algorithms

MDPI

Article

A Real-Time Car Towing Management System Using ML-Powered Automatic Number Plate Recognition

Ahmed Abdelmoamen Ahmed * and Sheikh Ahmed

Department of Computer Science, Prairie View A&M University, Prairie View, TX 77446, USA; gagunsoye@student.pvamu.edu or sahmed13@student.pvamu.edu
* Correspondence: amahmed@pvamu.edu

Abstract: Automatic Number Plate Recognition (ANPR) has been widely used in different domains, such as car park management, traffic management, tolling, and intelligent transport systems. Despite this technology's importance, the existing ANPR approaches suffer from the accurate identification of number plats due to its different size, orientation, and shapes across different regions worldwide. In this paper, we are studying these challenges by implementing a case study for smart car towing management using Machine Learning (ML) models. The developed mobile-based system uses different approaches and techniques to enhance the accuracy of recognizing number plates in real-time. First, we developed an algorithm to accurately detect the number plate's location on the car body. Then, the bounding box of the plat is extracted and converted into a grayscale image. Second, we applied a series of filters to detect the alphanumeric characters' contours within the grayscale image. Third, the detected the alphanumeric characters' contours are fed into a K-Nearest Neighbors (KNN) model to detect the actual number plat. Our model achieves an overall classification accuracy of 95% in recognizing number plates across different regions worldwide. The user interface is developed as an Android mobile app, allowing law-enforcement personnel to capture a photo of the towed car, which is then recorded in the car towing management system automatically in real-time. The app also allows owners to search for their cars, check the case status, and pay fines. Finally, we evaluated our system using various performance metrics such as classification accuracy, processing time, etc. We found that our model outperforms some state-of-the-art ANPR approaches in terms of the overall processing time.

Keywords: real-time; image processing; ANPR; machine learning; mobile computing

Citation: Ahmed, A.A.; Ahmed, S. A Real-Time Car Towing Management System Using ML-Powered Automatic Number Plate Recognition. *Algorithms* **2021**, *14*, 317. https://doi.org/10.3390/a14110317

Academic Editor: Frank Werner

Received: 22 September 2021
Accepted: 29 October 2021
Published: 30 October 2021

1. Introduction

Automatic Number Plate Recognition (ANPR) is a computer vision technology that automatically recognizes vehicle number plates using high-speed cameras [1]. It involves detecting the position of the plate in the vehicle, identifying the characters and digits in the plate, and converting the captured image to text [2]. ANPR has been used in a wide range of applications such as vehicle access control [3], site security [4], car park management [5], traffic management [6], tolling [7], counter-terrorism [8], etc. In many of these applications, ANPR needs to operate in real-time, which increases the complexity of the plate recognition process.

The state-of-the-art approaches for ANPR utilize Machine Learning (ML) models (e.g., Convolutional Neural Networks (CNN) [9], Recurrent Neural Networks (RNN) [10], Bi-Directional Long Short Term Memory (BLSTM) [11] and Support Vector Machine (SVM) [12]) to recognize the text on license plates. However, these ML models require large and robust datasets of plate photos of different sizes, orientations, and shapes across different regions worldwide to train them. Moreover, these datasets may contain sensitive identifying information related to the vehicle, driver, and location, making it a challenging task to acquire [3]. Therefore, there is a need for alternative ANPR approaches that can

accurately identify the plate's area and recognize its alphanumeric characters in real-time without the need for substantial ANPR datasets that may contain sensitive information related to car owners.

In this paper, we present a mobile-based ANPR system that uses different approaches and techniques to enhance the accuracy of recognizing number plates (The source code and dataset are available online: https://github.com/ahmed-pvamu/Car-Towing-Management, accessed on 10 September 2021). The developed ANPR system has two main modules for detecting the number plate's location on the car body and identifying the plate's alphanumeric characters, respectively. We developed an algorithm for sketching a bounding box around the input image's plate in the first module. The plate box is then fed to the second module, which generates the contours for all alphanumeric characters within the plate box by applying a series of image filters and transformations such as Gaussian filter [13], morphological transformation [14], etc. We used the K-Nearest Neighbors (KNN) model [15] to recognize the actual characters and digits in the number plate.

We implemented a case study for smart car towing management based on our ANPR system. We developed an Android mobile app to allow law-enforcement personnel to capture a photo of the towed car. The captured image is then fed to our ANPR system to recognize the actual characters on the number plate, which is automatically recorded into the car towing management system. Car owners can also search for their towed vehicles, check the case status, submit complaints, and pay fines.

We used more than 160 car images to calculate the classification accuracy of our ANPR system. Figure 1 shows some examples of the various types of number plates with different sizes, orientations, and shapes across various regions worldwide. Our model achieves a classification accuracy of 95% in recognizing number plates in our testing dataset. We also compared the performance of our ANPR system to some existing ML models (i.e., SVM [12]) and computer vision models (i.e., YOLO detector [16]). We found that our ANPR system outperforms SVM and YOLO in terms of processing time, which is critical for the ANPR applications which need to operate in real-time.

(**a**) Example 1 (**b**) Example 2 (**c**) Example 3 (**d**) Example 4

Figure 1. Sample Examples from our Dataset Showing Different Types of Plates with Different Orientations and Shapes Across Various Regions Worldwide.

The contributions of this paper are fourfold. First, we propose a mobile-based AI-powered ANPR system that can help law-enforcement personnel to automate the process of car towing in real-time. We proposed two novel plate detection and character recognition algorithms. Second, we developed a user-friendly interface on top of the proposed ANPR model to allow users to interact with the car towing management system conveniently at both the mobile and cloud sides. Third, we carried out several sets of experiments to evaluate the performance and classification accuracy of our system. We found that our ANPR model outperforms some state-of-the-art ANPR approaches (e.g., SVM and YOLO) in terms of the overall processing time. We tested our system with an image dataset of various shaped and sized number plates, where crowded backgrounds, low contrast, and diverse illumination condition images are taken into consideration. Fourth, the system is designed to be generic, making it applicable to different fields requiring real-time processing and using cameras such as in the transportation field, security, intelligent surveillance, military, etc.

The rest of the paper is organized as follows: Section 2 presents related work. Sections 3 and 4 present the design and prototype implementation of the ANPR system, respectively. Section 5 experimentally evaluates our ANPR system in terms of processing time and classification accuracy. Finally, Section 6 summarizes the results of this work.

2. Related Work

There have been several compelling projects–both in academia and industry–involving vehicle number plate recognition in different domains [1,2]. This section focuses on existing work on the ANPR technology using machine learning [5,6,9–12] and computer vision approaches [3,4,7,8,17].

Shivakumara et al. [11] proposed an ANPR method based on ML to address the challenge of the lack of uniform format of capturing and displaying number plates across different countries. The authors used a combination of three ML models, namely CNN, RNN, and BLSTM, to recognize number plates with diverse background and foreground color. The proposed method has achieved a slightly better classification rate compared to some existing ANPR approaches. Another application of deep learning in license plate recognition was proposed in [10]. The authors tried to address three technical challenges: (1) develop an algorithm to improve the quality of input images by handling the license plate skew, image noise, and license plate blur; (2) use different deep learning models (e.g., RCNN [18] and CNN [9]) in the license plate detection process to enhance the classification rate; and (3) compare the developed method to some public license plate datasets which contain different image resolutions and environmental complex conditions.

Laroca et al. [16] implemented an ANPR system based on the YOLO detector for identifying the location of the license plate. The authors used a set of postprocessing rules for improving the recognition results, such as employing a different confidence threshold for different geographical regions. After locating the plate on the car's body, all plate characters are recognized simultaneously, avoiding the character segmentation process. The system is trained using plate images from several datasets, with the addition of various data augmentation techniques. The proposed method achieved an average classification accuracy of 96.8% over the used datasets. Similar to the work presented in [16], Henry et al. [17] proposed an ANPR method for international license plate recognition based on the YOLO detector. The proposed layout detection algorithm was able to extract the bounding box of the characters in the plate in the correct sequence. The proposed system was evaluated using datasets from five countries, including South Korea, Taiwan, Greece, USA, and Croatia. The experimental evaluation showed that this system could recognize the actual number plate within 42 ms per image.

An online ANPR system for recognizing dirty vehicle plate numbers for high-speed applications is proposed in [7]. The authors used the Connected Component Analysis (CCA) algorithm for character recognition in the plate. A dataset consists of 10,000 images captured by traffic cameras is collected for Persian license plates for evaluating the system. The system achieved classification accuracies of 98.7%, 99.2%, and 97.6% for plate detection, character segmentation, and plate recognition, respectively. Another technique is proposed in [8] to provide an affordable ANPR technology that can be adopted at a large scale in developing countries. The authors implemented a low-cost crowd-sourced system that can select the appropriate cameras for capturing the plate photos and improve the system's performance under various optoelectronic and environmental conditions. The developed optoelectronic model showed usefulness in selecting the surveillance cameras in streets, capturing the highest possible quality of the plates' photos.

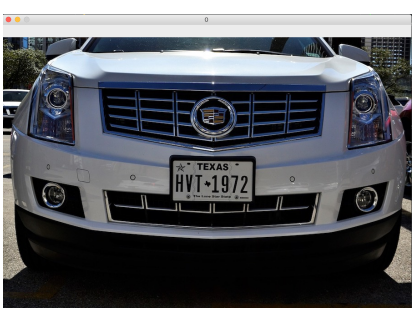
Arafat et al. [4] proposed an ANPR framework for detecting, segmenting, and recognizing various-shaped license plates. The proposed framework has a preprocessing module that converts the input image into a grey-scale version and applies arithmetic-based dilation. Then, the preprocessed image's vertical and horizontal edge densities are extracted and smoothed using a Gaussian filter. The framework was able to detect various shapes

of license plates from both high definition and lower resolution images under different illumination conditions.

In summary, the review of license plate recognition using either machine learning [9–11] or computer vision [4,7,16,17] shows that most of these approaches focus on particular countries, regions, or format of the license plates. Furthermore, most of the ML-based methods require large datasets of plate photos to train them, which may not be available for all regions worldwide due to privacy reasons.

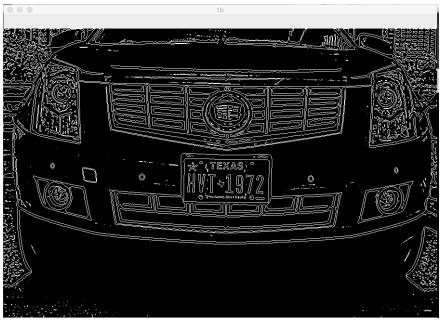
3. Methods

This section presents the design of our ANPR system, which is divided into two main modules: plate detection, and character segmentation and recognition. Figure 2 illustrates the phases of the plate detection, and character segmentation and recognition modules. In the rest of this section, we discuss these modules separately.



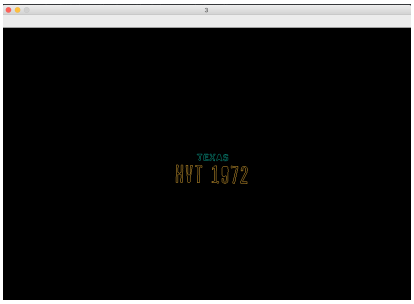
(**a**) Input image

(**b**) Image after grayscaling

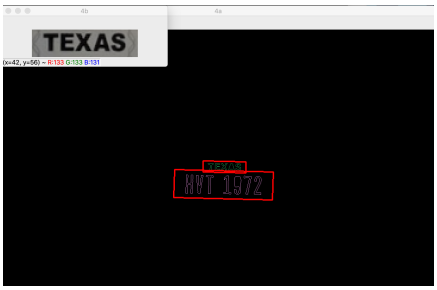


(**c**) Image after thresholding

(**d**) Image after morphological transformation



(**e**) Image after postprocessing



(**f**) Image after segmentation

Figure 2. *Cont*.

(**g**) Character segmentation

(**h**) Character recognition

Figure 2. Phases of the plate detection, character segmentation, and recognition modules.

3.1. Plate Detection

Algorithm 1 describes the process of plate localization and detection. Image preprocessing is the first in our plate detection module, which is required to reduce the complexity of processing color images by converting the colored input image into a grayscale version by applying image desaturation. The desaturation process makes pixel colors more muted by adding more black and white colors. The `cv2.cvtColor` function uses the following formula to convert the color RGB value (24-bit) of every pixel (i, j) to a gray-scale level (8-bit) value:

$$S(i,j) = \frac{1}{10}(3 * R(i,j) + 6 * G(i,j) + B(i,j)) \quad (1)$$

where the spectrum of R, G and B is $R(i,j)$, $G(i,j)$ and $B(i,j)$, respectively; and $S(i,j)$ is the converted grey-scale image.

The `cv2.adaptiveThreshold` function is used to convert the grayscale image into a binary image (see Figure 2b). To achieve the best accuracy of determining every pixel as an object or background, we used the adaptive threshold method that works better in the case of having different lighting conditions in the input image. The adaptive threshold algorithm calculates a different threshold value for every small region of the image, which gives better results for images with varying illumination. The `cv2.ADAPTIVE_THRESH_GAUSSIAN_C` is used as a threshold parameter, which is the weighted sum of neighborhood pixel values where weights are represented by a Gaussian window [13]. The effect of applying the adaptive thresholding function is demonstrated in Figure 2c.

Algorithm 1 Plate Detection Algorithm.

```
procedure FINDING PLATE LOCATION
Input: Input Vehicle Image (carImage)
Output: Extracted Plate Image (plateImage)
/* Image Preprocessing */
imgGrayscale= cv2.cvtColor(carImage, cv2.COLOR_ BGR2HSV.value);
imgThreshold = cv2.adaptiveThreshold(imgGrayscale,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C);
/* Morphological Transformation */
contourList = extractImageContours (imgThreshold);
for each contour c in contourList do
  t_c = morphological_Transformation(n_c);
  redcuedContourList.add(t_c);
end for
/* Image Segmentation */
charList = imageSegmentation(redcuedContourList);
/* Draw a bounding box over the candidate regions */
for each charSequence c in charList do
  region_box = drawBoundingBox(c);
  if isActualPlate (region_box) is true then
    plateImage = extractPlate(region_box);
    return plateImage;
  end if
end for
```

The adaptive threshold is calculated based on the local mean of pixels intensity in Gaussian windows of $m \times n$ pixels, as shown below:

$$O(x,y) = \begin{cases} 255, & \text{if } I(x,y) < \dfrac{\sum_{i=x-\frac{m}{2}}^{x+\frac{m}{2}} \sum_{j=y-\frac{n}{2}}^{y+\frac{n}{2}} I(i,j)}{m*n} \\ 0, & \text{if } I(x,y) \geq \dfrac{\sum_{i=x-\frac{m}{2}}^{x+\frac{m}{2}} \sum_{j=y-\frac{n}{2}}^{y+\frac{n}{2}} I(i,j)}{m*n} \end{cases} \quad (2)$$

where I and O are the input and output images, respectively; and m and n are the window size parameters, which are selected based on the character's size in the region.

To reduce the irrelevant contours from the thresholded image, we used a series of morphological transformations: erosion, opening, closing and gradient operations. Figure 2d shows the effect of the used morphological adaptations on the image. Our model used the Connected Component Analysis (CCA) method [19] for image postprocessing to remove the remaining irrelevant contours and objects in the image. The CCA method groups the connected regions (pixels) that belong to the same object. In particular, the CCA method labels each discovered area of connected pixels by a unique component number. We used the eight-connected component exploration strategy to find the candidate regions in the image. Figure 2e illustrates the image after postprocessing using CCA.

The last step in the plate detection module is the image segmentation process, which separates the relevant objects in the image. Our model identifies candidate regions in the image and draws a bounding box over them. Each candidate region goes through another process to determine whether it is an actual plate or not by counting the sequence of characters in the region. Then, we draw a bounding box around the plate region with the most extended sequence of characters. The output image is shown in Figure 2f.

3.2. Character Segmentation and Recognition

To recognize alphanumeric characters from the segmented image, the candidate number plate is segmented further into individual characters before the optical character recognition process occurs, as shown in Figure 2g. This process is summarized in Algorithm 2. First, we draw a bounding box over each character region using the CCA method. Then, we generate a feature vector from each extracted character image. This feature vector is fed into our KNN classier to recognize the plate's actual characters.

Algorithm 2 Character Segmentation and Recognition Algorithm.

```
1: procedure IDENTIFYING PLATE CHARACTERS
2: Input: Plate Image (plateImage)
3: Output: Plate Characters (plateCharList)
4: /* Character Segmentation */
5: charImageList = charSegmentation(plateImage);
6: /* Character Recognition */
7: for each charImage img in charImageList do
8: /* extract feature vector from every character image */
9:    char_box = drawBoundingBox(img);
10:   vs. = getFeatureVector(char_box);
11: /* Select K nearest characters to the feature vector by comparing their Euclidean distances */
12:   charCandidateList = getNearestChar(v);
13: /* vote between K characters */
14:   plateCharList.add(vote(charCandidateList));
15: end for
16: return plateCharList;
```

The KNN model is trained using 20×20 pixel images for all alphanumeric characters. We validated the KNN model using the cross-validation strategy that assesses how our model's prediction results will generalize to an independent dataset. To recognize an individual character using KNN, we select the `K` nearest characters to the feature vector by comparing their Euclidean distances. The `K` value is set to be 7, which gave us a better classification accuracy. The character recognition module prints the output number plate on the image, as illustrated in Figure 2h.

The `getNearestChar` function is implemented based on the following similarity equation between the two feature vectors (x_1, y_2):

$$S(x_1, y_2) = 1 - d(x_1, y_2) \tag{3}$$

where $d(x_1, y_2) \in [0, 1]$ is the Euclidean distance between x_1 and y_2, and d is calculated as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4}$$

where x_i and y_i are the Euclidean feature vectors, and n is the n-space dimension of x_i and y_i.

4. Experimental Implementation

This section presents the implementation details of our ANPR model, including the mobile-based car towing management system.

Both the plate detection and character recognition modules are implemented using Python programming language. We used various Python OpenCV 4.5 [20] libraries including CV2, Pandas, and Shutils. The user interface of the car towing management is implemented as a self-contained mobile app developed using the Android Development Environment ADT bundle (64 bit). The app uses different technologies and tools including

the Android SDK and XML to build the front-end activities, Python 3.9 as a middleware between the app and the database server on the cloud, and MYSQL 8.0 for the database.

As illustrated in Figure 3, the mobile app allows users to capture a photo of the towed vehicle with proper alignment and orientation. The orientation handler, which runs as a background service thread in the mobile app, is responsible for correcting the tilt and camera angle of capturing the numberplate photo. Once the right image is captured, the app uploads it to a cloud server to detect the plate characters by applying our ANPR model.

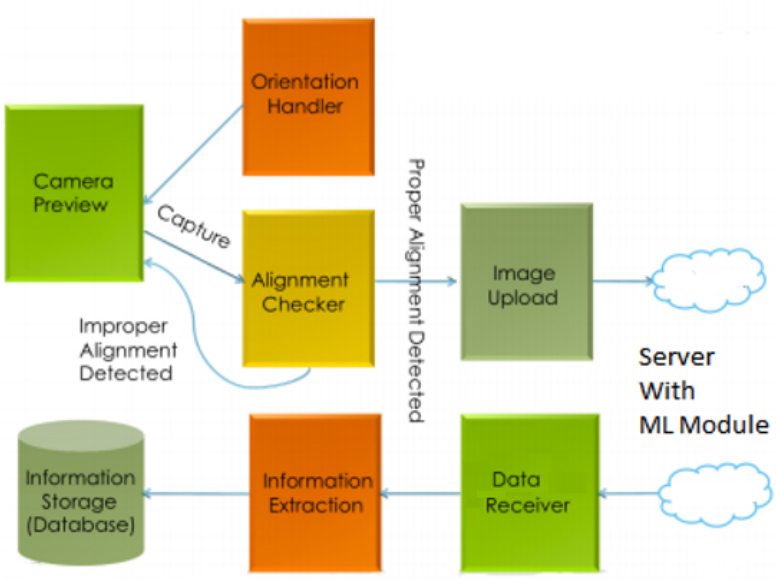


Figure 3. System architecture of the car towing management system.

Figure 4 illustrates the class diagram of mobile-based car towing management system. As shown in the figure, we define three types of system users: administrator, law-enforcement authority, and car owners (users). The system administrator can view/add/edit /delete the law-enforcement personnel and car owners, view pending/paid fines on the towed cars, and handle user complaints. The law enforcement authority users can start the towing process of a car by scanning its number plate using the app, viewing their towing history, and responding to user complaints. The car owner users can view/pay their fines, file complaints, and view their fine history.

Figure 5 shows some selective screenshots of the mobile app from the perspective of the three system users. Figure 5a shows a screenshot of the landing and login screens of the app, which allows all users to log in and register into the system. It also enables car owners to use our system as guest users without the need for registration. The system administrator must register law-enforcement users in the system to get access to their dashboard. User credentials are verified and authenticated on each page to ensure continued verification and authorization of the system.

Figure 5b shows a screenshot of the user registration form on the admin side. We need to gather a minimal amount of information about both the law-enforcement personnel and car owners. Specifically, we asked for the user's full name, email address, phone number, and the desired username and password. Indeed, we could ask for more information at this point, but a long-form is always a turn-off for many people. We also put some form validation code in place, so that we make sure that we have all the data required to create the user account. We check if the name and email, and password are filled in and that both the email address and phone number are in the proper format.

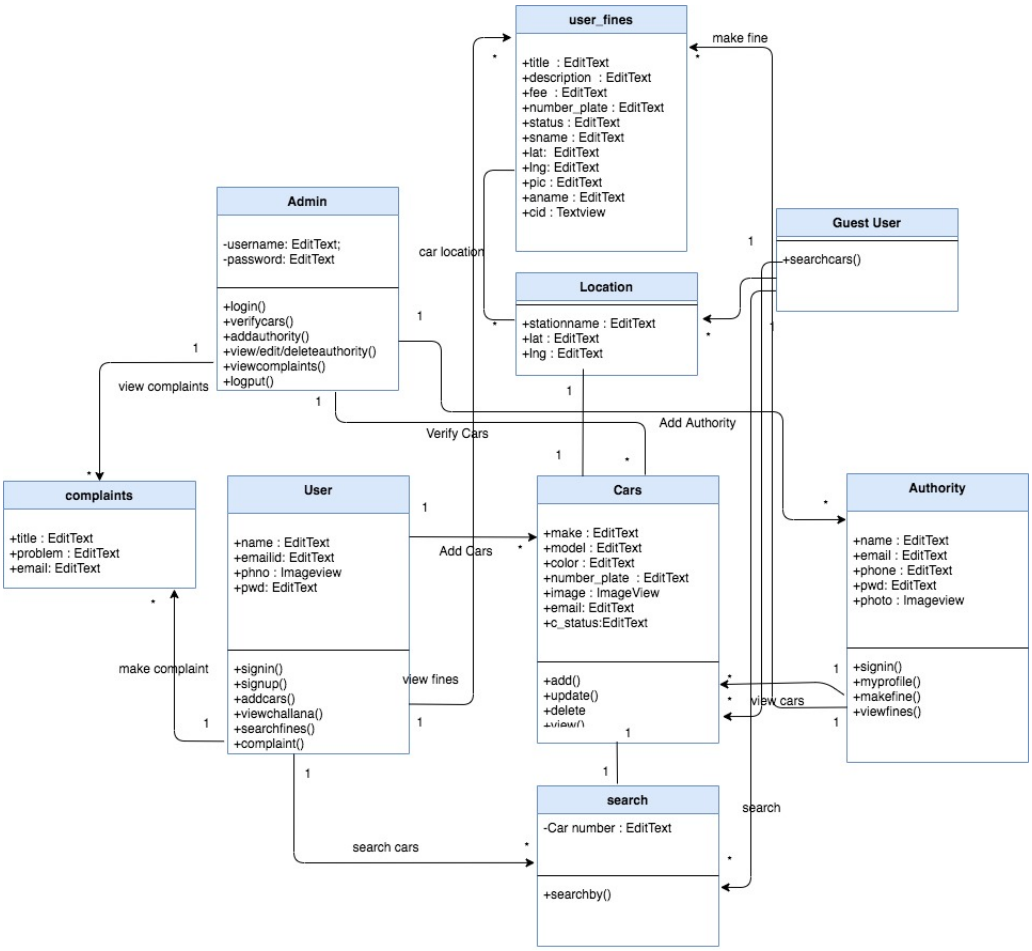
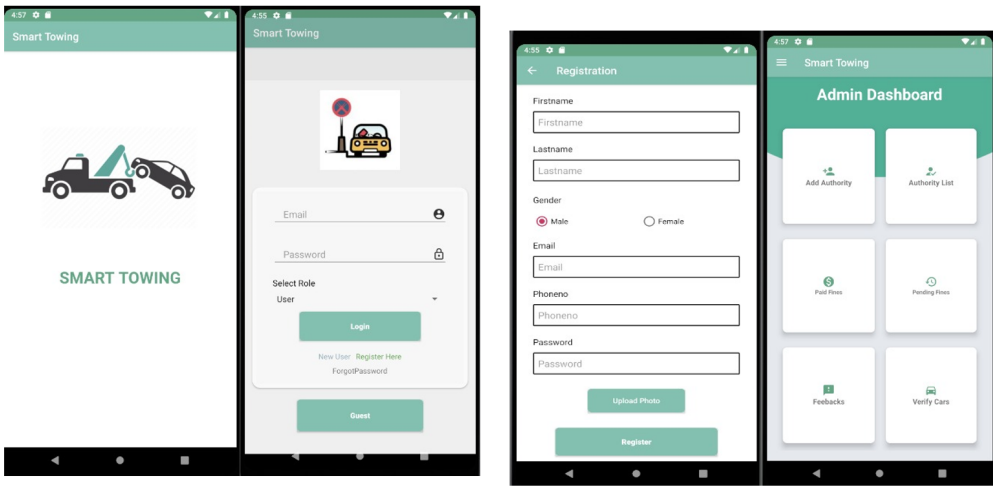


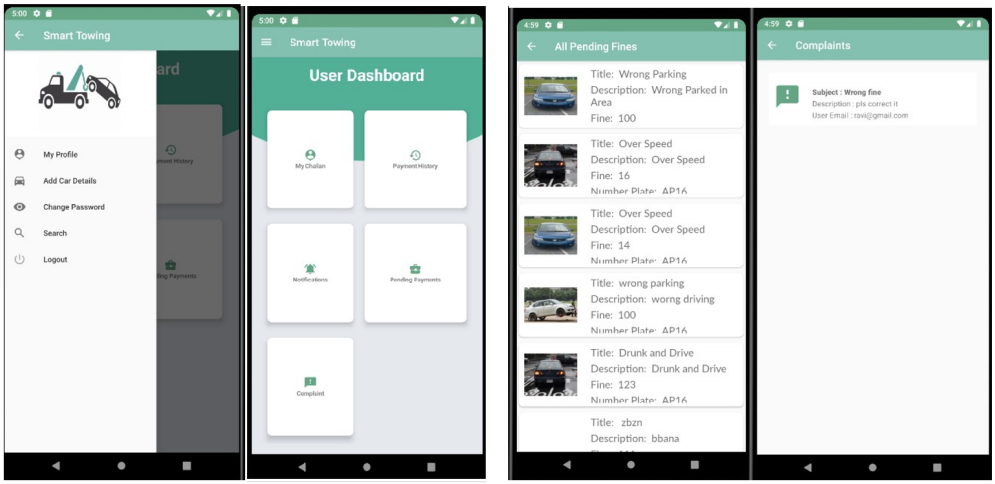
Figure 4. Class diagram of the car towing management system.

Figure 5c shows a screenshot of the car owners (users) menu and dashboard. Mobile users can manage their profile and cars, view the status of their complaints and file a new one, and contact the law-enforcement personnel who issued the ticket for towing their vehicle. Figure 5d shows a screenshot of the user's pending fines and user complaints screens, which show the fine description and amount, as well as the plate number for each case.



(**a**) (**b**)

Figure 5. *Cont.*



(**c**) (**d**)

Figure 5. Screenshots of the mobile app for the car towing management system. (**a**) Landing and login screens. (**b**) User registration and admin dashboard screens. (**c**) User menu and dashboard. (**d**) Pending fines and user complaints.

Figure 6 shows a screenshot of issuing a new towing ticket at the law-enforcement authority side. As shown in the figure, the app gives two options to the law-enforcement officer for entering the number plate: manual and automatic scan. If the officer user selects the latter option, the app uploads the image to the cloud server and displays the results in less than 30 ms, including all processing and communication overheads. All files and data are sent to the cloud in a JSON format, which is then converted into Java objects using the Retrofit API [21]. Retrofit is a REST client for Java and Android, enabling us to retrieve and upload JSON files in Android.

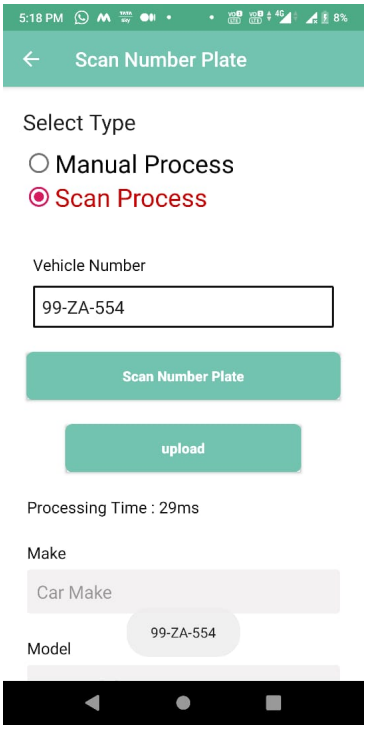


Figure 6. Number plate scanning.

5. Results and Discussion

We experimentally evaluated our ANPR model regarding classification accuracy and performance. We installed instrumentation in the mobile app running on a smartphone to measure the processor time taken to perform various tasks, including plate detection, character segmentation, and character recognition processes. Each experiment presented in this section is carried out for ten trials, then we took the average of these trials' results.

The model training and testing were carried out using a server computer equipped with a 4.50 GHz Intel Core™ i7-16MB CPU processor, 16 GB of RAM, and CUDA GPU capability. The app ran on a Samsung S8 (2.3 GHz Octa-core CPU, 4 GB RAM, Adreno 540 GPU, LTE Category 16) running Android 10.

5.1. Classification Accuracy

Although standard object detection datasets (e.g., Microsoft COCO [22]) exhibit volume and a variety of examples, they are not suitable for plate detection as they annotate a set of object categories not include number plates because of privacy reasons. Therefore, we collected more than 160 images for testing our ANPR model from different sources such as Kaggle [23] and Google Web Scraper [24]. Many images in our dataset are in their natural environments because plate detection is highly dependent on contextual information.

We observed that our model delivers good results even when the car images are captures from different distances from the camera, orientations and illumination conditions. Figure 7 shows some samples of the successful recognition of all plate characters. However, sometimes the model fails to recognize some characters correctly as the model confuses alphanumeric characters with similar structures such as D with O, 6 with 8, K with I, etc. Figure 8 illustrates an example of a character mismatch error.

Figure 9 depicts the average classification rates of our model with respect to different dataset sizes and accuracy granularities. We found that our model achieves classification accuracies of 94.6%, 96.9%, and 99.4% for recognizing the plate characters with 0, 1, and 2 character(s) mismatch error, respectively. We also calculated the overall classification accuracy of the two ANPR modules: plate detection, character segmentation and recognition. We found that our model achieves an overall classification accuracy of 95% in recognizing number plates across different regions worldwide.

The overall classification accuracy is calculated based on the following formula:

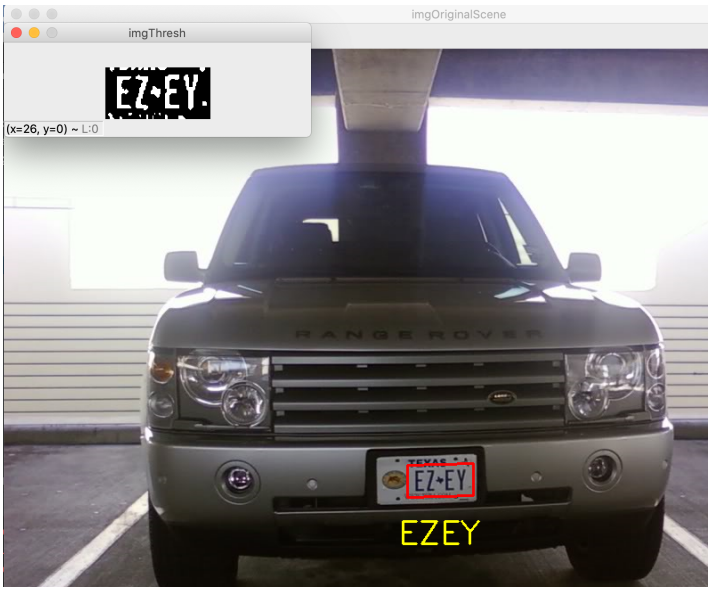
$$Accuracy = (DSR)\% \tag{5}$$

where D is the plate detection rate, S is the character segmentation rate, and R is the character recognition rate. D is calculated as the number of detected plates divided by the total number of plates in our testing dataset. S is calculated as the ratio of the number of correctly segmented characters to the total number of characters in our dataset. R is the proportion of correctly classified characters over our dataset.

5.2. Processing Time

Given that many ANPR applications need to operate in real-time, the processing time for detecting the plate characters is a critical factor in evaluating the different ANPR models. We found that our model, on average, requires around 10ms only to detect the actual characters in the number plate in an image captured by our mobile app without communication overheads. The overall processing time, on average, was measured to be 29 ms, including the app processing and communication overheads. This shows that our model is adequate to be deployed on an IoT surveillance camera to recognize the number plates in real-time.

(**a**) Example 1



(**b**) Example 2

(**c**) Example 3

Figure 7. Examples of successful recognition of all plate characters.

Figure 8. An example of a character mismatch error.

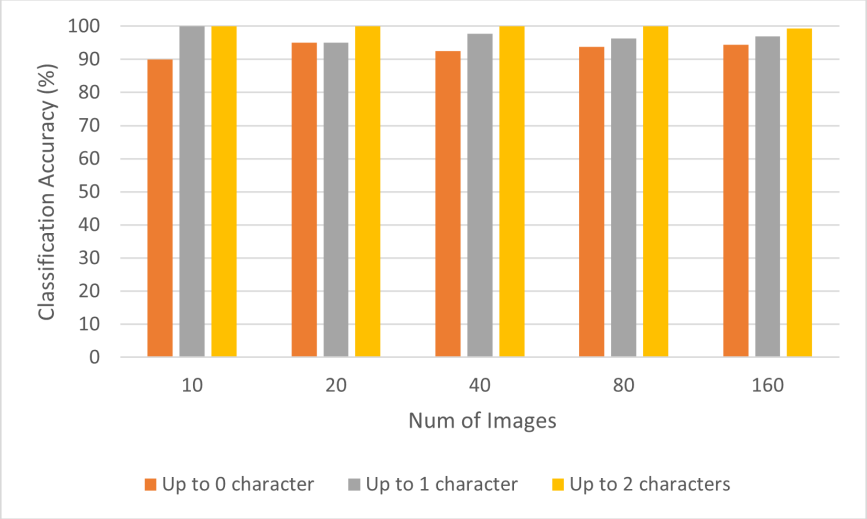


Figure 9. The Average classification accuracy of our model.

To put these numbers in some context, consider a commercial surveillance camera that can run at 30 Frames Per Second (fps), which means it can generate up to one image every 33 ms. Our ANPR model can process more than 3 images every 33 ms, or support a super-resolution camera that runs at the rate of 90 fps.

We compared the performance of our model to two state-of-the-art ANPR approaches, namely SVM [12] and YOLO detector [16]. We found that our model outperforms both methods in terms of the overall processing time to recognize all characters in the license plates over our testing dataset. Figure 10 shows the average CPU processing time of our model compared to SVM and YOLO. We found that our model outperforms both approaches in terms of the overall CPU processing time, even after including the mobile app overheads.

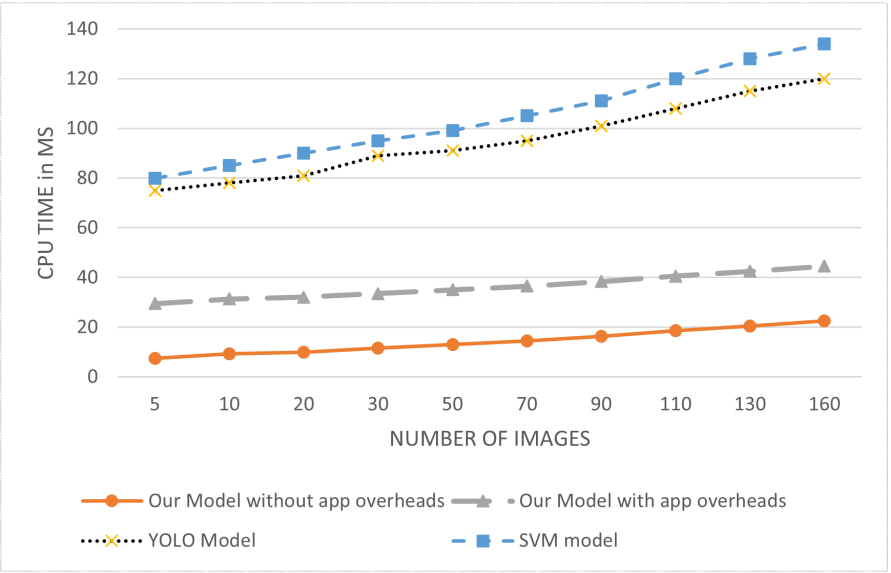


Figure 10. The CPU processing time of our model compared to some existing models such as SVM and YOLO.

6. Conclusions

This paper presented the design and implementation of an ML-powered car towing management system using automatic number plate recognition. We believe that the proposed system would create a better opportunity for law-enforcement personnel to automate the process of car towing. The proposed ANPR model approach comprises three phases: plate detection, character segmentation, and character recognition. We tested our system with an image dataset of various shaped and sized number plates, where crowded backgrounds, low contrast, and diverse illumination condition images are taken into consideration.

We carried out several sets of experiments for evaluating the performance and classification accuracy of our system, paying particular attention to the classification and processing time. Most notably, our model could process a number of images per second more than triple the commercial fps. This proves that our ANPR model is suitable for real-time inference at the edge with high prediction accuracy and response time, which could be used for various applications such as tolling, traffic management, security surveillance, etc. Moreover, we found that our model outperforms some state-of-the-art ANPR approaches in terms of the overall processing time.

We expect that this research would increase the open-source knowledge base in the area of computer vision and real-time video surveillance on the network edge by publishing the source code and dataset to the public domain (The source code and dataset are available online: https://github.com/ahmed-pvamu/Car-Towing-Management, accessed on 10 September 2021).

In on-going work, we are looking into opportunities for generalizing our approach to be used for commercial surveillance cameras. We are developing a motion detection module to be integrated with our model to capture an image when a new scene is detected. It will be useful for supporting the sensing needs of a wide range of researches [25–35] and applications [36–46]. Finally, experiments with more massive datasets are needed to study the robustness of our system at a large scale, and improve the prediction accuracy and minimize the character mismatch errors.

Author Contributions: Conceptualization, A.A.A. and S.A.; methodology, A.A.A.; software, S.A.; validation, A.A.A.; formal analysis, A.A.A.; investigation, A.A.A. and S.A.; resources, A.A.A.; data curation, S.A.; writing—original draft preparation, A.A.A. and S.A.; writing—review and editing, A.A.A.; visualization, S.A.; supervision, A.A.A.; project administration, A.A.A.; funding acquisition, A.A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research work is supported in part by the National Science Foundation (NSF) under grant # 2011330. Any opinions, findings, and conclusions expressed in this paper are those of the authors and do not necessarily reflect NSF's views.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data and source code that support the findings of this study are openly available at: https://github.com/ahmed-pvamu/Car-Towing-Management (accessed on 16 October 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Qadri, M.T.; Asif, M. Automatic number plate recognition system for vehicle identification using optical character recognition. In Proceedings of the International Conference on Education Technology and Computer, Singapore, 17–20 April 2009; pp. 335–338.
2. Du, S.; Ibrahim, M.; Shehata, M.; Badawy, W. Automatic license plate recognition (alpr): A state-of-the-art review. *IEEE Trans. Circuits Syst. Video Technol.* **2013**, *23*, 311–325. [CrossRef]
3. Beibut, A.; Magzhan, K.; Chingiz, K. Effective algorithms and methods for automatic number plate recognition. In Proceedings of the IEEE 8th International Conference on Application of Information and Communication Technologies (AICT), Astana, Kazakhstan, 15–17 October 2014; pp. 1–4.
4. Arafat, M.Y.; Khairuddin, A.S.M.; Paramesran, R. Connected component analysis integrated edge based technique for automatic vehicular license plate recognition framework. *IET Intell. Transp. Syst.* **2020**, *14*, 712–723. [CrossRef]
5. Zou, Y.; Zhang, Y.; Yan, J.; Jiang, X.; Huang, T.; Fan, H.; Cui, Z. A robust license plate recognition model based on bi-lstm. *IEEE Access* **2020**, *8*, 30–41. [CrossRef]
6. Mondal, M.; Mondal, P.; Saha, N.; Chattopadhyay, P. Automatic number plate recognition using cnn based self synthesized feature learning. In Proceedings of the IEEE Calcutta Conference (CALCON), Kolkata, India, 2–3 December 2017; pp. 378–381.
7. Panahi, R.; Gholampour, I. Accurate detection and recognition of dirty vehicle plate numbers for high-speed applications. *IEEE Trans. Intell. Transp. Syst.* **2017**, *18*, 767–779. [CrossRef]
8. Rademeyer, M.C.; Barnard, A.; Booysen, M.J. Optoelectronic and environmental factors affecting the accuracy of crowd-sourced vehicle-mounted license plate recognition. *IEEE Open J. Intell. Transp. Syst.* **2020**, *1*, 15–28. [CrossRef]
9. Hendryli, J.; Herwindiati, D.E. Automatic license plate recognition for parking system using convolutional neural networks. In Proceedings of the International Conference on Information Management and Technology (ICIMTech), Bandung, Indonesia, 13–14 August 2020; pp. 71–74.
10. Weihong, W.; Jiaoyang, T. Research on license plate recognition algorithms based on deep learning in complex environment. *IEEE Access* **2020**, *8*, 61–75. [CrossRef]
11. Shivakumara, P.; Tang, D.; Asadzadehkaljahi, M.; Lu, T.; Pal, U.; Anisi, M. Cnn-rnn based method for license plate recognition. *CAAI Trans. Intell. Technol.* **2018**, *3*, 169–175. [CrossRef]
12. Noprianto; Wibirama, S.; Nugroho, H.A. Long distance automatic number plate recognition under perspective distortion using zonal density and support vector machine. In Proceedings of the International Conference on Science and Technology (ICST), Yogyakarta, Indonesia, 11–12 July 2017; pp. 159–164.
13. Chen, B.H.; Tseng, Y.S.; Yin, J.L. Gaussian-adaptive bilateral filter. *IEEE Signal Process. Lett.* **2020**, *27*, 1670–1674. [CrossRef]
14. Huang, W.; Liu, J. Robust seismic image interpolation with mathematical morphological constraint. *IEEE Trans. Image Process.* **2019**, *29*, 819–829. [CrossRef] [PubMed]
15. Zhang, S.; Li, X.; Zong, M.; Zhu, X.; Wang, R. Efficient knn classification with different numbers of nearest neighbors. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 1774–1785. [CrossRef] [PubMed]
16. Laroca, R.; Zanlorensi, L.A.; Gonçalves, G.R.; Todt, E.; Schwartz, W.R.; Menotti, D. An efficient and layout-independent automatic license plate recognition system based on the YOLO detector. *arXiv* **2019**, arXiv:1909.01754.
17. Henry, C.; Ahn, S.Y.; Lee, S. Multinational license plate recognition using generalized character sequence detection. *IEEE Access* **2020**, *8*, 85–199. [CrossRef]
18. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 1137–1149. [CrossRef] [PubMed]
19. Ciarach, P.; Kowalczyk, M.; Przewlocka, D.; Kryjak, T. Real-time fpga implementation of connected component labelling for a 4k video stream. In Proceedings of the International Symposium on Applied Reconfigurable Computing, Darmstadt, Germany, 9–11 April 2019; pp. 165–180.
20. Opencv: A Python Library for Real-Time Computer Vision. Available online: https://pypi.org/project/opencv-python/ (accessed on 16 August 2021).
21. Retrofit: A Type-Safe Http Client for Android and Java. Available online: https://square.github.io/retrofit/ (accessed on 16 August 2021).

22. Lin, T.; Maire, M.; Belongie, S.J.; Bourdev, L.D.; Girshick, R.B.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common objects in context. *Comput. Vis.* **2014**, *1405*, 0312.
23. Kaggle: Machine Learning and Data Science Community. Available online: https://www.kaggle.com/ (accessed on 16 August 2021).
24. Google Web Scraper. Available online: https://chrome.google.com/webstore/detail/web-scraper/jnhgnonknehpejjnehehllkliplmbmhn?hl=en (accessed on 16 August 2021).
25. Mo'men, A.M.A.; Hamza, H.S.; Saroit, I.A. New attacks and efficient countermeasures for multicast aodv. In Proceedings of the 7th International Symposium on High-capacity Optical Networks and Enabling Technologies, Cairo, Egypt, 19–21 December 2010; pp. 51–57.
26. Moamen, A.A.; Nadeem, J. ModeSens: An approach for multi-modal mobile sensing. In Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity, ser. SPLASH Companion 2015, Pittsburgh, PA, USA, 25–30 October 2015; pp. 40–41.
27. Abdelmoamen, A. A modular approach to programming multi-modal sensing applications. In Proceedings of the IEEE International Conference on Cognitive Computing, ser. ICCC '18, San Francisco, CA, USA, 2–7 July 2018; pp. 91–98.
28. Moamen, A.A.; Jamali, N. Coordinating crowd-sourced services. In Proceedings of the 2014 IEEE International Conference on Mobile Services, Anchorage, AK, USA, 27 June 2014; pp. 92–99.
29. Moamen, A.A.; Jamali, N. An actor-based approach to coordinating crowd-sourced services. *Int. J. Serv. Comput.* **2014**, *2*, 43–55. [CrossRef]
30. Moamen, A.A.; Jamali, N. CSSWare: A middleware for scalable mobile crowd-sourced services. In *Proceedings of MobiCASE*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 181–199.
31. Moamen, A.A.; Jamali, N. Supporting resource bounded multitenancy in akka. In Proceedings of the ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH Companion 2016), Amsterdam, The Netherlands, 30 October–4 November 2016; ACM: Pittsburgh, PA, USA, 2016; pp. 33–34.
32. Moamen, A.A.; Wang, D.; Jamali, N. Supporting resource control for actor systems in akka. In Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2017), Atlanta, GA, USA, 5–8 June 2017; pp. 1–4.
33. Abdelmoamen, A.; Wang, D.; Jamali, N. Approaching actor-level resource control for akka. In Proceedings of the IEEE Workshop on Job Scheduling Strategies for Parallel Processing, ser. JSSPP '18, Vancouver, BC, Canada, 25 May 2018; pp. 1–15.
34. Moamen, A.A.; Jamali, N. ShareSens: An approach to optimizing energy consumption of continuous mobile sensing workloads. In Proceedings of the 2015 IEEE International Conference on Mobile Services (MS '15), New York, NY, USA, 27 June–2 July 2015; pp. 89–96.
35. Moamen, A.A.; Jamali, N. Opportunistic sharing of continuous mobile sensing data for energy and power conservation. *IEEE Trans. Serv. Comput.* **2020**, *13*, 503–514. [CrossRef]
36. Moamen, A.A.; Jamali, N. CSSWare: An actor-based middleware for mobile crowd-sourced services. In Proceedings of the 2015 EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous '15), Coimbra, Portugal, 22–24 July 2015; pp. 287–288.
37. Ahmed, A.A.; Olumide, A.; Akinwa, A.; Chouikha, M. Constructing 3d maps for dynamic environments using autonomous uavs. In Proceedings of the 2019 EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous '19), Houston, TX, USA, 8–11 November 2019; pp. 504–513.
38. Moamen, A.A.; Jamali, N. An actor-based middleware for crowd-sourced services. *EAI Endorsed Trans. Mob. Commun. Appl.* **2017**, *3*, 1–15.
39. Abdelmoamen, A.; Jamali, N. A model for representing mobile distributed sensing-based services. In Proceedings of the IEEE International Conference on Services Computing, ser. SCC '18, San Francisco, CA, USA, 2–7 July 2018; pp. 282–286.
40. Ahmed, A.A. A model and middleware for composable iot services. In Proceedings of the International Conference on Internet Computing & IoT, ser. ICOMP '19, Las Vegas, NV, USA, 12–15 July 2019; pp. 108–114.
41. Ahmed, A.A.; Eze, T. An actor-based runtime environment for heterogeneous distributed computing. In Proceedings of the International Conference on Parallel & Distributed Processing, ser. PDPTA '19, Las Vegas, NV, USA, 27–30 July 2019; pp. 37–43.
42. Ahmed, A.A.; Omari, S.A.; Awal, R.; Fares, A.; Chouikha, M. A distributed system for supporting smart irrigation using iot technology. *Eng. Rep.* **2020**, *3*, 1–13.
43. Ahmed, A.A. A privacy-preserving mobile location-based advertising system for small businesses. *Eng. Rep.* **2021**, e12416. [CrossRef]
44. Ahmed, A.A.; Echi, M. Hawk-eye: An ai-powered threat detector for intelligent surveillance cameras. *IEEE Access* **2021**, *9*, 63283–63293. [CrossRef]
45. Ahmed, A.A.; Reddy, G.H. A mobile-based system for detecting plant leaf diseases using deep learning. *AgriEngineering* **2021**, *3*, 478–493. [CrossRef]
46. Ahmed, A.A.; Agunsoye, G. A real-time network traffic classifier for online applications using machine learning. *Algorithms* **2021**, *14*, 1–20. [CrossRef]