gives access to different functionalities and features depending on which type of user was logged in. Furthermore, as an added quality-of-life feature, the water utility administrators will also have access to a web application that can be accessed through a website dashboard that allows them to manage the database efficiently. All of the back-end services such as authentication, database, and file storage are handled by Firebase. On the other hand, the data processing and the pulling and pushing of data to the database are all facilitated by a local machine. These tasks were automated with a batch run file using the built-in task scheduler in the machine, wherein unprocessed data is pulled from the database, processed, and then re-uploaded by batch at a set time every day. The data processing performed to extract the reading data from the meter images was done using deep learning methods built using TensorFlow.

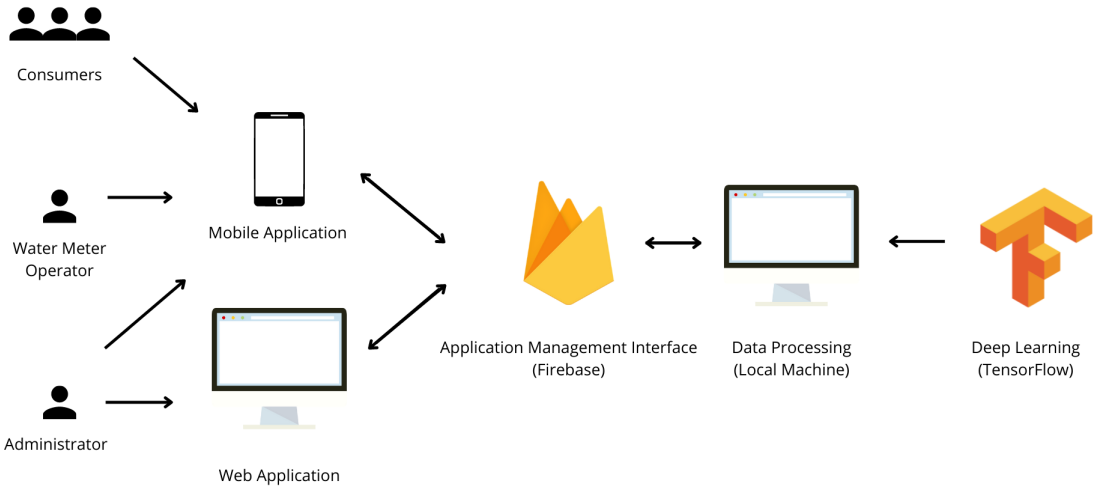

Figure 4.1: System Architecture

4.2 Image-Based AMR

4.2.1 Pipeline Architecture

According to the literature, the optimal approach to building an image-based AMR is to utilize deep learning methods [7], [8], [5]. Image recognition systems that utilize deep learning models generally exhibit better overall performance and robustness, with resistance to noise, distortion, and changes in angle and perspective. The current state-of-the-art also suggests splitting the task into two stages to improve the overall performance of the AMR, namely the counter detection and counter recognition stages [5]. The counter detection stage involves detecting the location of the meter counter within a given image, while the counter recognition stage involves identifying the digits within the counter. As for implementation for both of these stages, TensorFlow was used as the deep learning library of choice for its ease of use and thorough documentation. Implementing both stages with the same deep learning library also comes with the added benefit of compatibility and easy integration since both can be created and run on the same Python environment without issue. Aside from this, the Image-Based AMR pipeline also consists of a pre-processing stage before each of the above mentioned stages and a final post-processing stage at the very end. The complete pipeline architecture is illustrated in Fig. 4.2.

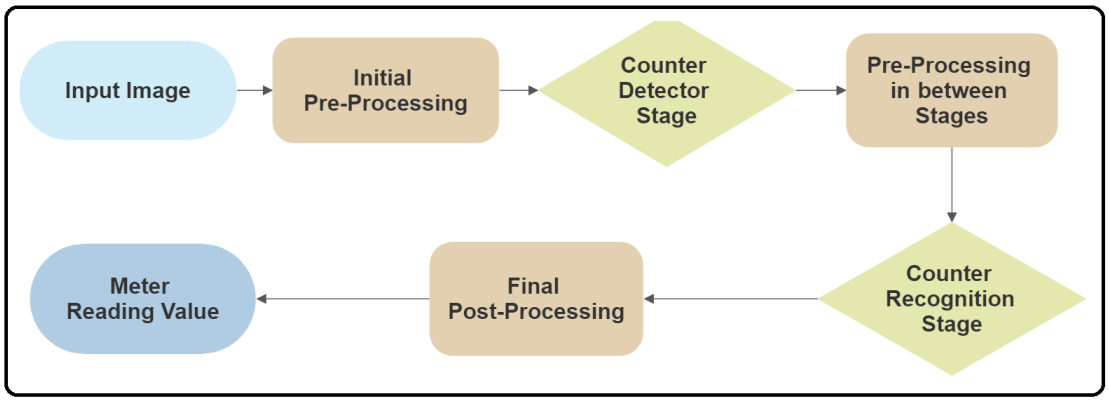

Figure 4.2: Architecture of the Image-Based AMR Pipeline

4.2.2 Image Datasets Utilized

The project used two open-access image datasets to train and test the deep learning models within the pipeline. The primary dataset utilized in the project was the open-source dataset assembled by Roman K. called the *Water Meters Dataset* [30]. The database consists of 1244 unique images of water meters taken from different perspectives and angles, with the file name of each image containing the value of the meter within it, as well as a spreadsheet containing the coordinates of the meter counter for each image. The dataset was also partially annotated by Olivier K. for object detection, adding individual annotations for each digit within the meter counter[31]. Though the dataset was taken from a foreign environment that used different meter models compared to the ones used locally, the meter models are similar enough for the deep learning model to gain inference. The variety of meter models within the dataset also comes with the added benefit of better emulating the Philippine setting, as water meters used in the Philippines also do not share a universal standard model and allow the deep learning model trained with it to account for the differences that arise from different meter models. This dataset was utilized as the training data for both the pipeline's counter detection and counter recognition stages and as the validation data for the complete pipeline. The secondary dataset utilized was the UFPR-AMR Dataset created by Laroca et al., which was used to train an image-based electric meter reader [13]. The dataset consists of 2,000 unique images of electrical meters taken from different perspectives and angles, with an individual text file for each image containing the annotations for the meter counter and each digit within it. This dataset was used to supplement the training dataset for the counter recognition stage of the pipeline. In addition to the aforementioned open-source datasets, a new dataset consisting of 107 unique images of water meters of various models found within the local area was manually collected and compiled by the project's proponents. This dataset, hereby referred to as the PH Meters Dataset, has all the meter reading values of the images compiled in spreadsheet, but

the images were not annotated in any form. The PH Meters Dataset was only used for the validation of the complete pipeline, and was not used during training due to its small sample size and lack of annotations due to time constraints. A summary of the datasets utilized and when they were utilized can be found in Table 4.1.

	Counter Detection Training	Counter Recognition Training	Complete Pipeline Validation
Water Meters Dataset	X	X	X
UFPR-AMR Dataset		X	
PH Meters Dataset			X

Table 4.1: Summary of Datasets Utilized

4.2.3 Counter Detection

Based on the literature, creating an image segmentation model is the most efficient deep learning method for the counter detection stage [16], [31]. The openly available segmentation models library [32] was used to implement the said model, which was chosen for its high level of abstraction and because it was built with the TensorFlow infrastructure. With the segmentation models library's available image segmentation resources, a binary segmentation model was created with the U-Net [33] architecture using a Resnet34 [34] backbone, the schematic for this architecture can be seen in Fig. 4.3.

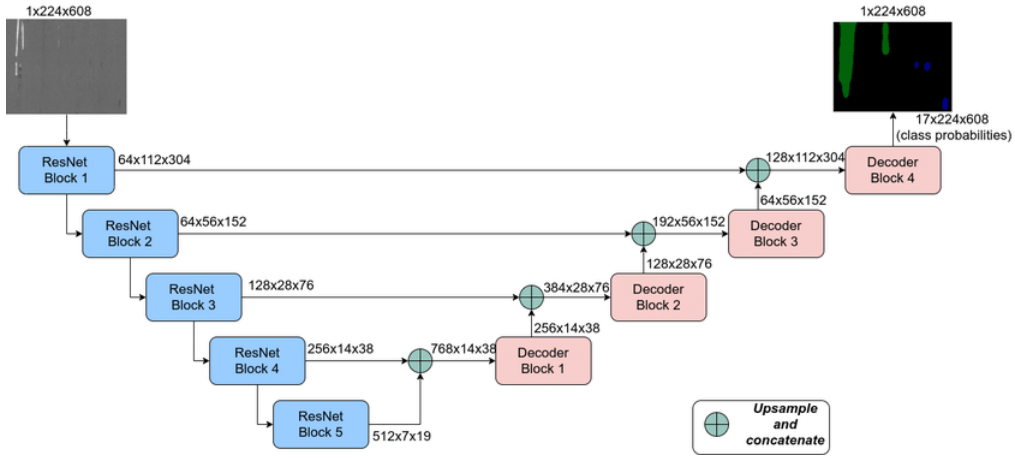

Figure 4.3: Schematic Illustration of the Unet Architecture with a ResNet Backbone, image taken from [35]

This configuration was chosen for its relatively light model architecture and because it has been demon-

strated to perform well for the counter detection task [31]. The binary segmentation model was trained using the *Water Meters Dataset*. In order to train the model, individual mask files were created for each image using the annotation spreadsheet data that showed the exact location of the meter counter, as shown in Fig. 4.4.

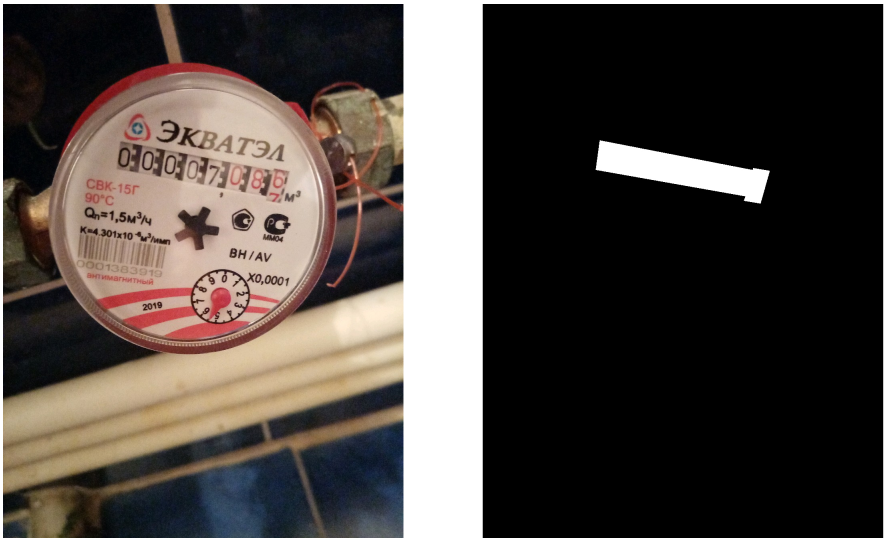

Figure 4.4: Sample Image-Mask Pair, meter image taken from [30]

Furthermore, the images and masks also had to be resized to 244 by 244 pixels as that was the maximum size allowed for the model and were split into training and validation sets using the 70:30 ratio commonly employed when training a deep learning model. The images were then pre-processed using the available pre-processing function in the library for models built on a Resnet backbone before all the data was loaded through the library's dataloader. Finally, the model was created with its default parameters and training configurations and was trained for 50 epochs. The training was configured to monitor and save the model weights of the epoch with the highest test accuracy, which was found to be around 98% at the end of the final epoch.

4.2.4 Counter Recognition

According to the literature, the counter recognition stage is usually achieved using an object detection model [5], [18]. The TensorFlow Object Detection API [36] was used to implement the model. Just as with the counter detection stage, this option was chosen for its high level of abstraction and because it was built with the TensorFlow infrastructure. Within the models available in the API, the Faster RCNN [37] architecture with a ResNet101 [34] backbone was selected to implement an object detection model for the

project, the schematic for this architecture can be seen in Fig. 4.5.

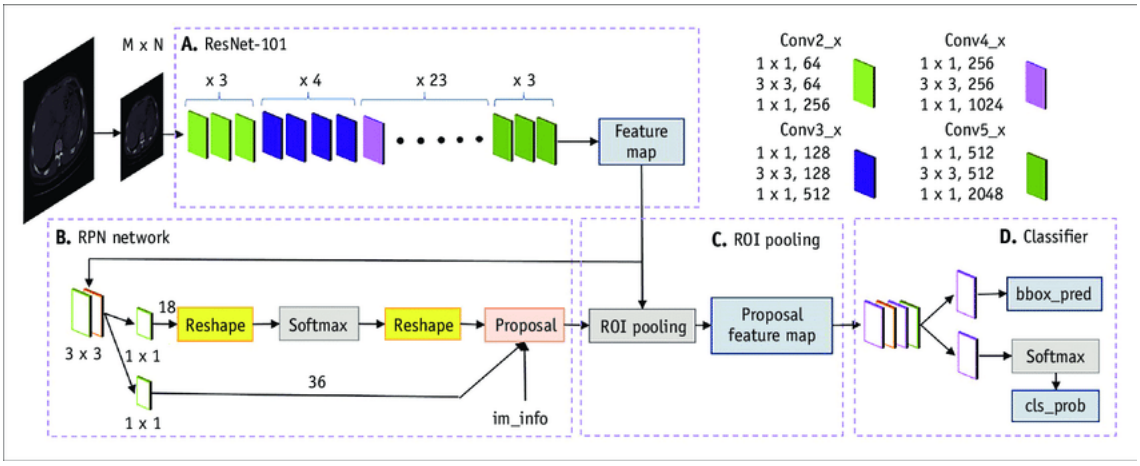

Figure 4.5: Schematic Illustration of the Faster R-CNN Architecture with a ResNet101 Backbone, image taken from [38]

Faster RCNN was chosen as the object detection architecture for the project for its higher accuracy compared to other architectures in exchange for a slightly longer inference time than average [39], which is an acceptable trade-off since the model was not designed for real-time object detection and its performance will primarily be based on its accuracy. On the other hand, ResNet101 was chosen as the backbone as it had the best balance between accuracy and inference time. According to the documentation, compared to its lighter counterpart ResNet50, it gains an increase in accuracy of 5% in exchange for a 4% increase in total inference time, while the heavier model after it, ResNet152, only exhibits a 2% increase in accuracy for a 16% extra inference time[40]. In addition, the ResNet101 also has a larger memory requirement and a longer training time compared to ResNet50 since it has 101 layers while the latter only has 50. However, this does not pose much of an issue as the model is intended to run on a machine that could handle the large model size of the ResNet101.

The object detection model was trained using the *Water Meters Dataset* in conjunction with the UFPR-AMR Dataset. As mentioned before, the *Water Meters Dataset* was only partially annotated for object detection, with only 621 of 1244 images of the water meters having annotations for each digit, which may be insufficient data for the model to gain enough inference for precise predictions. To overcome this limitation, the deep learning principle of transfer learning was used, which dictates that if a model is trained for a task similar to the main objective, it carries over the experience it gained for that task and results in better overall performance for the main objective. In the case of this project, this was achieved by training the object detection model using the large UFPR-AMR Dataset first before training it again with the more

limited *Water Meters Dataset* to fine-tune the model for water meters.

In preparation for training, both datasets were split into training and validation using the same 70:30 ratio mentioned earlier. The UFPR-AMR Dataset also needed each image to be cropped only to contain the meter counter. Furthermore, all the images had to be cropped to have a maximum width of 640 pixels, as that was the maximum allowable size for the model, and the annotation files for these images had to be adjusted accordingly. After this, all the annotation files had to be converted to the API's required format before they could be used for training. Once these were completed, the datasets were fed using the API's dataloader to commence training. No further pre-processing was necessary as the API handles all extra processing and data conversions. The model was created with its default parameters and training configurations and was trained for 50 epochs with the UFPR-AMR Dataset first before training it again under the same conditions with the *Water Meters Dataset* for another 50 epochs.

4.2.5 Assembling the Pipeline

With both stages and their respective models now trained, the pipeline for the image-based AMR was assembled. The pipeline begins by creating a copy of the original image and resizing that copy to 244 by 244 pixels, which is then pre-processed and fed into the image segmentation model of the counter detection stage. The image segmentation model then outputs a mask for the predicted location of the meter counter. Before this output can be utilized for the counter recognition stage, a few pre-processing steps are necessary, as the mask alone cannot be used for inference. The mask first has to be resized to the original image's dimensions before being overlaid onto the said image as a separate image. This step was done to preserve the image quality of the original for a more precise inference with the object detection model later on. The resulting overlaid image is then checked for skewness using the deskew [41] library, and both original and overlaid images are rotated to correct that skew. The original image is then cropped based on the overlaid image by using both axes' maximum and minimum pixels to isolate the meter counter within the original image. Afterward, the resulting image is put into the object detection model of the counter recognition stage, which returns a set of items detected and their respective classes, prediction confidence value and coordinates. In order to extract the final meter reading, the items have to be shortlisted by discarding those with a prediction confidence value of below 70%. In addition to this, as the counter detection stage is not capable of distinguishing decimal and whole digits from the meter counter, these parameters would have to be provided by the operator of the pipeline. Items that completely overlap in coordinates are decided based on whichever has the higher prediction confidence. They also have to be sorted by their order of appearance from left to right, as the object detector sorts them according to prediction confidence by default. If the remaining entries exceed the number of expected counter digits, the excess is discarded starting from the

rightmost position. Finally, once this has been completed, the items are split according to the number of expected whole and decimal places before being converted to a float using their respective class, which serves as the final meter reading value. The schematic of the complete Image-Based AMR Pipeline is shown in Fig. 4.6.

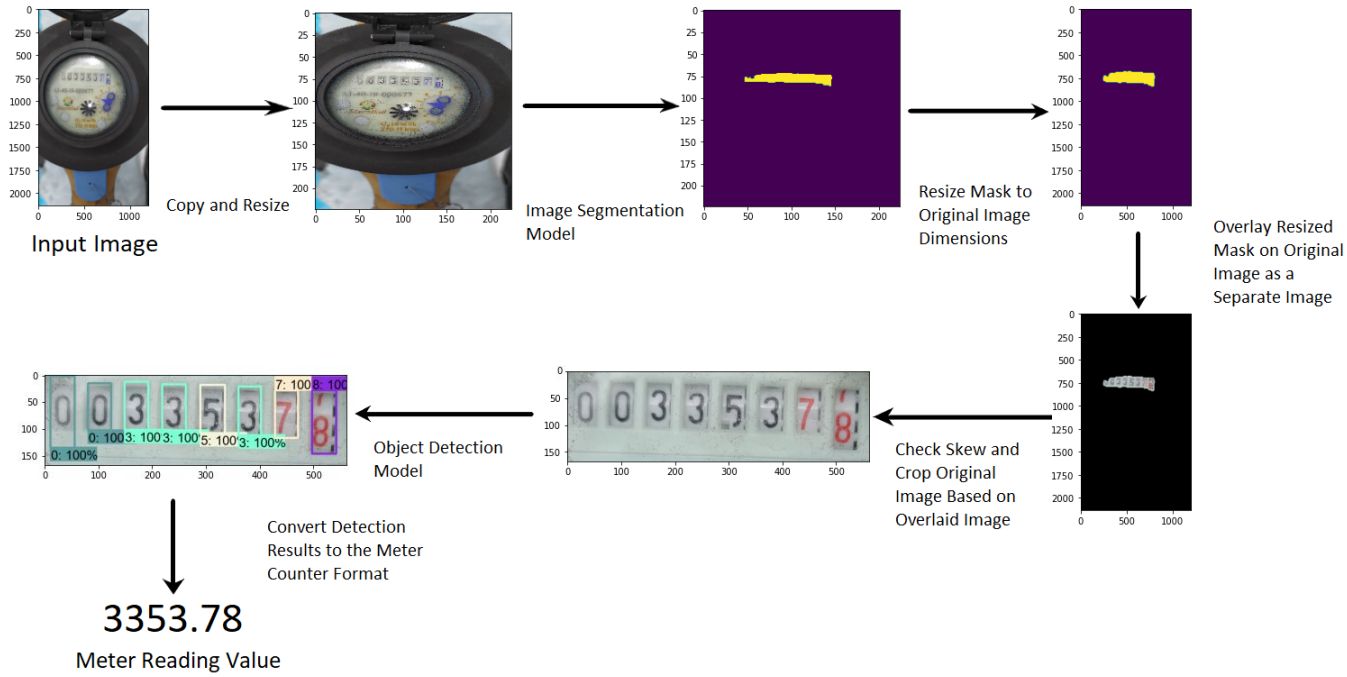

Figure 4.6: Complete Image-Based AMR Pipeline

4.2.6 Assessing the Pipeline

In order to assess the performance of the complete image-based AMR pipeline, it was validated with images of foreign water meters similar to what it was trained with and images of Philippine water meters, which are the actual use case for the pipeline. The *Water Meters Dataset* was used to validate the pipeline's performance for foreign water meters. Only the portion of the dataset not used to train any of the models used in the final pipeline was used for validation to avoid any forms of bias within the result. As for the case of Philippine water meters, the entirety of the PH Meters Dataset was used for validation with no restrictions, as none of the images within it was used during the training process. Unlike foreign meters, Philippine water meters that differ in meter model may also differ in the number of digits within the counter and the number of decimal places; thus, those pieces of information need to be added to the label of each image for the pipeline to perform as designed.

4.3 Flutter-Based App Development

Flutter was used to develop the mobile application for the end-users, which are classified into three groups—consumers, water utility administrators, and water meter readers. Each target group has a specific set of in-app functions available to use, and these are summarized in the use case diagram shown in Fig. 4.7. On the other hand, Fig. 4.8 shows the constructed home screens for the consumer, admin, and meter reader sides, respectively.

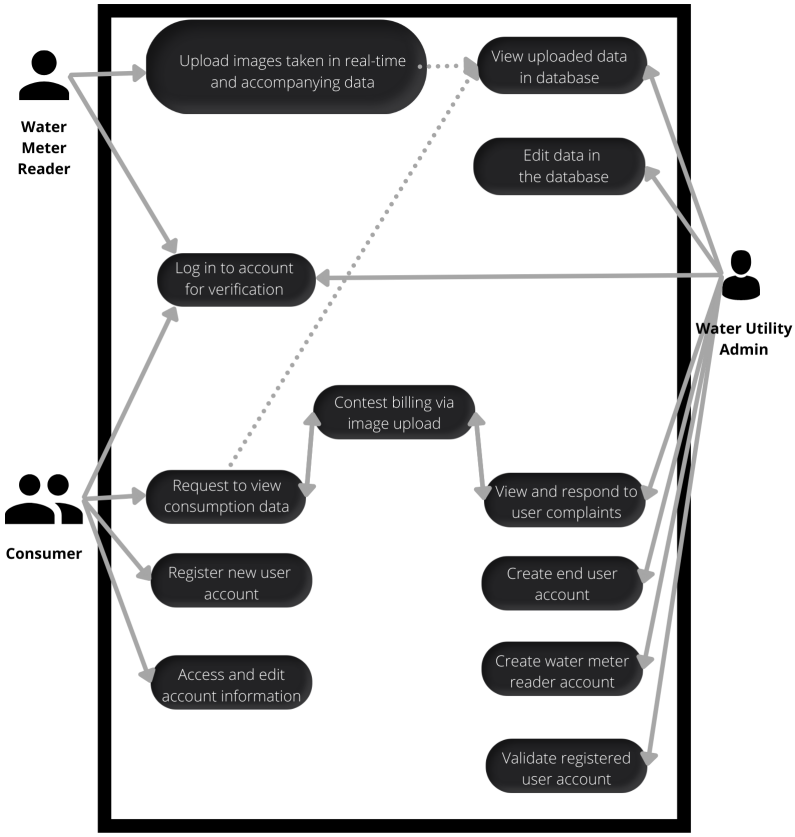

Figure 4.7: Use Case Diagram

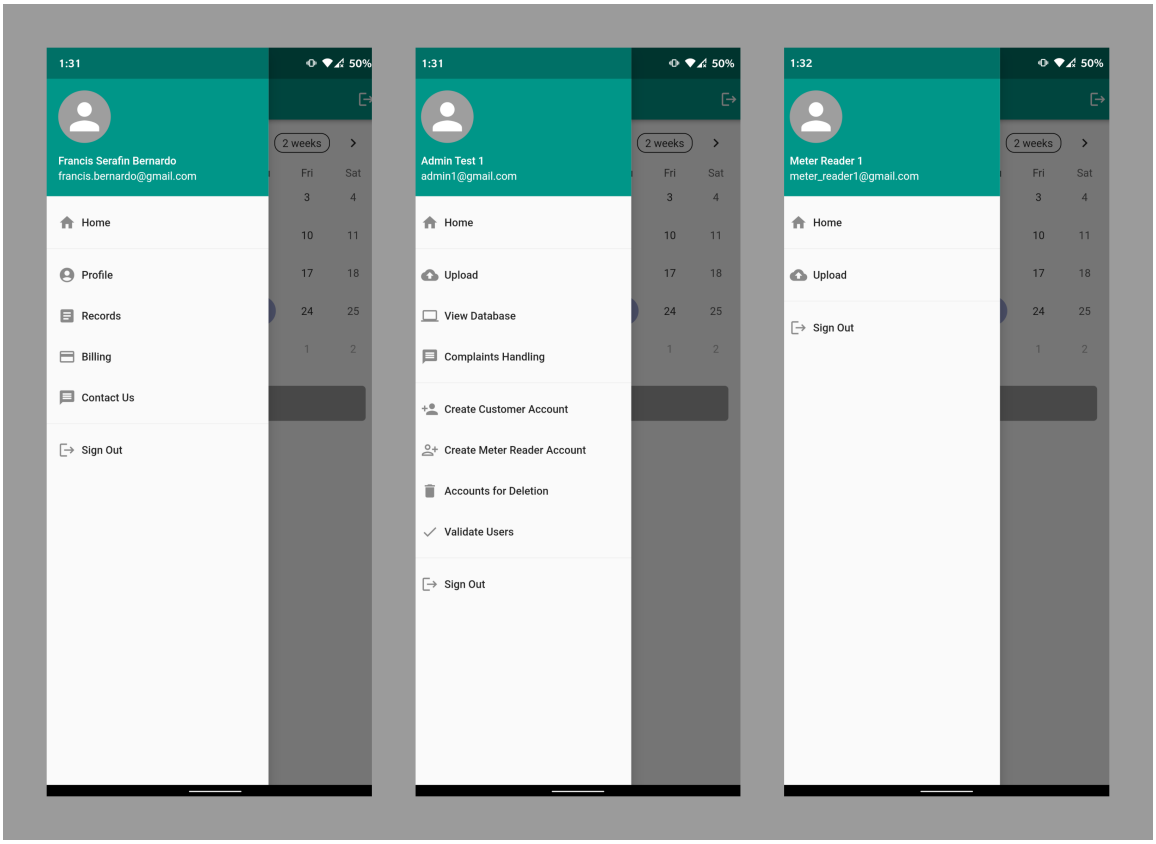

Figure 4.8: App Home Screens

4.3.1 Consumer Functionality

The consumers, also known as the end-user, must register an account using their email address and water meter number to use the app. Given that meter numbers are easily obtained from the physical meter itself, newly registered end-users are restricted from accessing the app's full functionality until an admin verifies the legitimacy of their account in order to maintain security and data privacy. Once validated, end-users can manage their account information, such as their name, contact number, email address, and password. They can also view the images taken for their monthly meter reading, including essential details such as the date and time it was uploaded to the server and the location it was taken. There is also an option to view a soft copy of their monthly billing statement. In the event that an end-users notices an error in the meter reading or the bill, a contact information page is made available for them to submit a complaint ticket addressing their issue, along with an optional image field for proof.

4.3.2 Admin Functionality

For the admin side of the app, the handlers will be assigned an account with elevated privileges and a different set of functionalities to manage the database. Through the app, the database management option gives a complete list of registered users displayed based on their meter numbers to make it easier to find specific entries compared to a name-based system. A search function was also implemented to account for an extensive user base. Upon selecting a meter number, the associated account information, water meter image records, and billing statements are available for viewing or editing. As mentioned in the previous section, consumers can forward their complaints through the app, and the admin side receives them as a list. Each complaint entry contains the consumer's name, the associated meter number, the description of their issue, and the submission date. Once the complaint is addressed, they can mark it as resolved to remove it from the list. The last set of functions is involved with account management. Admins can create accounts for consumers and meter readers. There are also options to view the list of new accounts waiting to be validated and the accounts for deletion. Accounts can only be queued for deletion under the database management option. The actual deletion process must be done through Firebase itself, as these functions are not made available in the packaged application for authentication security.

4.3.3 Water Meter Reader Functionality

In contrast with the other two account types, meter reader accounts are set to have only the image uploading function. Since the water meters in the Philippines vary in design, the meter number needs to be indicated, along with the total number of digits and the number of whole numbers, denoted as black digits, on the dial. Choosing an image requires the reader to launch the camera from the app while connected to the internet to ensure that the photo is taken at the prescribed reading date. Once the image is uploaded, essential details such as the name of the meter reader, the time and date it was taken, and the location are appended to the database entry. This upload function is also present on the admin side of the app. Fig. 4.9 shows the implemented upload screen.

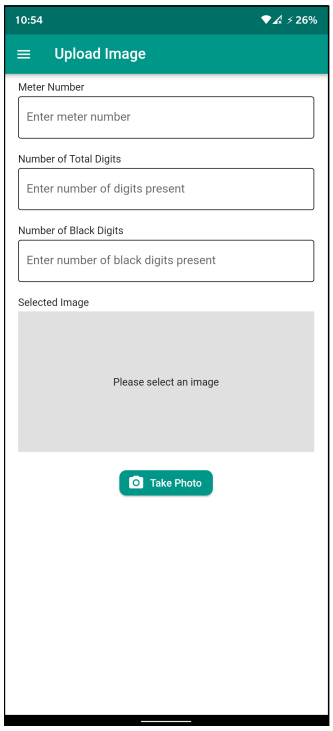

Figure 4.9: Meter Upload Screen

4.4 Firebase Database and Storage Structure

As mentioned, the database used for the project was implemented using Firebase as a backend which can be accessed through the mobile and web applications. Both platforms support the editing and upload of data within the database. Furthermore, the Firebase console serves as the website dashboard for the web application that can be reached via logging into Firebase using an authorized Google account. The database was divided into three main collections—*users*, *complaints*, and *admin uploads*. The *users* collection contains all user data, cataloged according to the unique UID assigned by Firebase when the user was registered. Each document contains the key information about the user, such as the name, verification status, and type of user, to name a few, as shown in Fig 4.10.

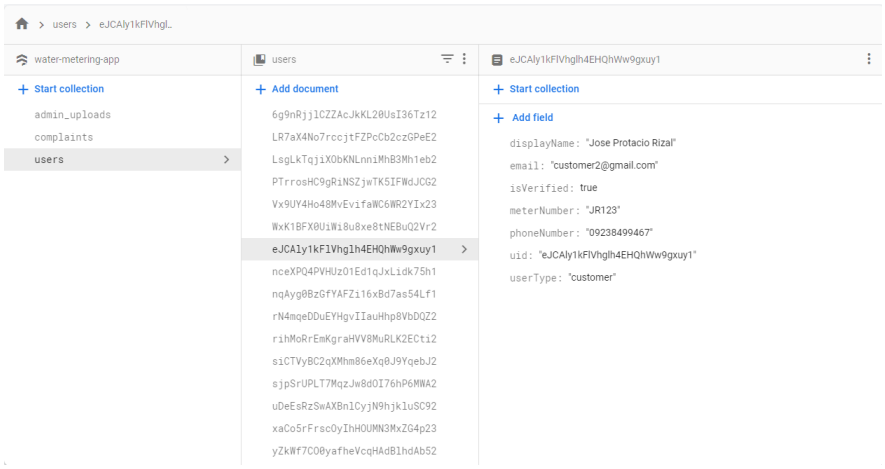

Figure 4.10: Firebase Database Structure - Users Collection

On the other hand, the *complaints* collection contains all complaint information logged when a customer submits a complaint ticket. Each document is named based on the unique complaint ID issued when the complaint is submitted. Each document contains the key information about the complaint, including the date and time it was filed, the meter number, and the end-user who filed it, among many others, as shown in Fig. 4.11.

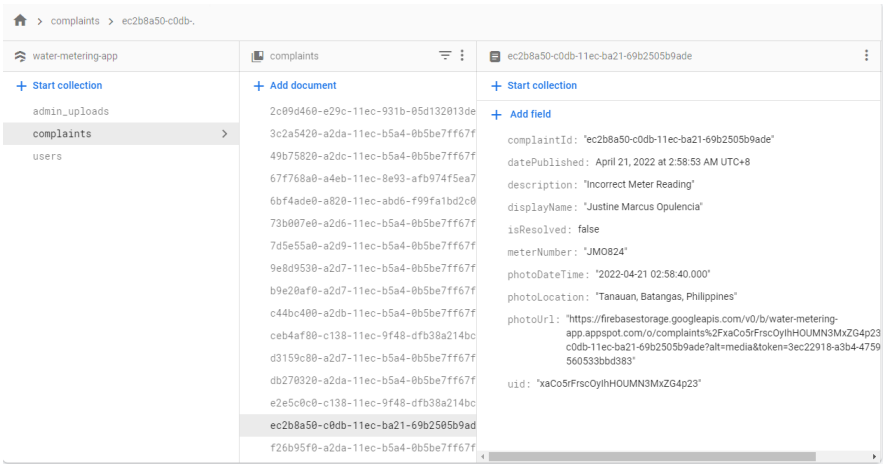

Figure 4.11: Firebase Database Structure - Complaints Collection

Finally, the *admin uploads* collection contains all the end-users' reading and billing information, and all catalogued according to the customer's water meter number. Each directory named after a meter number has two sub-directories called billings and readings, which contain documents related to the reading and billing information for that specific meter, the required fields are shown in Figs. 4.12 and 4.13 respectively.

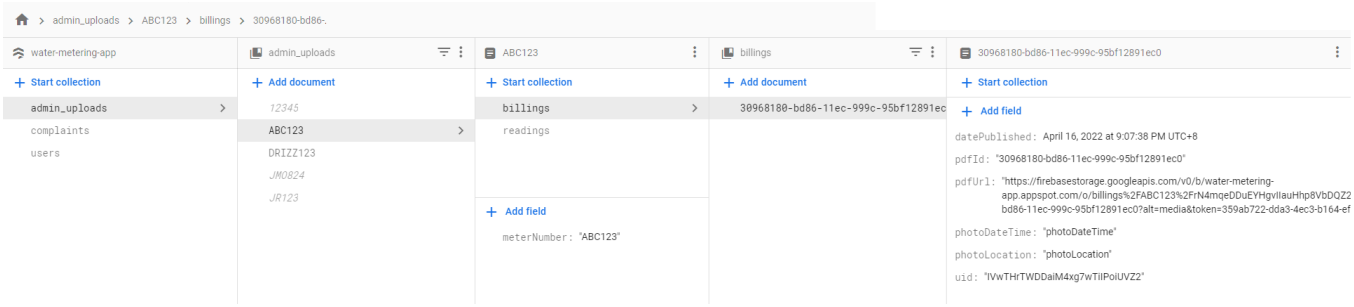

Figure 4.12: Firebase Database Structure - Admin Uploads Collection, Billings Folder

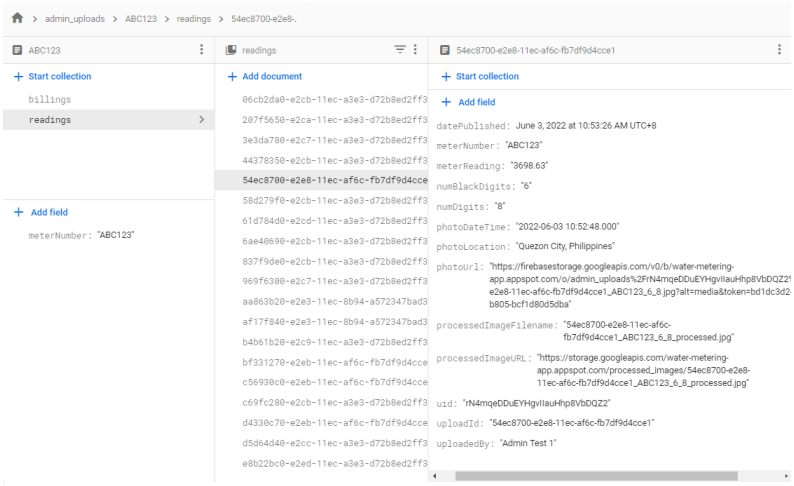

Figure 4.13: Firebase Database Structure - Admin Uploads Collection, Readings Folder

The files and images for the database are stored in Firebase Storage within different folders—*admin uploads*, *users*, *complaints*, *for processing*, and *processed images*. The *admin uploads* folder contains the images uploaded by water meter readers. The image files are stored under a sub-folder named after the UID of the operator that took it. The *billings* folder contains all the files related to the customer billings that are stored under different sub-folders named after the water meter number of the end-user. Likewise, the *complaints* folder contains all the images uploaded through the complaints function, which are stored separately for each end-user under a sub-folder named after their UID. The last two folders, *for processing* and *processed images* are used during the processing of the meter images. The *for processing* folder contains a copy of all the admin uploaded images that have yet to be processed to keep track of the images pending processing. On the other hand, the *processed images* folder stores images of the meters that have been processed, including the detection results, to be viewed by the admin and end-user. During processing, as the local machine processes the images by batch, the resulting images are uploaded to the *processed images*

folder as they are finished. At the end of the batch processing, the *for processing* folder is cleared of its contents to reset the queue.

Chapter 5

Results and Analysis

5.1 Image-Based AMR

As mentioned before, the pipeline's performance was assessed with foreign water meters, which it was trained with, and Philippine water meters, which better reflect the actual use case of the image-based AMR. The primary goal of this approach was to compare the performance of the pipeline used under conditions it was and was not trained with. The pipeline's performance was primarily based on its meter reading accuracy, using the following formula:

$$Accuracy = \frac{Number\ of\ Correct\ Test\ Cases}{Total\ Number\ of\ Test\ Cases} \tag{5.1}$$

It is also worth noting that the counter detection stage was found to have a 100% accuracy for both datasets, and none of the extra processing done before or after the two main stages contributed to an error. Thus all erroneous test cases when discussing the pipeline accuracy are purely from the counter recognition stage. That said, this does not mean that the rest of the pipeline operated as planned. The deskew library was found to only detect skewness within 30 degrees clockwise and counterclockwise for water meter counters, likely because it was designed for pure text and thus struggled when used on the digits within the meter counter. This problem was overcome by checking the dimensions of the cropped meter counter image. Suppose the height is greater than the width by a certain ratio. In that case, the overlaid and original meter images are rotated by an amount determined by the ratio, and the deskew and cropping process is repeated. However, even this approach has its limitations, as there is no way for the pipeline to identify if the resulting meter counter has the correct orientation and is not upside down. This issue was alleviated during testing by

comparing the meter reading result with the expected value. If the predicted result is not the same, the meter counter is rotated by 180 degrees, and the meter reading is attempted again. Whichever of the two readings was closer to the expected value is then chosen as the final meter reading value. It is important to note that this method was only viable during testing as the actual value of the meter reading was known beforehand. Thus, it would not work in the actual use case of the pipeline where this information is unavailable.

Three separate test metrics were used to assess the pipeline's accuracy: reading accuracy for all digits on the counter, for whole digits only, and for individual digits. The first metric is used to measure the accuracy of the pipeline meter reading for the entire meter counter. A correct test case for this would require the pipeline to correctly predict all digits on the counter. The second metric is fairly similar to the first, except the decimal places are omitted in the evaluation. This metric is used to gauge the viability of the pipeline for commercial use since water utility companies normally only consider the whole numbers when it comes to billing. Finally, the last metric splits every digit within the meter counter as a separate test case and assesses the accuracy of the pipeline for individual digit predictions. This metric was conceived to measure its performance in a vacuum, as the other metrics only consider correct predictions for a group of digits rather than for each digit separately, which may oversimplify the actual performance of the pipeline. The corresponding results of all three test metrics for each set of meters are tabulated in Table 5.1.

	Foreign Meters	Philippine Meters
All Counter Digits	62%	38.5%
Whole Numbers Only	91.5%	75%
Individual Digits	94.5%	84%

Table 5.1: Meter Test Metrics

Based on the accuracy results, the pipeline's accuracy was significantly lower when used for local water meters compared to foreign meters on every test. These results were expected as the deep learning models within the pipeline were trained only with foreign meters that are similar but not identical to the type of meters used locally, which causes a decrease in performance as models struggle to account for the differences in features between these meters.

Another critical observation shared between both foreign and local meter results is the drastic difference between the accuracy for the whole numbers only meter reading compared to the ones for all counter digits. This indicates that the bulk of the erroneous digits read are from the decimal digits of the meter counter and that the position of the counter digits affects its reading accuracy. While there is no definitive method to determine the reason why this error occurs as the deep learning model abstracts most of the image

processing, based on the erroneous test cases observed, it is likely because the decimal digits are more likely to be in-between numbers since those digits naturally move faster when in use and makes them more difficult to get a precise reading. One more notable observation is that the pipeline's per digit accuracy is much higher than either the all counter digits or the whole numbers only meter reading. It demonstrates that gauging the accuracy of the pipeline based on how well it can read a group of digits does not reflect the overall performance of the deep learning model utilized, as the error rate for individual digit readings compounds when multiple digits are involved resulting in a higher error rate. This idea is further supported by the fact that 80% of the errors found for the all counter digits readings in the foreign meter test case are only incorrect for one digit, which was usually the final digit. Similarly, for the local meters test case, 40% of the errors for the all counter digits readings only got 1 digit incorrect, and another 40% only got 2 digits incorrect, all of which are usually the last few digits of the meter counter.

In addition to the quantitative data presented, a few qualitative details have to be discussed to fully understand the pipeline's performance on local meters. As mentioned before, the decrease in the pipeline's performance is derived from the limitations of the training dataset, such as the differences between local and foreign water meter models and the conditions of the meters within the dataset. Among all the erroneous test cases observed, the differences within the meter composition were the most significant factor in the pipeline's poor performance for local meters. Many of the local meters are constructed with a meter dial at the edge of the meter counter, as shown in Fig. 5.1, which partially obstructs the last two decimal digits that causes the pipeline to either misread or entirely miss them, samples of which can be seen in Fig. 5.2.

Figure 5.1: Sample Problematic Water Meter Model

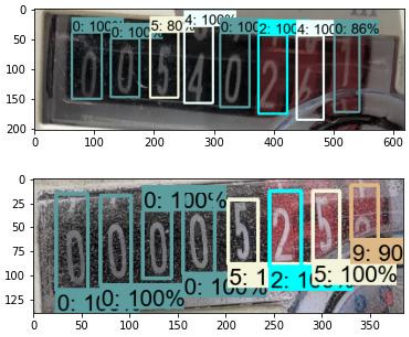

Figure 5.2: Example of Incorrect Readings due to the Meter Dial

Furthermore, within these same meters, an extra space exists between the meter counter and the meter body that gives way to the possibility of false detection of a counter digit, as shown in Fig. 5.3.

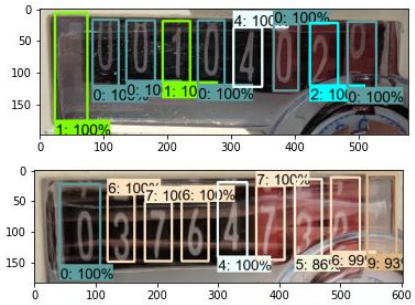

Figure 5.3: Example of Incorrect Readings due to the Extra Space

On the other hand, the condition of the local water meters was another contributing factor in many of the pipeline's errors. It was observed that many counter digits were misread or undetected due to the dust and dirt that tended to collect on local water meters that were not present in the training dataset, an example of which can be observed in Fig. 5.4.

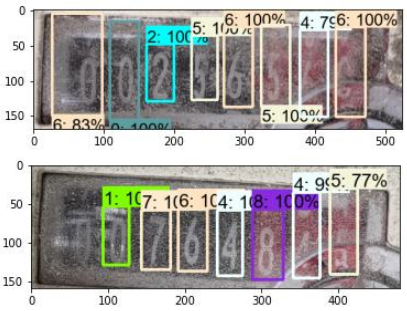

Figure 5.4: Example of Incorrect Readings Due to Dirty Meter Counter

Despite the pipeline's shortcomings due to the use of foreign meters in the training dataset, this has also given the pipeline a strong foundation for many of its tasks, as seen in the pipeline's perfect accuracy for the counter detection and an 84% accuracy for individual digit readings. On top of this, the pipeline has also been shown to be capable of reading the digits of multiple different meter models used locally, and it generally performs well on test samples that do not have any of the problematic factors mentioned before, as demonstrated by the examples in Fig. 5.5.

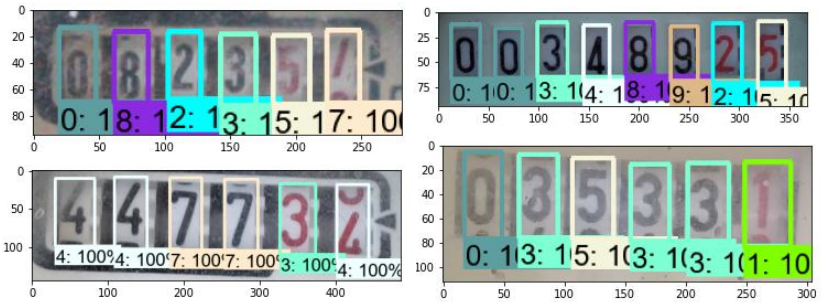

Figure 5.5: Example of Correct Readings on Different Unique Models

Overall, the performance results demonstrate the feasibility of the proposed pipeline Image-Based AMR for the task it was designed for. However, the pipeline's performance for local meters is insufficient for its actual use case due in large part to the limitation of the dataset used for training. In order to be feasibly deployed within the Philippine setting for the proposed end-to-end system, the pipeline would have to be trained with local meters to overcome the aforementioned issues and maximize its performance.

5.2 Flutter-Based App Development

This project aims to implement a fully-functional system using a defined set of platforms for each sub-system. There is no definitive method of conducting the quantitative analysis of the mobile application's performance due to the lack of available benchmarks for different Android development frameworks such as React Native or Xamarin for comparison. Thus only qualitative analysis was conducted based on its performance from an end-user's perspective.

Three main aspects of the mobile application were evaluated. The first was the app's performance on an actual smartphone. As an Android Package File (APK), the app was only 25 megabytes (MB) in size and took up around 42 MB when installed. It is designed to be lightweight to ensure that it will operate smoothly even on older devices. Furthermore, it was confirmed to run as intended upon testing different devices with varying Android versions and processing power. In order to achieve this result, the animations and transitions between pages were kept to a minimum, as using anything other than Flutter's default rendering engine would require a larger app size. The second mode of analysis was the app's connection with the back-end, which mainly focused on how well the app can communicate with the database, whether it be updating or pushing new entries or simply accessing them through the app. It was found that simple data such as strings and timestamps load instantly, whereas images take a few seconds to download or upload fully. This result has been found to largely depends on the quality of the user's internet connection rather than an issue with the app itself. The third and most important metric is user authentication security. As a security measure to avoid spam or fake accounts with actual meter numbers, the registration of accounts within the database does not allow for duplicate emails. In addition, another layer of security is provided through a mandatory verification step which locks the app's functionalities until the user's identity can be confirmed to be legitimate. An end-user can access their data only once they are verified, as these records may contain sensitive information such as emails and contact numbers. Overall, based on the analysis conducted, the final developed application was found to be without issue, both from a security and performance point of view.

5.3 System Integration

The proposed system was initially designed to incorporate real-time processing by utilizing Google's cloud computing platform to automate the extraction of the meter reading from the uploaded images and update the database accordingly. Unfortunately, the actual deployment of the two deep learning models utilized in the Image-Based AMR pipeline to the cloud was found to be beyond the project's budget. Due to this,

some changes were made to the implementation of the system. Rather than processing the images directly within the cloud, a local machine was used to pull the necessary data from the database at a scheduled time each day and are processed by batch within the same machine. Once completed, the processed images are re-uploaded back to the storage in specific folders, and the resulting readings are appended to the corresponding database entries. The entire process was automated through Windows Task Scheduler and done through the use of a batch run file.

The implementation for the local machine portion of the system was found to be not too resource-intensive and only requires around 660 MB of disk space to store the Image-Based AMR pipeline. A large part of the required memory comes due to the two deep learning models, with the image segmentation model and object detection model requiring 287 MB and 369 MB, respectively. In addition to this, the images to be processed by the pipeline would also have to be accounted for once it is deployed, which would have to be determined by the water utilities admin depending on the number and quality of the images to be used. As for the total processing time of the system, initialization time for the deep learning models was observed to take around 10 seconds, while the processing time for each image takes approximately 4 seconds. Nevertheless, these values may vary depending on the availability of a GPU and the internet speed of the local machine, and are factors that will need to be taken into consideration if the system was deployed. Overall, the new approach was found to be significantly cheaper to implement than the original and proved to be just as efficient, with the only caveats being that the data now has to be processed by batch rather than immediately after the image was uploaded and the processing now has to be done on a specified machine rather than outsourcing the workload through a third-party server, which is an additional point of failure.

5.4 Cost Analysis

The cost incurred in the proposed end-to-end system can be divided into two categories: hardware and software. For the hardware portion of the system, to avoid complicating the discussion, the assumption is that the system will not be utilizing the existing infrastructure for the local machine part of the implementation. In the project implementation, the local machine used had a Ryzen 7 5800H CPU with an RTX 3060 laptop GPU. To achieve similar performance statistics on the local machine for the proposed system, a machine with an Intel Core i3-12100F CPU paired with an RTX 2060 GPU should suffice based on their specifications [42], [43]. In the current Philippine market, a device with these components would cost around 27,000 PHP [44]. The cost of the other peripherals of the machine is not accounted for in this discussion since the machine's intended use case is solely to handle the data processing as a server device. According

to Maynilad, the company had around 1.4 million water service connections during the time of writing and is the assumed number of end-users for the cost analysis of the system [45]. Given the number of images to be processed each month, and assuming that the processing time for each image is still 4 seconds as in the project implementation, the amount of time it would take for a single machine to process everything would far exceed a month. However, this limitation can be overcome by utilizing several machines in tandem to distribute the total amount of processing to be done and significantly cut down the total processing time. The proposed number of machines is around five units running in tandem to ensure that the processing can be finished in less than half a day, assuming that the number of images is distributed uniformly while also providing enough leeway to handle double that amount. This brings the total amount on the hardware side to be around 135,000 PHP for the cost of the machines alone, not accounting for the cost to keep them running. Though the front-load cost for this implementation may be high, an equivalent system of 5 machines for a cloud-based approach would cost around 8400 PHP per day to deploy the deep learning models onto the server and does not even account for the cost of the actual processing [46]. Once totaled for a month, the cost of running the cloud implementation would surpass the cost of the local machine implementation.

On the software side of costs, the following calculations for the Firebase plan cost are made with the worst-case scenarios in mind, which could be more or less than the stated amount and can still be adjusted upon acquiring real-world usage statistics. The first cost to consider is for the Cloud Firestore, which would need to account for three main document writing processes, namely the initial account creation, the allotment of two complaint submissions per month, and the appending of the processed water meter image and the corresponding meter reading to the database entry. With 1.4 million end-users to consider, this would total to around 5.6 million writes per month, which will be doubled to 11.2 million in the cost calculation for the sake of extra leeway, and would cost around 1,120 PHP per month based on Firebase's Blaze plan calculator [47]. Furthermore, assuming that each user accesses their app at least once a day would yield 42 million document reads, which cost an additional 1,400 per month. The primary cost would be for the Cloud Storage, which would require 1.4 million images, each with a duplicate, to be stored in the database. Assuming that each image is 2.5 MB at most, the worst-case scenario is that the storage would require around 7,000 GB of space during each batch processing, costing 7952 PHP per month. Finally, accounting for the 1.4 million images to be uploaded and the functionality for the end-user to download their monthly billing from the database, adding these operations would require an additional 392 PHP per month. Combining all of these prices, the total cost of the proposed end-to-end system would be 135,000 PHP for the initial set-up of the local machines to be utilized and 10,864 PHP of monthly fees to maintain the backend server. Overall, much of the cost is derived from the initial investment of building the system and is relatively inexpensive to maintain considering the number of end-users it is intended to serve.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In the Philippine setting, local water utilities still heavily rely on manual meter reading, which is often considered inefficient due to the amount of manpower required and is prone to human error. While effective, existing smart solutions for the problem, such as smart water meters, are costly and impractical to implement. To alleviate this problem, the project leverages the capability of deep learning models to extract the meter reading value from an image to develop a system that completely streamlines the existing meter reading process. This objective was achieved by reducing the original multi-step operation of collection and manual transcription into taking a photo of the meter with a mobile application, and the system will handle the actual reading and the logging of data to a centralized database.

The system primarily used openly available resources for its implementation, especially for the mobile application and database development, which used Flutter and Firebase, respectively. On the other hand, the image recognition pipeline was constructed based on an already established architecture for image-based water meter reading, which suggests using a counter detection and a counter recognition stage to parse the meter reading information from an image. Additionally, only existing deep learning models and datasets available online were used for the pipeline implementation. The resulting completed pipeline was found to perform well when used on the foreign meters it was trained with but performed comparatively poorly on local water meters that better reflect its actual use case. Furthermore, the original plan for the data processing to be performed on the cloud was ultimately dropped in favor of using a local machine due to budgetary constraints. This change in architecture, while significant, did not affect the project's main objective much, with the only caveats being that the data now has to be processed by batch rather than

immediately after the image was uploaded, and the processing now has to be done on a specified machine rather than outsourcing the workload through a third-party server, which is an additional point of failure. Ultimately, as a proof of concept, the constructed end-to-end system proved to be a viable solution to the issue of outdated water meter reading methods that plagues many of the water utility companies in the Philippines and may potentially serve as a cost-effective alternative for smart metering, and though the Image-Based AMR pipeline's performance for local meters was found to be insufficient for actual real-life deployment, its performance for the foreign meters it was trained on demonstrated that the approach used was indeed viable, given that a suitable training dataset is used.

6.2 Recommendations

As mentioned throughout the paper, one of the most significant factors that kept the Image-Based AMR pipeline from being feasibly deployed in the local setting was the training dataset utilized, which only consisted of foreign meters. Future studies could alleviate this issue by training the deep learning models within the pipeline with local meters that better reflect the intended use case, and also account for the extra space within the meter counter when creating the mask for the image during the training of the image segmentation model. On a similar note, subsequent research on the topic could also focus on creating new deep learning models and redesigning the pipeline to better prediction on local meters. A model identification stage can be added that could identify the model of the meter, and use the amount of whole and decimal digits for that model rather than requiring these values be declared by the operator. Another notable improvement to the existing pipeline is the addition of the corner detection stage that was put forward by Laroca et al. [5] which was found to improve the pipeline's performance significantly but required extra annotations and a key-point detection model specifically designed for the task.

Another key recommendation for future work would be on the implementation side of the end-to-end system. Many potential features for the mobile application and the backend service could not be implemented due to the need for a paid subscription service to access them. These would include a variety of functions and APIs that would help in user account management and add more layers of authentication security. The project proponents also suggest exploring different combinations of Android frameworks and backend services to find a potentially more effective implementation and extend it to the IOS framework for increased accessibility. Lastly, due to budgetary constraints, the project had to abandon the idea of using cloud processing as its mode of performing data processing. Future work that could get past this constraint could explore the potential of this approach, as that would remove the need for a dedicated machine and the restriction of having to wait for the data to be processed by batch.

Bibliography

[1] M. T. H. van Vliet *et al.*, "Global water scarcity including surface water quality and expansions of clean water technologies", *Environmental Research Letters*, vol. 16, no. 2, p. 024 020, 2021. DOI: 10.1088/1748-9326/abbfc3. [Online]. Available: https://doi.org/10.1088/1748-9326/abbfc3.

[2] B. Coelho and A. Andrade-Campos, "Efficiency achievement in water supply systems—a review", *Renewable and Sustainable Energy Reviews*, vol. 30, pp. 59–84, 2014, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2013.09.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032113006692.

[3] C. Li, Y. Su, R. Yuan, D. Chu, and J. Zhu, "Light-weight spliced convolution network-based automatic water meter reading in smart city", *IEEE Access*, vol. 7, pp. 174 359–174 367, 2019. DOI: 10.1109/ACCESS.2019.2956556.

[4] "Hiraya water: Smart meter", *Hiraya Water*, [Online]. Available: https://www.hirayawater.com/smartmeter.

[5] R. Laroca, A. Belézia Araujo, L. Zanlorensi, E. Almeida, and D. Menotti, "Towards image-based automatic meter reading in unconstrained scenarios: A robust and efficient approach", *IEEE Access*, vol. 9, pp. 67 569–67 584, May 2021. DOI: 10.1109/ACCESS.2021.3077415.

[6] D. Quintanilha *et al.*, "Automatic consumption reading on electromechanical meters using hog and svm", Jan. 2017, 11 (5 .)–11 (5 .) DOI: 10.1049/ic.2017.0036.

[7] Q. Hong *et al.*, "Image-based automatic watermeter reading under challenging environments", *Sensors*, vol. 21, p. 434, Jan. 2021. DOI: 10.3390/s21020434.

[8] A. Naim, A. Aaroud, K. Akodadi, and C. El Hachimi, "A fully ai-based system to automate water meter data collection in morocco country", *Array*, vol. 10, p. 100 056, 2021, ISSN: 2590-0056. DOI: https://doi.org/10.1016/j.array.2021.100056. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005621000047.

[9] M. Vanetti, I. Gallo, and A. Nodari, "Gas meter reading from real world images using a multi-net system", *Pattern Recognition Letters*, vol. 34, Apr. 2013. DOI: 10.1016/j.patrec.2012.11.014.

[10] S. Liao, P. Zhou, L. Wang, and S. Su, "Reading digital numbers of water meter with deep learning based object detector", in *Pattern Recognition and Computer Vision*, Z. Lin *et al.*, Eds., Cham: Springer International Publishing, 2019, pp. 38–49.

[11] F. Yang, L. Jin, S. Lai, X. Gao, and Z. Li, "Fully convolutional sequence recognition network for water meter number reading", *IEEE Access*, vol. 7, pp. 11 679–11 687, 2019. DOI: 10.1109/ACCESS.2019.2891767.

[12] P. He, L. Zuo, C. Zhang, and Z. Zhang, "A value recognition algorithm for pointer meter based on improved mask-rcnn", in *2019 9th International Conference on Information Science and Technology (ICIST)*, 2019, pp. 108–113. DOI: 10.1109/ICIST.2019.8836852.

[13] R. Laroca, V. Barroso, M. Diniz, G. Gonçalves, W. Schwartz, and D. Menotti, "Convolutional neural networks for automatic meter reading", *Journal of Electronic Imaging*, vol. 28, p. 013 023, Feb. 2019. DOI: 10.1117/1.JEI.28.1.013023.

[14] K. Koščević and M. Subašić, "Automatic visual reading of meters using deep learning", *Proc. Croatian Comput. Vis. Workshop*, pp. 1–6, 2018.

[15] C.-M. Tsai, T. D. Shou, S.-C. Chen, and J.-W. Hsieh, "Use ssd to detect the digital region in electricity meter", in *2019 International Conference on Machine Learning and Cybernetics (ICMLC)*, 2019, pp. 1–7. DOI: 10.1109/ICMLC48188.2019.8949195.

[16] A. Calefati, I. Gallo, and S. Nawaz, "Reading meter numbers in the wild", in *2019 Digital Image Computing: Techniques and Applications (DICTA)*, 2019, pp. 1–6. DOI: 10.1109/DICTA47822.2019.8945969.

[17] F. Yang, L. Jin, S. Lai, X. Gao, and Z. Li, "Fully convolutional sequence recognition network for water meter number reading", *IEEE Access*, vol. 7, pp. 11 679–11 687, 2019. DOI: 10.1109/ACCESS.2019.2891767.

[18] R. C. da Silva Marques *et al.*, "Image-based electric consumption recognition via multi-task learning", in *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, 2019, pp. 419–424. DOI: 10.1109/BRACIS.2019.00080.

[19] L. Gómez, M. Rusiñol, and D. Karatzas, "Cutting sayre's knot: Reading scene text without segmentation. application to utility meters", in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, 2018, pp. 97–102. DOI: 10.1109/DAS.2018.23.

[20] "Flutter", *Flutter.dev*, [Online]. Available: https://flutter.dev/.

[21] "Skia in flutter fuschia", *User Documentation*, [Online]. Available: https://skia.org/docs/dev/flutter/.

[22] M. Olson, "A comparison of performance and looks between flutter and native applications", 2020. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1442804/FULLTEXT01.pdf.

[23] I. Demedyuk and N. Tsybulskyi, "Flutter vs react native vs native: Deep performance comparison", 2020.

[24] "Add firebase to your android project", *Firebase Documentation*, [Online]. Available: https://firebase.google.com/docs/android/setup.

[25] "Firebase", *Firebase Documentation*, [Online]. Available: https://www.firebase.google.com/docs.

[26] A. Rajappa, A. Upadhyay, A. S. Sabitha, A. Bansal, B. White, and L. Cottrell, "Implementation of pinger on android mobile devices using firebase", in *2020 10th International Conference on Cloud Computing, Data*

Science Engineering (Confluence), 2020, pp. 698–703. DOI: 10.1109/Confluence47617.2020.9058306.

[27] A. K. S, *Mastering Firebase for Android Development*. Birmingham, England: Packt Publishing, Jun. 2018.

[28] K. M. Hlaing and D. E. Nyaung, "Electricity billing system using ethereum and firebase", in *2019 International Conference on Advanced Information Technologies (ICAIT)*, 2019, pp. 217–221. DOI: 10.1109/AITC.2019.8920931.

[29] N. Jazdi, "Cyber physical systems in the context of industry 4.0", in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, 2014, pp. 1–4. DOI: 10.1109/AQTR.2014.6857843.

[30] R. Kucev, *Water meter dataset*, 2019. DOI: 10.21227/tdqj-cw20. [Online]. Available: https://dx.doi.org/10.21227/tdqj-cw20.

[31] K. Merric, *Water meters*, 2021. [Online]. Available: https://github.com/K-Merrick/water-meters.

[32] P. Yakubovskiy, *Segmentation models*, https://github.com/qubvel/segmentation_models, 2019.

[33] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", vol. 9351, Oct. 2015, pp. 234–241, ISBN: 978-3-319-24573-7. DOI: 10.1007/978-3-319-24574-4_28.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", vol. 7, Dec. 2015.

[35] R. Neven and T. Goedemé, "A multi-branch u-net for steel surface defect type and severity segmentation", *Metals*, vol. 11, p. 870, May 2021. DOI: 10.3390/met11060870.

[36] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors", Nov. 2016.

[37] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[38] Q.-Q. Zhou *et al.*, "Automatic detection and classification of rib fractures on thoracic ct using convolutional neural network: Accuracy and feasibility", *Korean Journal of Radiology*, vol. 21, Jul. 2020. DOI: 10.3348/kjr.2019.0651.

[39] J. Hui, *Object detection: Speed and accuracy comparison (faster r-cnn, r-fcn, ssd, fpn, retinanet and...* 2019. [Online]. Available: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359#.

[40] TensorFlow, *Tensorflow 2 detection model zoo*, 2021. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md#tensorflow-2-detection-model-zoo.

[41] S. Brunner, *Deskew*, 2022. [Online]. Available: https://github.com/sbrunner/deskew.

[42] 2022. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/132223/intel-core-i312100f-processor-12m-cache-up-to-4-30-ghz.html.

[43] [Online]. Available: https://www.nvidia.com/en-me/geforce/graphics-cards/rtx-2060/.

[44] [Online]. Available: https://www.pchubonline.com.

[45] Maynilad, *Maynilad connects over 5m customers since re-privatization*. [Online]. Available: https://www.mayniladwater.com.ph/maynilad-connects-over-5m-customers-since-re-privatization/.

[46] Google, *Pricing nbsp;—nbsp; compute engine: Virtual machines (vms) nbsp;—nbsp; google cloud*. [Online]. Available: https://cloud.google.com/compute/all-pricing#general_purpose.

[47] *Firebase pricing*. [Online]. Available: https://firebase.google.com/pricing.