PAY AS YOU PARK SMART PARKING SOLUTION : DEEP LEARNING-BASED PARKING SURFACE CLASSIFICATION AND STANDARDIZATION

M.D. Sachintha Madushan Antany

(IT18012552)

B.Sc. (Hons) Information Technology – Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

September 2021

PAY AS YOU PARK SMART PARKING SOLUTION : DEEP LEARNING-BASED PARKING SURFACE CLASSIFICATION AND STANDARDIZATION

M.D. Sachintha Madushan Antany

(IT18012552)

Dissertation submitted in partial fulfillment of the requirement for the Bachelor of Science in Information Technology specialization in Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

September 2021

DECLARATION

I declare that this is our own work and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
M.D.S.M Antany	IT18012552	[illegible]

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor Date

....

DEDICATION

This research is dedicated to all the people who daily travel in the road who facing many difficulties in finding a better parking yard to park and the people who willing to rent their own unused places as parking places. In urban cities parking is an essential task. People face to this problem in their day-to-day life. My solution is standardizing the parking yard before registering to the system for public use by giving the better experience to both parking owner and the drivers. Considering the problems faced by drivers my effort is to introduce a smart optimal parking for a better parking.

ACKNOWLEDGEMENT

I would like to present my heartfelt thanks for Ms. Nadeesha Pemadasa (Assistant Lecturer – Faculty of Computing SLIIT) and Ms. Thamali Dissanayake (Lecturer – Faculty of Computing SLIIT) for the support, guidance and motivation throughout the research as our supervisor for make it successful and effective research.

In addition to that I would like thank Dr. Dharshana Kasthurirathna for the special guidance given in the beginning of the research. He guided us with the research components in the earlier stages.

ABSTRACT

One of the most important procedures before deciding on a parking spot is to assess the suitability of the land. As a result, people will have to waste more time and money in order to benefit from their expertise. When contemplating the real-world scenario of a vehicle parking solution, the system administrator's busy daily routine would make registering and validating the appropriateness of new land for a car parking application difficult. In addition, there are millions of acres and garages owned by people in various places while looking at the globe map. As a result, once registration begins following the deployment of a Parking application, and if this procedure is managed by a human, there will be a massive queue for private land and garage registration and verification processes. As a result, in order to address these issues, this study will present a machine learning and image processing-based automated solution. The parking system may evaluate the parking owner's image data of the parking land/garage that was taken during the registration process and provide the compatibility level of the ground as well as the necessary advice to resubmit images after the parking lot owner makes adjustments using this approach.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

CNN	Convolutional Neural Network
SVM	Support Vector Machine
API	Application Programming Interface
UI	User Interface
RAM	Random-access memory
MERN	MongoDB, Express, React, Node
DOM	Document Object Model
GPS	Global Positioning System

1 INTRODUCTION

1.1 Background & Literature survey

1.1.1 Background

There are many Smart Parking solutions available in the worldwide which was created with different modern technologies [1].But every solution does not cover all the problems users have. Usually, a normal parking solution provides a service to the users to track a parking location and park as they want without finding a place searching here and there in the town. But when it comes to SMART parking, users must have SMART features to overcome the current problems in the world. The “Pay as you park” Smart parking concept is such a solution to make life easier for users all over the country.

When people arrive on the road in their own vehicle, they get rude and irritated by the current situation. All drivers are battling with one another for the right spot to park their vehicle. When there is a cinema or a big event in that location, people have a lot of difficulties inside the main parking lots and cause a lot of traffic on the road. There has been no parking solution that has presented a strategy to address the shortage of space issues in urban areas. In urban areas, road traffic is a common thing. There are numerous reasons for such heavy traffic on the road. However, in the majority of cases, traffic jams are caused by vehicles who park inappropriately off the main roads [2]. This is largely due to drivers' inability to find suitable parking spaces and the lack of parking lots near their final destination.
So, in this Smart parking solution, we have considered that matter and defined a feature to add private lands and garages to the internal map of parking lots by the owner of private properties. But in this case, we were found that there are many such lands and garages available in urban cities without taking advantage of their owners. As shown in Figure 1 which depicts the data retrieved by the survey, there are a lot of people who have their own lands and garages. Most of them are not using those properties (Figure 2) and monthly earning is less than 5000 lkr (Figure 3). Most of the

people who have empty lands and garages would like to earn extra money from those properties (Figure 4). According to the survey also it was confirmed that these kinds of scenario can widely adaptable all over the country.

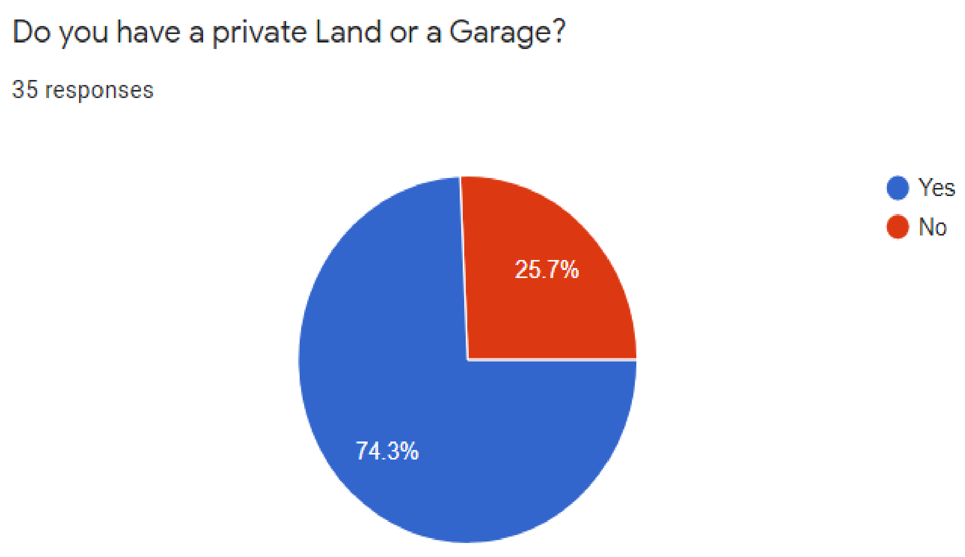


FIGURE 1 PEOPLE WHO HAVE PRIVATE LANDS AND GARAGES.

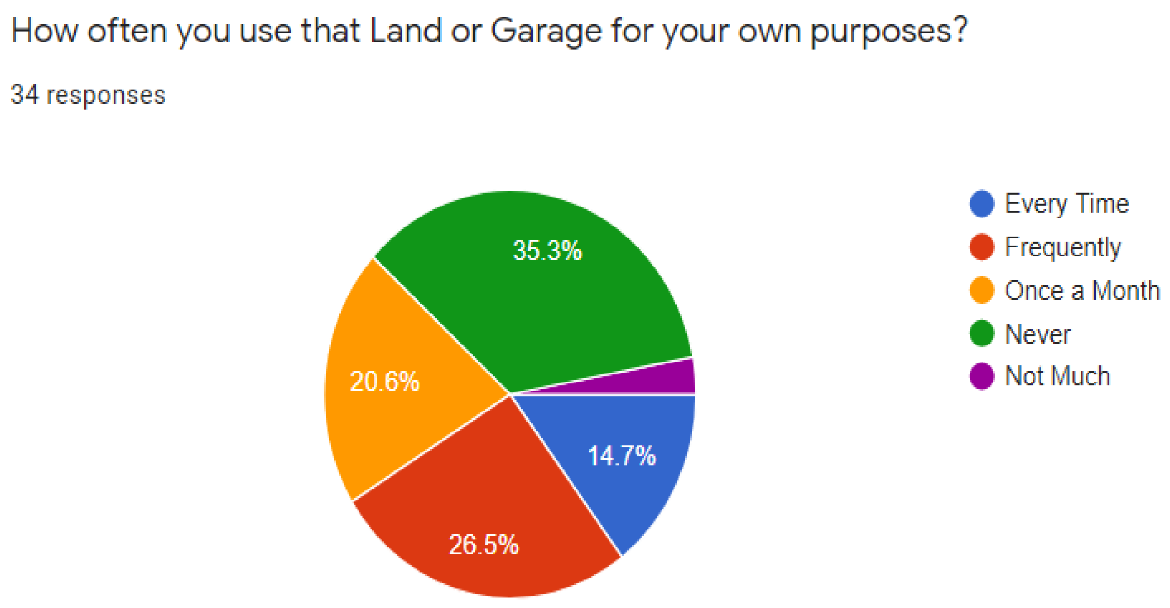


FIGURE 2 USAGE OF THOSE LANDS AND GARAGES.

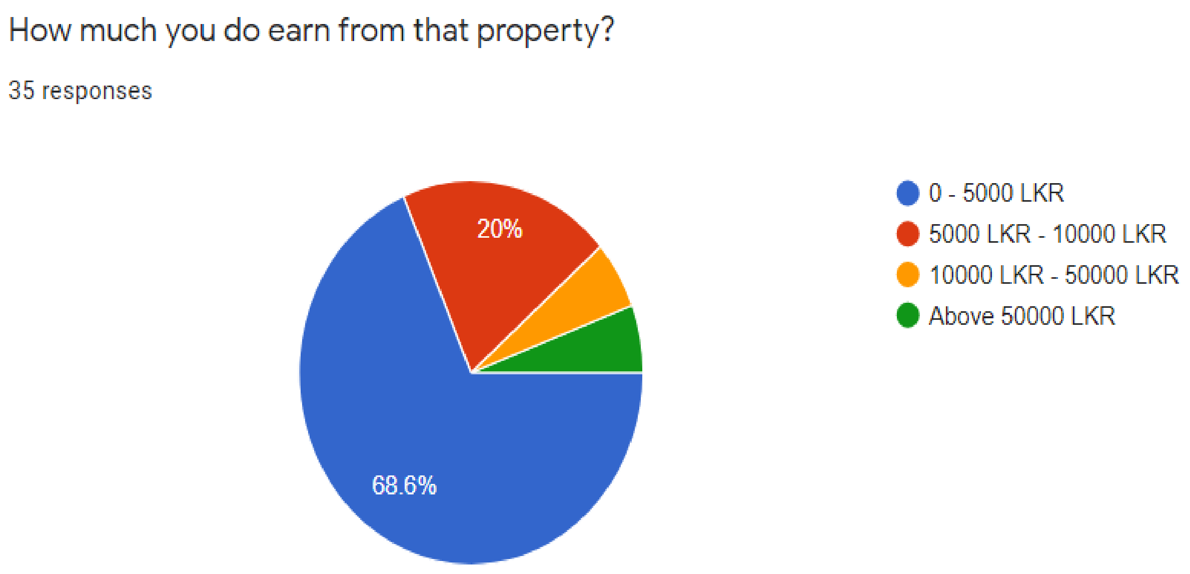


FIGURE 3 EARNINGS RANGE FROM LANDS AND GARAGES.

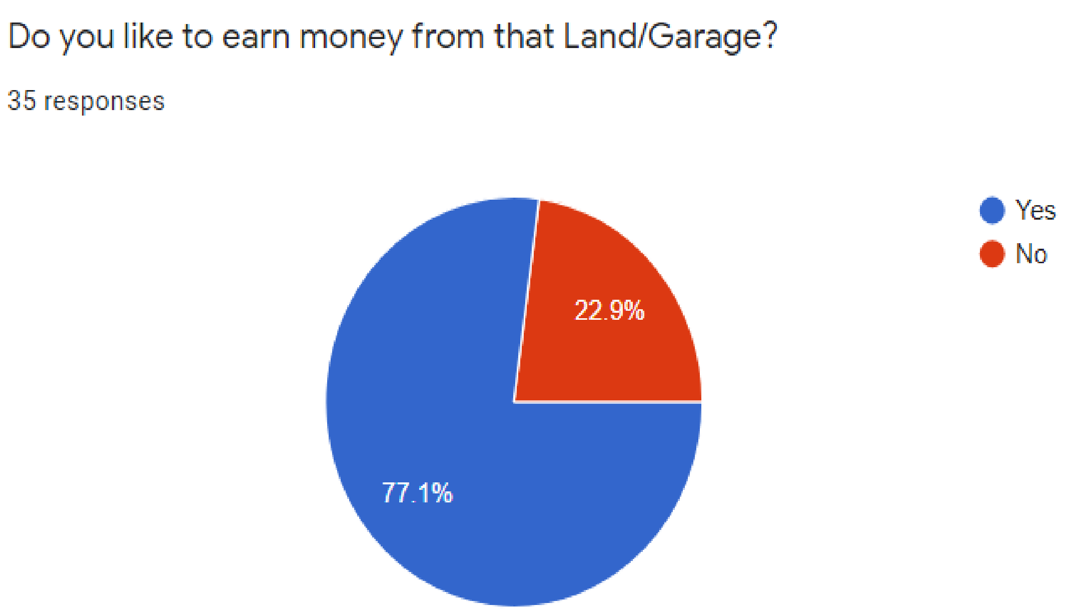


FIGURE 4 EARNING PREFERENCE FROM LANDS AND GARAGES.

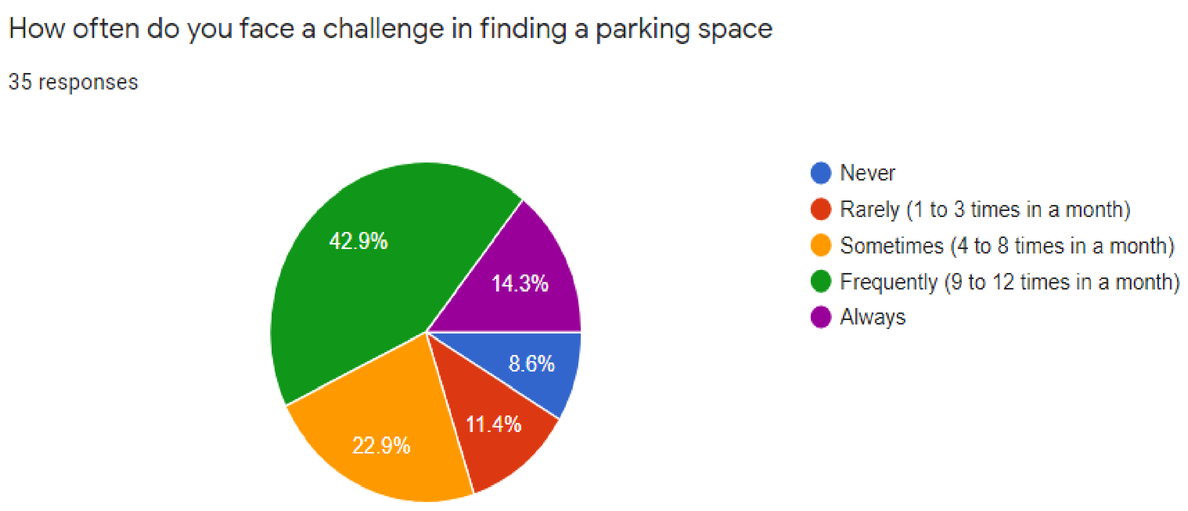


FIGURE 5 FREQUENCY OF STRUGGLE TO FIND A PARKING PLACE.

So, when come to registering these lands in the application, there should be a special procedure to conduct for the verification of the suitability of the land in each one of them. This is a time-consuming process because it needs to verify each place by a special agent whether that place is suitable for vehicle parking. This process should continue at least term basis because parking standardization is an important thing to maintain the client’s trust on us and development of the business all over the country. But when comparing the survey (Figure 4) 77.1% of the crowd would like to start a business at home using their properties. So, this is not an easy task to assign agents for the verification to these lands and garages. Therefore, automating the verification method using the technology-based solution is needed by the interaction of the parking lot owner and system.

1.1.2 Literature Survey

There are numerous parking applications available in various regions throughout the Sri Lanka, but no one has provided a space for registering new locations (such as private car parks, lands, garages, and so on) as parking slots on the map. Car parking is a severe issue in both developed and developing countries' urban centers. As a result of the rapid expansion in car ownership, many cities are experiencing a shortage of vehicle parking spaces, resulting in an imbalance between parking supply and demand, which may be viewed as the fundamental cause of metropolitan parking troubles [3]. It is impossible to manage private properties to use as public parking spaces while using the virtual parking solution. As a result, the smart parking solution "Pay as you park" introduces a new function to address the lack of parking facilities in urban areas by standardized private spaces for public use. So, before registering to the "Pay as you park" system, the "pay as you park" team examined the best technology to detect the standard of private parking spaces.

Detecting potholes and determining the surface type and quality of the parking space are the main tasks of this system while registering lands in this scenario. Otherwise, vehicles may find a variety of issues while parking in those new areas, leading them to lose faith in the solution's new car parks. To find potholes, objects, and surface quality, many types of research have used sensor-based systems [4] and machine learning methodologies [5][6]. For various challenges such as soil classification, pothole detection, surface crack detection, and so on, many authors have used a variety of machine learning and image processing algorithms.

A group of researchers presents a literature evaluation of current papers for soil type classification as part of their related work. They built a model of an image processing-based soil classifier to determine the soil type of the ground using this method. They used seven types of soil for classification in that model, and the preprocessed images were used to extract the features, which were then used to train the Support Vector Machine (SVM) classifier [6].

Another study produced a CNN-based road surface crack detection model that responds to brightness changes. In that instance, they develop a semantic segmentation model with an autoencoder structure for detecting road surfaces, as well as a CNN-based image preprocessing model for detecting road surface cracks under various lighting situations [7].

A group of researchers conducted another experiment to determine the most accurate model for monitoring the road surface and detecting potholes. They investigated and utilized various deep learning models, including convolutional neural networks (CNN), four long short-term memory network (LSTM) networks, and reservoir computing (RC) models, in that study. They presented the confusion matrices and accuracy for each tested model at the conclusion of the study. They also demonstrated that, unlike CNN, other models failed to classify the other classes, and that the majority of potholes were accurately classified by themselves [8].

1.2. Research Gap

Detecting potholes and determining the surface type and quality of the parking lot are the main tasks of this system while registering lands and entrances in this scenario. Otherwise, drivers may encounter a variety of challenges when parking in those new locations, causing them to lose faith in us. Many forms of study have been applied sensor-based systems to locate land potholes [9], but these are not appropriate for our circumstance because the entire process should be carried out through the internet. As a result, the best option to deal with this challenge is to use image processing and machine learning-based technologies for verification [10][11].

Despite the fact that there have been numerous studies on road pothole recognition and surface type detection, there have been none that have merged the two features. Furthermore, we must identify the parking land surface type and predict conditions in different seasons of the country while using surface type classification and identification techniques. A verification procedure must be carried out as a result of this.

Research	**Surface classification**	**Pothole detection**	**Gravel Soil, and Clay Soil detection**	**Surface detection**
Research A [6]	✓			
Research B [9]		✓		
Research C [12]			✓	
Research D [11]				✓
“Pay as you park” Smart parking solution	✓	✓	✓	✓

TABLE 1 COMPARISON OF THE FORMER RESEARCHES

1.3. Research Problem

Car parking is a major problem in urban areas in both developed and developing countries. With the rapid incense of car ownership, many cities are suffering from lacking car parking areas with an imbalance between parking supply and demand which can be considered the initial reason for major cities parking problems. As a solution for that this system giving a function to register the new parking areas all over the country. Because of the high demand for such a system, people await to register as fast as possible to the application (Figure 1.4). In this situation as system admin cannot engage with all requests for the verification process. So that avoid such a problem, we have produced an automated verification process using Image processing and machine learning-based technology.

When we going through deep into the problem of the verification process, we have identified that we need to consider the quality of the surface type of the parking land and the road condition of the entrance to the garage or land. We found that users care about the safety and comfortability of a parking lot when choosing a parking area. To ensure those criteria we register the user to our system as a valid parking lot provider. And to ensure the standard of the land and garage we take the distance to the parking lot from the main road, availability of the garage or land, images of the entrance road to the garage or the land and if the place is not concreted basement, we ask for an image of sample land surface images to check whether that place is okay to park vehicles daily without having any trouble in the rainy seasons also. We planned for an automated term-based verification process to ensure the satisfaction of both vehicle owner and private parking lot owners by protecting the vehicles, private lands, and garages.

2 OBJECTIVES

2.1. Main Objectives

The main objective of this project is to implement the “Pay as you park” smart parking system by giving Smart solutions for all identified drawbacks of previous parking solutions and main problems of the drivers faced in the past years. However, this research paper focused on one of a major part of the main four research component in the system which was registration and image-based land/ garage verification of the parking lots.

2.2. Specific Objectives

To accomplish the main objective, there are some specific objectives to cover up. Those sub-objectives are listed down below in brief.

- Collecting the all data needs to create the models

  To create accurate models to retrieve the correct output from the given data to the system, we have the challenge to train the models with enough datasets. So that predefined collection of datasets should be collected gradually with the process of the research project.

  i. Road pothole datasets [13]
  ii. Surface types datasets [14]
  iii. Cement and concrete datasets [15]

- Setup a flow to get the user side information to the system for the registration

To do the proper verification on the registration of lands or garages system needs accurate images in different views. So that making registration flow is an important thing when considering user aspects and system aspects.

i. Samples photos of surface of the land
ii. Sample photos of garage in different views
iii. Photos of the entrance to the garage or land

- Creating an optimal architecture to handle the system

Planning a way to build the system to run in an optimal way is another specific objective of this process. So, choosing the best and cost-effective cloud infrastructure provider for the system and how effective it is to integrate with such a platform to build up this system in an optimal way is a considerable matter.

3 METHODOLOGY

In the proposed system, we focus on the smart solution to achieve our goal to overcome the current users of other existing application problems and the main problems people face during the time they spend on the road with their vehicle. So, to identify those problems we conducted a survey on that topic as a group and we collected data from the people. In that survey, we were able to find that most people are struggling to find a spot to park in urban cities (Figure 1.5).

So that after the survey period we decide to give the opportunity to the lands or garage owners to register their places in our system. Therefore, to accomplish the requirement we introduce the verification procedure to the land/garage registration. After going through the old researches, we found that the optimal and the novelty method to do this verification process is using image processing based and machine learning-based technology. So based on those technologies planed our methodology to conduct the project.

After that, we search for the resources we need to feed our projects in the technology aspect and the data aspects. We were able to find open-source datasets of road surface type [14], cement pavements [15] and related to potholes on roads [13]. Based on these datasets we were able to accomplished the other tasks of the project After finding the image data resources, we have moved to look at the technology we want. We have identified some main technologies, concepts and algorithms based on image processing and machine learning.

When considering the process of registering a parking lot with the system, it is necessary to ensure that the parking lot surface meets the minimal requirements for parking a vehicle there. In that scenario, the recommended answer was to implement a monthly automated parking quality checking procedure in the system. This strategy works when a parking lot owner is attempting to register his parking lot/garage in the system for renting and performing a quality check of the parking lot at the end of the month in order to keep the business going. As a result, the "Pay as you park" mobile application is used to collect photos of garage/yards and images of the entry road to the garage/yard as part of the yard registration approval process. In this situation, the system must detect the parking yard surface type (asphalt, paved, and unpaved) as well as the quality of each surface type (Standard, Average, and Bad) in order to determine the parking yard surface standard (e.g., Asphalt Standard, paved average, etc).

So, For surface type classification, a custom model was trained using the following classes: asphalt, paved, and unpaved. Our model training and testing were done on a PC with an GeForce GTX 960 graphics card and an 8 GB RAM Windows 10 operating system. In our technique, we employ CNN to perform the parking surface classification job. For each type of surface, three different models were used to determine surface quality. All four models have the same structure. The first model's output is known as the specific quality model. Because the CNN structure does not require the full image to categorize the parking surface, a Region of Interest (ROI) (Figure 6) is defined in the pre-processing stage for each input image before the CNN structure. This ROI's objective is to preserve just the sections of the picture that include parking surface pixels. Because there are unnecessary items in certain frames that have a negative impact on the training model. As a result, the upper half of the image, and also the part of the image in bottom side have been ignored.

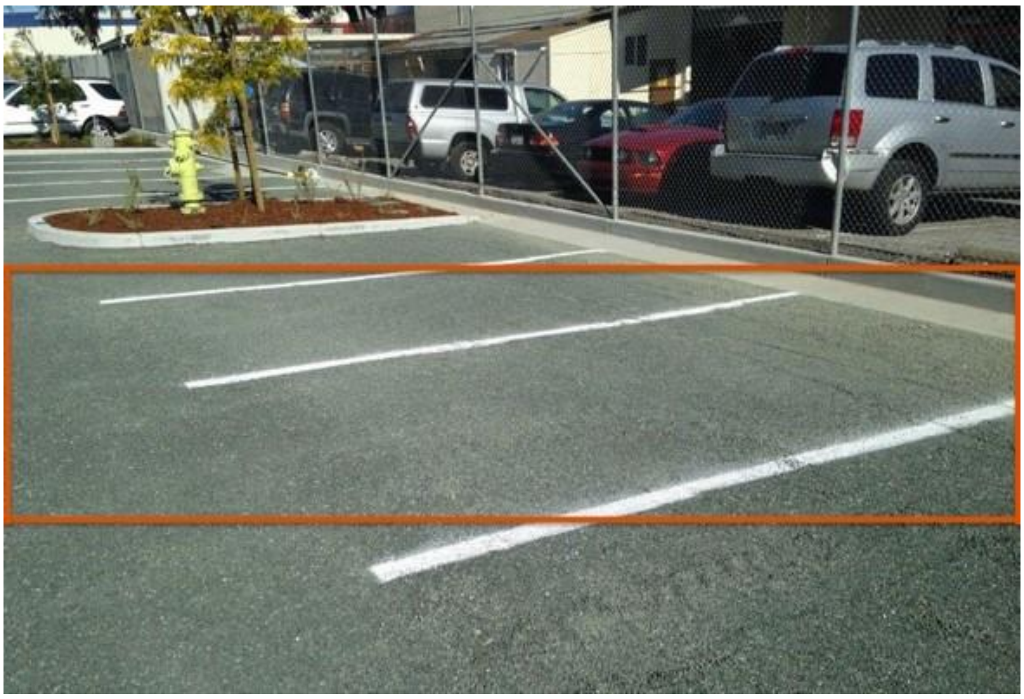


FIGURE 6 REGION-OF-INTEREST.

The brightness of each frame is then raised or reduced as part of the data augmentation, as seen in (Figure 7). This enhances our training input set and helps our system learn to detect the same sort of parking surface under different lighting conditions. Surface pictures are fed into a CNN structure with three convolution layers and two fully connected layers.

Then, during first convolution layer, all images with information on width, height, and number of channels were submitted to the training phase. To minimize input dimensionality, max-pooling is used in all of the convolution layers, which assists in the analysis of feature information in the inputs. After that, ReLU is used as an activation function at the conclusion of each convolution layer.

The flatten layer converts the convolution multi-dimensional tensor to a one-dimensional tensor after the convolutional layers. The final stage is to build two layers that are totally linked. The first fully connected layer utilises a ReLU activation function, while the second fully connected layer cottages the potential outputs, or desired classes, which are divided into three groups: parking surface type models, asphalt quality models, paved quality models, and parking surface unpaved quality models. The SoftMax method was used to compute the likelihood of each class since it makes more sense when subjectivity is absent, such as when determining pavement type. Finally, depending on the input frames utilized during the training phase, the Adam optimizer is employed to change the network weights.

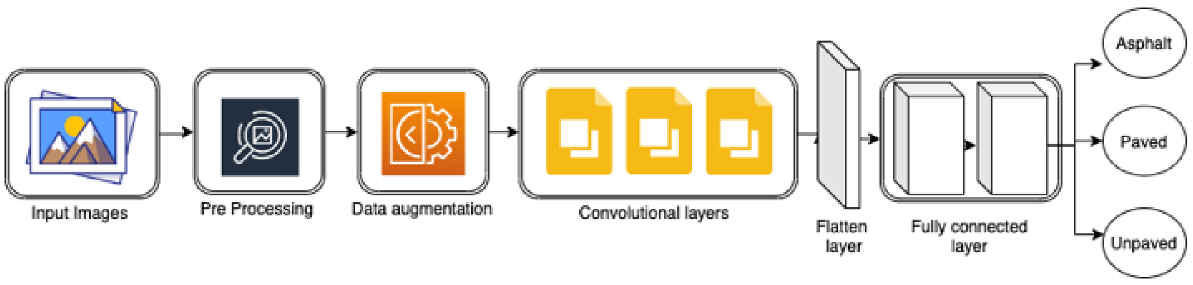


FIGURE 7 FLOWCHART OF PROPOSED WORK

4 TESTING AND IMPLEMENTATION

4.1 Implementation

Pay as You Park smart parking solution system have two applications. One for parking yard owners and it is a web application using MERN stack and mobile application is for the end users using Flutter.

For my research component I have develop the web application to register the parking yard after verifying the place standard by the respective yard owner. Users need an internet browser installed in a device with active internet connection to access the web application.

4.1.1. Web Application Implementation

Web application is developed with Reactjs which is a widely used web application development. Those who utilize ReactJS can expect higher performance than those that use other frameworks. Because ReactJS helps to prevent DOM updates, apps will run faster and provide a better user experience. ReactJS was created with the goal of improving the total number of pages rendered by the website server.

Mongo DB is utilized for the application's database because of its high availability and accuracy. It also includes an object-based database, which is beneficial for mobile platforms. When compared to other cloud services, Mongo offers more freedom. To execute operations with Mongo DB, we used a node.js application at the API level. The API uses node.js to conduct database operations, while Express.js is utilized as a framework for API development. The API project is also launched to Heroku, and the Heroku endpoint is obtained.

And to register parking yard we used Mapbox API for the location service with react and implement the location picking while the place registering to the system. And it also used to visualize the all-registered places in one custom map using the latitude and longitude of the registered places.

In order to validate the parking yard images we had to deploy predictive deep learning model in the flask server using Heroku. After creating the flask API our api could take the image and returned the json result for the further registration process.

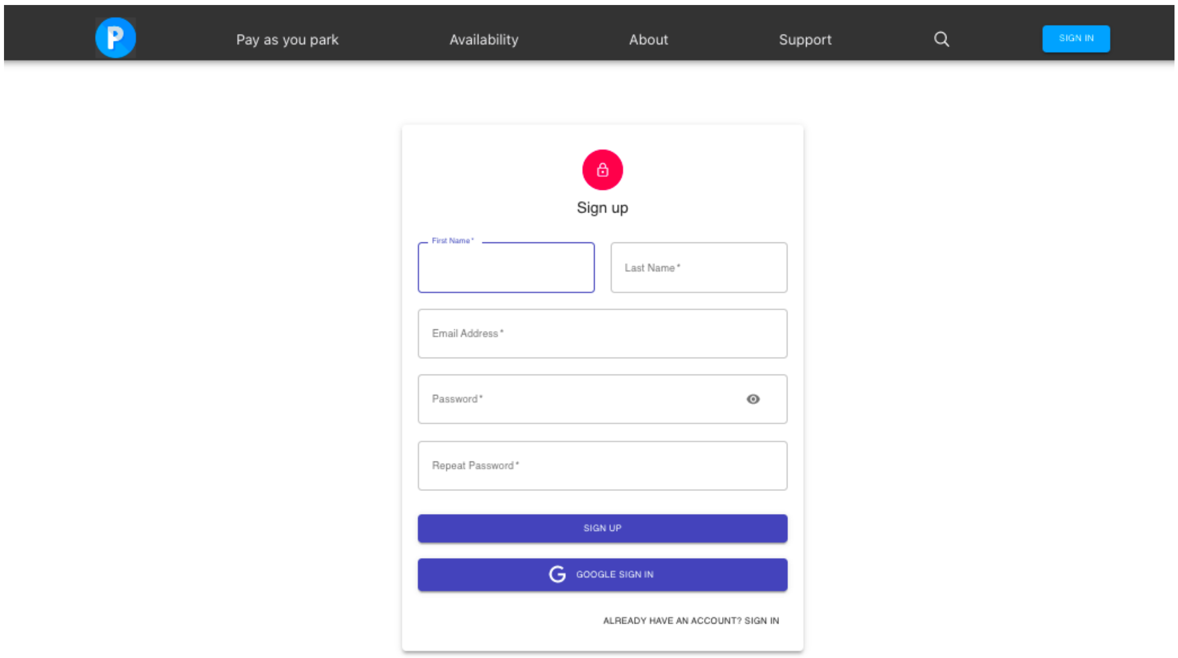
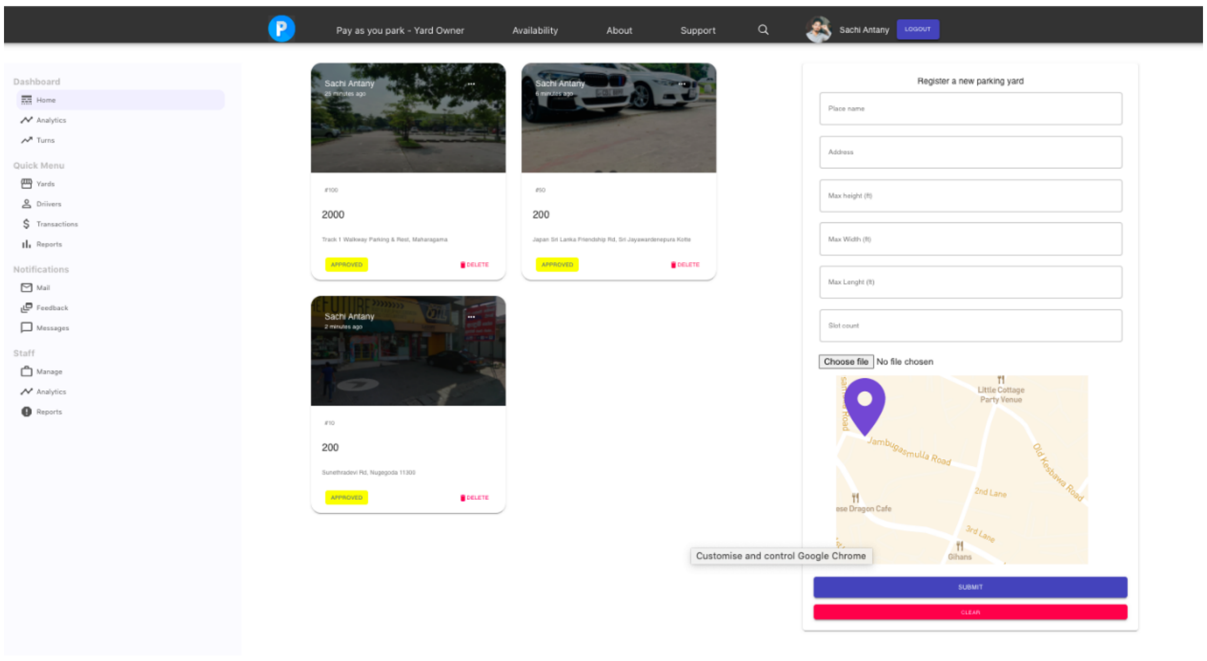


FIGURE 8 REGISTER PARKING YARD OWNER

In figure 9 it visualizes the parking yard owner registration to the system. In this functionality owner can register using his details by typing each field or else owner can register using his own Gmail address by using Google authentication.



Sachi Antany
25 minutes ago
#100
2000
Track 1 Walkway Parking & Rest, Maharagama
APPROVED DELETE

Sachi Antany
6 minutes ago
#50
200
Japan Sri Lanka Friendship Rd, Sri Jayawardenepura Kotte
APPROVED DELETE

Sachi Antany
2 minutes ago
#10
200
Sunethradevi Rd, Nugegoda 11300
APPROVED DELETE

FIGURE 9 REGISTER PARKING YARD

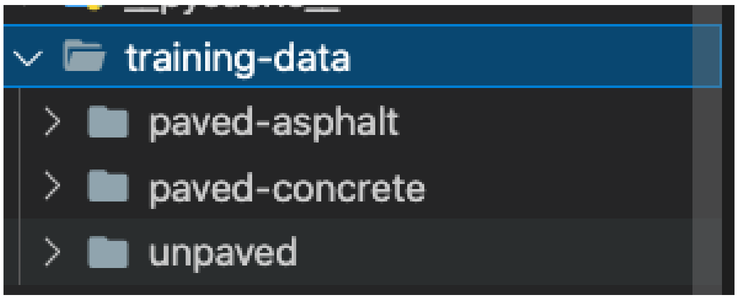
Once after register the owner, owner can register his parking yards to the system by giving necessary information to the system. In that case, we are asking for parking place name, Address of the place, Dimension of the parking place (Height, Width, Length), Parking slot count, Images of the parking yard and pick the location from the map. Initially map picker ask for GPS access for the location picking and gives the current location for the registration, but owner can change it as their wish by navigation the marker to the correct place. After that, data push to the mongo db and image data send to the

4.1.2. Parking Surface Quality Detection Model Implementation

To accomplish the quality measuring of the new parking yards for register in the system, I have implemented the CNN structure to classify the surface and the detect the quality of the surface using the yard images collected by from the parking yard owner. For this purpose, I have to train two types of models, which are parking yard classification model set and parking yard quality classification model.

- **Parking Surface Classification**

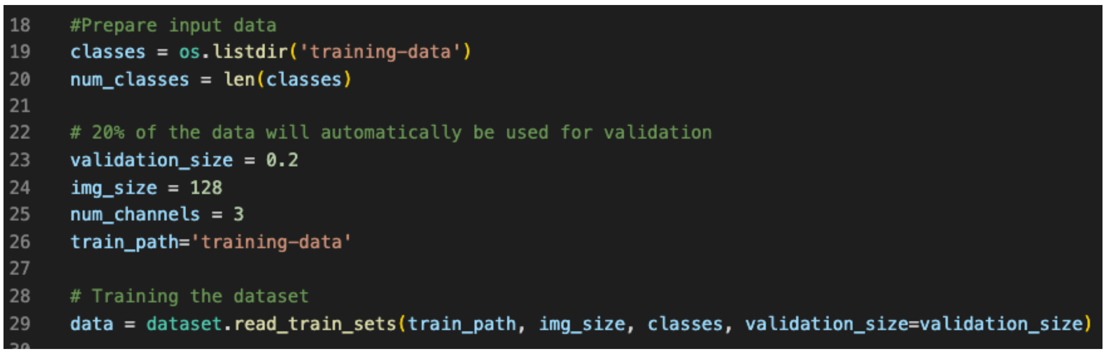
First of all, we have to prepare the data before the model training. So that collected data has categorized to three separate folders with the surface type name (Figure 10)



```
training-data
  paved-asphalt
  paved-concrete
  unpaved
```

FIGURE 10 TRAINING DATA FOLDER CATEGORIZING

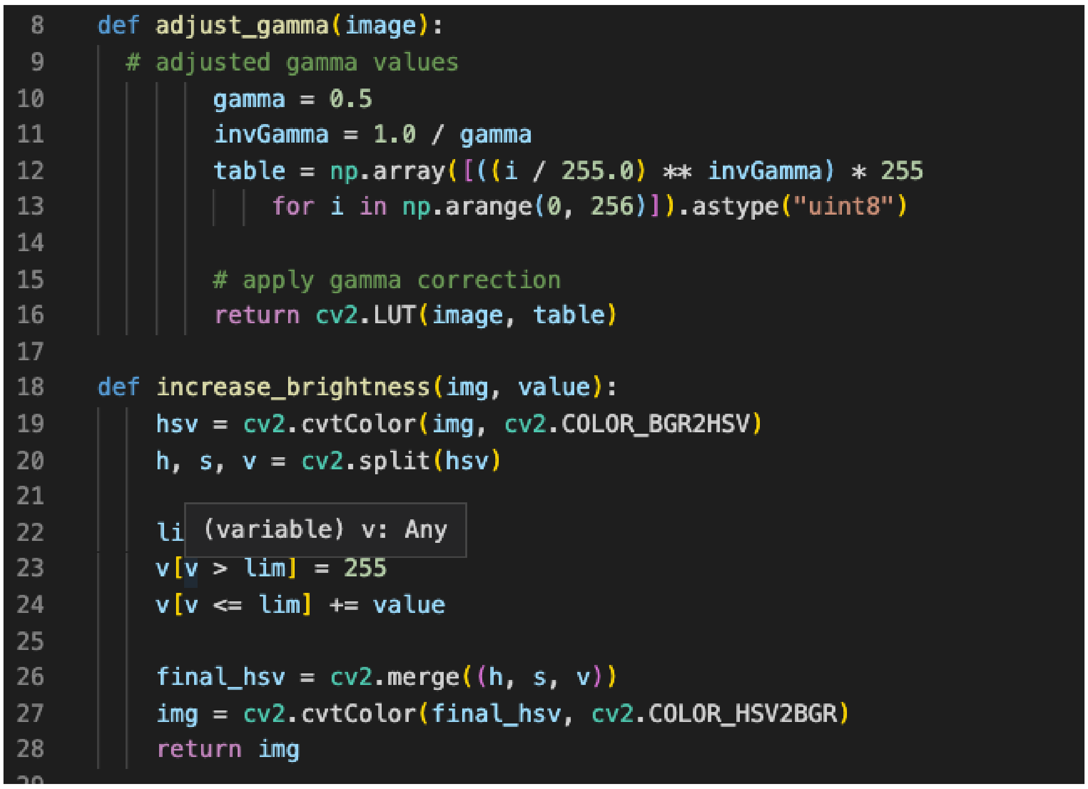
Then collected data separated to 20% for automictically validation, set batch size as 32 and then data pushed thorough the dataset.py class for the further training data utilization (Figure 11)



```
#Prepare input data
classes = os.listdir('training-data')
num_classes = len(classes)

# 20% of the data will automatically be used for validation
validation_size = 0.2
img_size = 128
num_channels = 3
train_path='training-data'

# Training the dataset
data = dataset.read_train_sets(train_path, img_size, classes, validation_size=validation_size)
```

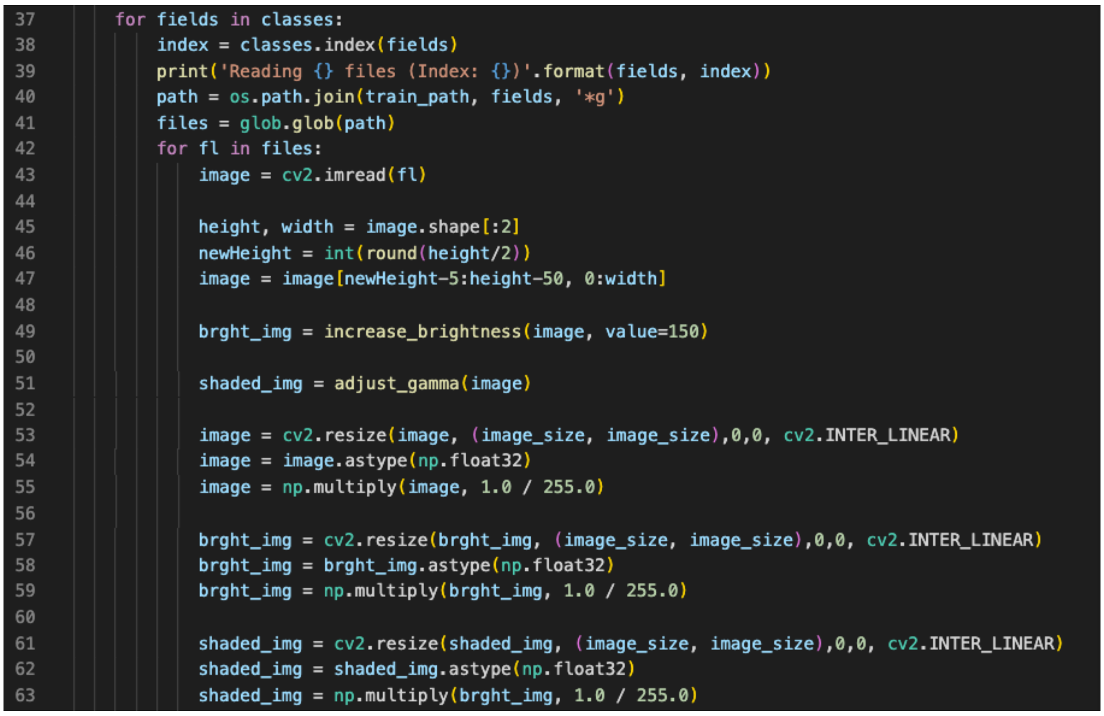
FIGURE 11 TRAIN.PY CODE PART 1

Using the dataset.py, we have adjusted the dataset quality and utilize the data frames with ROI and the data argumentation. (Figure 12 & 13)



```python
def adjust_gamma(image):
    # adjusted gamma values
                gamma = 0.5
                invGamma = 1.0 / gamma
                table = np.array([((i / 255.0) ** invGamma) * 255
                        for i in np.arange(0, 256)]).astype("uint8")

                # apply gamma correction
                return cv2.LUT(image, table)

def increase_brightness(img, value):
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        h, s, v = cv2.split(hsv)

        li (variable) v: Any
        v[v > lim] = 255
        v[v <= lim] += value

        final_hsv = cv2.merge((h, s, v))
        img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
        return img
```

FIGURE 12 DATASET.PY CODE PART 1

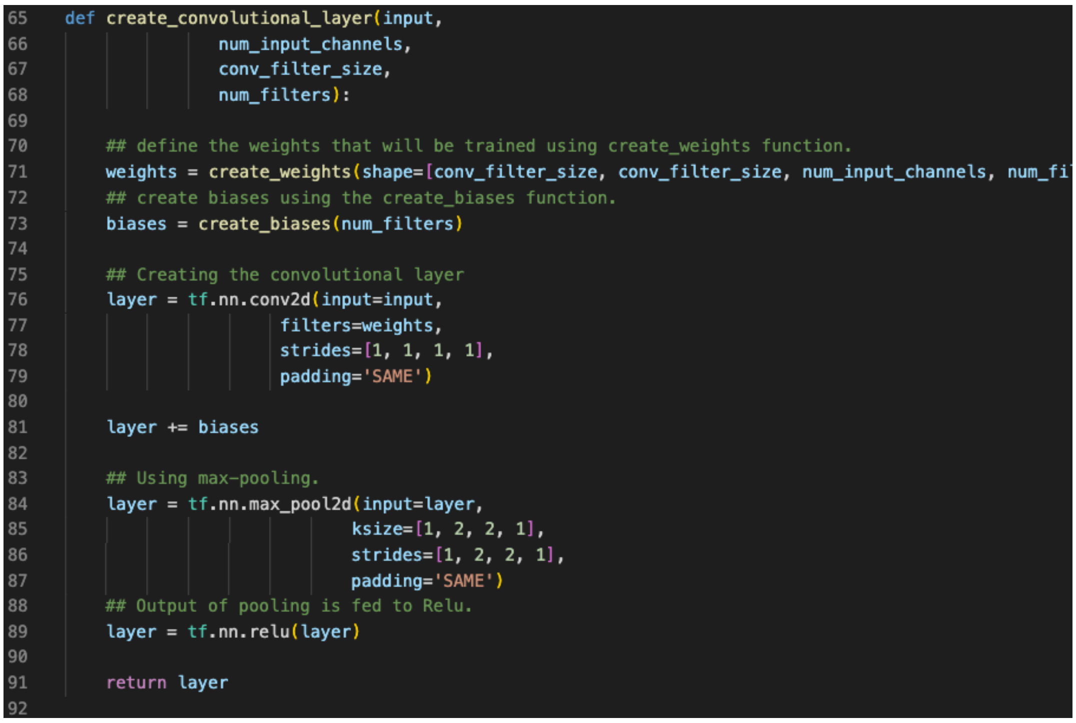


```
        for fields in classes:
            index = classes.index(fields)
            print('Reading {} files (Index: {})'.format(fields, index))
            path = os.path.join(train_path, fields, '*g')
            files = glob.glob(path)
            for fl in files:
                image = cv2.imread(fl)

                height, width = image.shape[:2]
                newHeight = int(round(height/2))
                image = image[newHeight-5:height-50, 0:width]

                brght_img = increase_brightness(image, value=150)

                shaded_img = adjust_gamma(image)

                image = cv2.resize(image, (image_size, image_size),0,0, cv2.INTER_LINEAR)
                image = image.astype(np.float32)
                image = np.multiply(image, 1.0 / 255.0)

                brght_img = cv2.resize(brght_img, (image_size, image_size),0,0, cv2.INTER_LINEAR)
                brght_img = brght_img.astype(np.float32)
                brght_img = np.multiply(brght_img, 1.0 / 255.0)

                shaded_img = cv2.resize(shaded_img, (image_size, image_size),0,0, cv2.INTER_LINEAR)
                shaded_img = shaded_img.astype(np.float32)
                shaded_img = np.multiply(brght_img, 1.0 / 255.0)
```

FIGURE 13 DATASET.PY CODE PART 2

Next, we defined CNN layers in the train.py and passed all images through those layers (Figure 14).

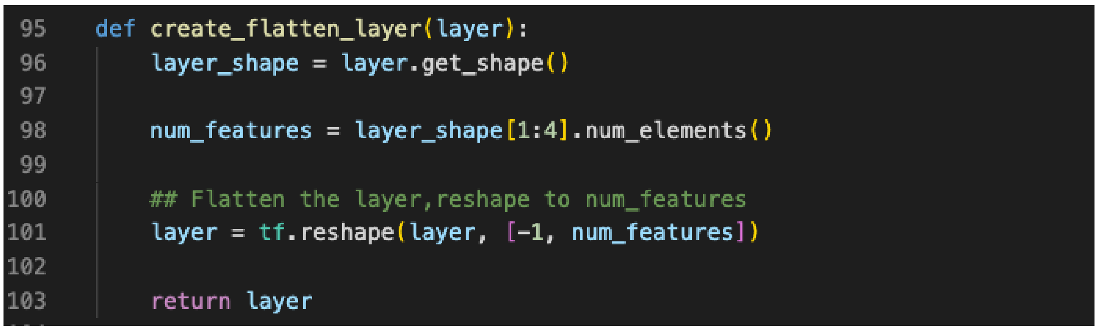
First two layers contain 32 filters with size of 3 by 3. Padding was set to 0 and all strides were set to 1. The weights are initialized using a normal distribution. A max-pooling is performed to all the convolution layers in order to decrease the inputs dimensionally, which aids in the analysis of features contained in the input sub-regions. A ReLU is used as an activation function at the conclusion of each convolution layer, following the max-pooling function.



```
def create_convolutional_layer(input,
               num_input_channels,
               conv_filter_size,
               num_filters):

    ## define the weights that will be trained using create_weights function.
    weights = create_weights(shape=[conv_filter_size, conv_filter_size, num_input_channels, num_fi
    ## create biases using the create_biases function.
    biases = create_biases(num_filters)

    ## Creating the convolutional layer
    layer = tf.nn.conv2d(input=input,
                     filters=weights,
                     strides=[1, 1, 1, 1],
                     padding='SAME')

    layer += biases

    ## Using max-pooling.
    layer = tf.nn.max_pool2d(input=layer,
                            ksize=[1, 2, 2, 1],
                            strides=[1, 2, 2, 1],
                            padding='SAME')
    ## Output of pooling is fed to Relu.
    layer = tf.nn.relu(layer)

    return layer
```

FIGURE 14 TRAIN.PY CODE PART 2

Afterward, the flatten layer is used to convert the convolution multi-dimensional tensor into a one-dimensional tensor after the convolutional layers (Figure 15).



```
def create_flatten_layer(layer):
    layer_shape = layer.get_shape()

    num_features = layer_shape[1:4].num_elements()

    ## Flatten the layer,reshape to num_features
    layer = tf.reshape(layer, [-1, num_features])

    return layer
```

FIGURE 15 TRAIN.PY CODE PART 2

At the end, two fully connected layers are added and a ReLU activation function is used after those layers (Figure 16).

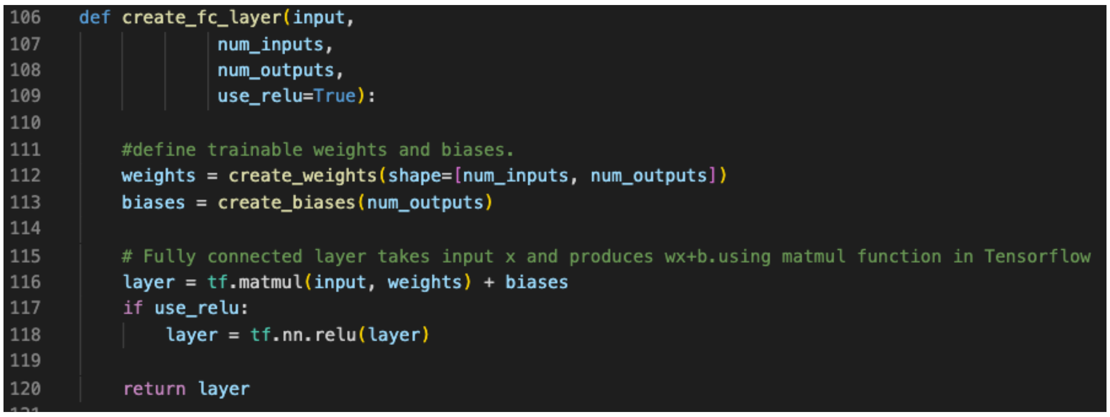
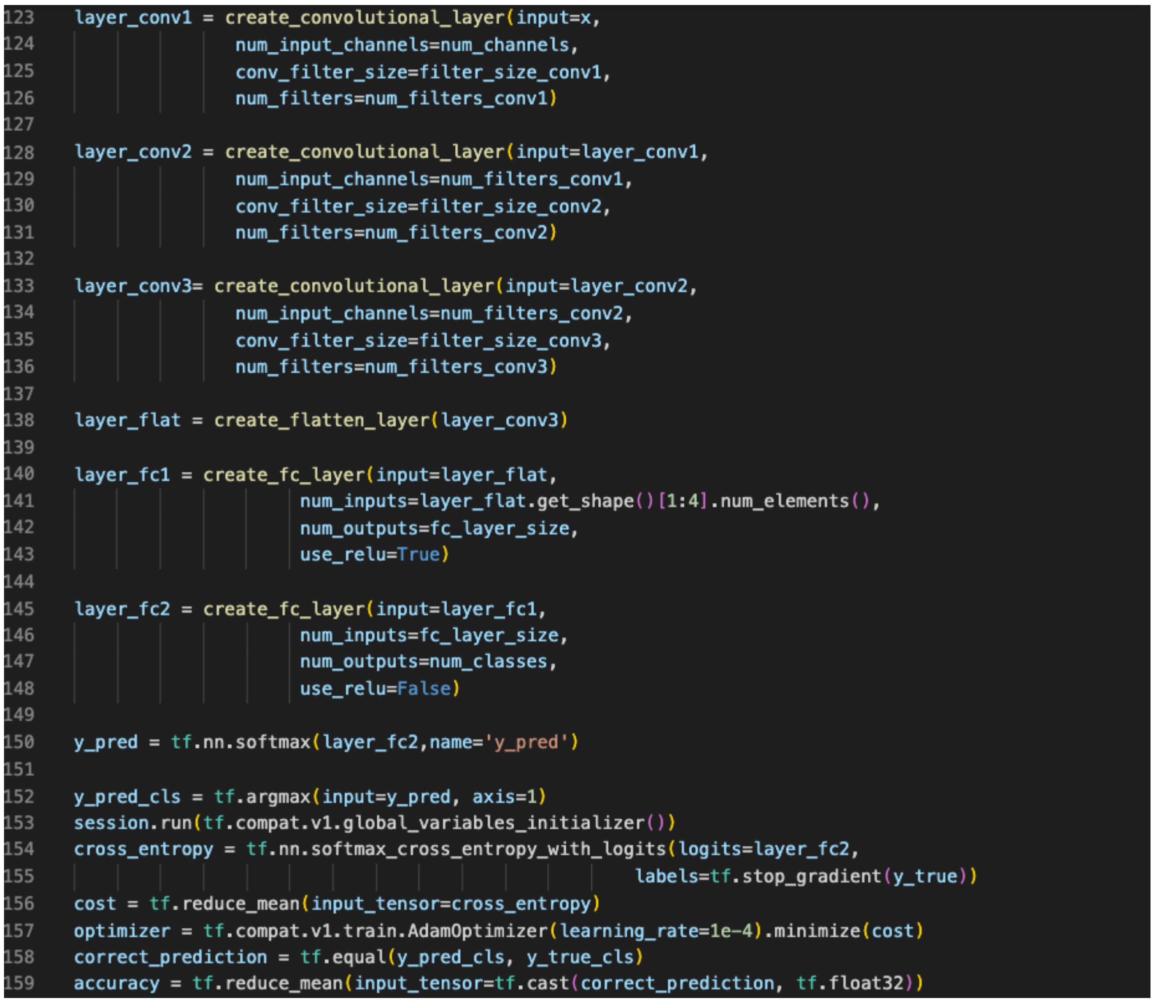

```python
def create_fc_layer(input,
                    num_inputs,
                    num_outputs,
                    use_relu=True):

    #define trainable weights and biases.
    weights = create_weights(shape=[num_inputs, num_outputs])
    biases = create_biases(num_outputs)

    # Fully connected layer takes input x and produces wx+b.using matmul function in Tensorflow
    layer = tf.matmul(input, weights) + biases
    if use_relu:
        layer = tf.nn.relu(layer)

    return layer
```

FIGURE 16 TRAIN.PY CODE PART 3

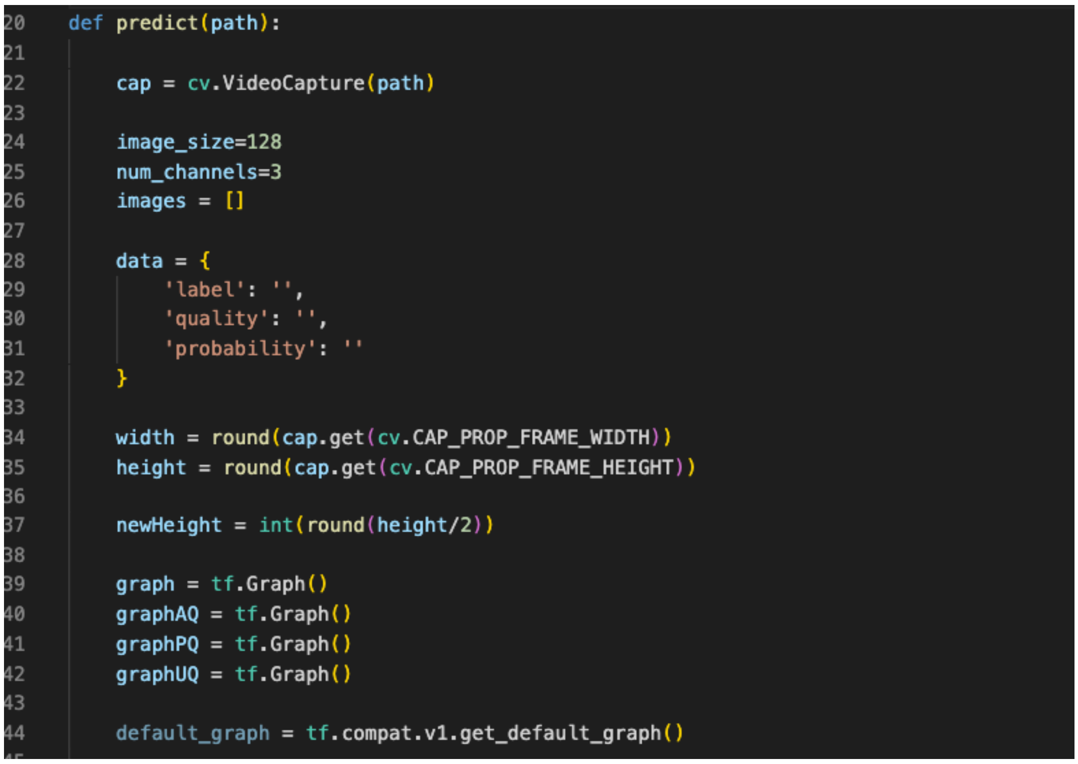
The probability of each class was calculated using the SoftMax algorithm. At the end, we utilize the Adam optimizer to adjust the network weights based on the input data used during training. After that we trained the model using the Train.py python script and prepared the model for performance (Figure 17).



```
layer_conv1 = create_convolutional_layer(input=x,
               num_input_channels=num_channels,
               conv_filter_size=filter_size_conv1,
               num_filters=num_filters_conv1)

layer_conv2 = create_convolutional_layer(input=layer_conv1,
               num_input_channels=num_filters_conv1,
               conv_filter_size=filter_size_conv2,
               num_filters=num_filters_conv2)

layer_conv3= create_convolutional_layer(input=layer_conv2,
               num_input_channels=num_filters_conv2,
               conv_filter_size=filter_size_conv3,
               num_filters=num_filters_conv3)

layer_flat = create_flatten_layer(layer_conv3)

layer_fc1 = create_fc_layer(input=layer_flat,
                     num_inputs=layer_flat.get_shape()[1:4].num_elements(),
                     num_outputs=fc_layer_size,
                     use_relu=True)

layer_fc2 = create_fc_layer(input=layer_fc1,
                     num_inputs=fc_layer_size,
                     num_outputs=num_classes,
                     use_relu=False)

y_pred = tf.nn.softmax(layer_fc2,name='y_pred')

y_pred_cls = tf.argmax(input=y_pred, axis=1)
session.run(tf.compat.v1.global_variables_initializer())
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc2,
                                                          labels=tf.stop_gradient(y_true))
cost = tf.reduce_mean(input_tensor=cross_entropy)
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(input_tensor=tf.cast(correct_prediction, tf.float32))
```

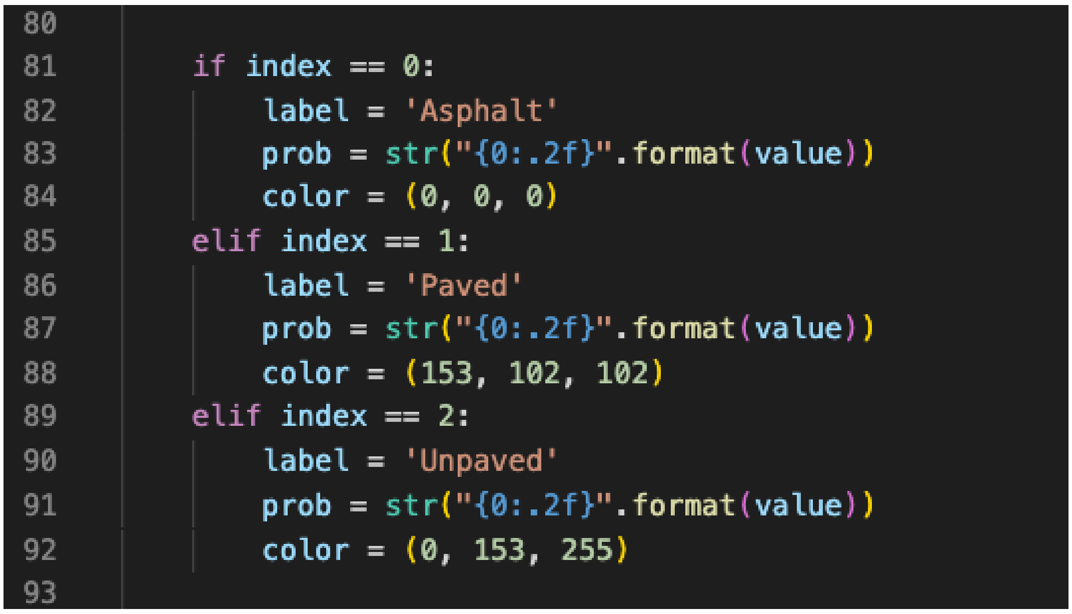
FIGURE 17 TRAIN.PY CODE PART 4

Afterward we define a class in test.py script to input the link of the image and read the image data to evaluate the result using the graphs of trained models (Figure 18). Then test.py script return the type of the surface (Asphalt, paved, unpaved) by predicting the image (Figure 19).



```python
def predict(path):

    cap = cv.VideoCapture(path)

    image_size=128
    num_channels=3
    images = []

    data = {
        'label': '',
        'quality': '',
        'probability': ''
    }

    width = round(cap.get(cv.CAP_PROP_FRAME_WIDTH))
    height = round(cap.get(cv.CAP_PROP_FRAME_HEIGHT))

    newHeight = int(round(height/2))

    graph = tf.Graph()
    graphAQ = tf.Graph()
    graphPQ = tf.Graph()
    graphUQ = tf.Graph()

    default_graph = tf.compat.v1.get_default_graph()
```

FIGURE 18 TEST.PY CODE PART 1

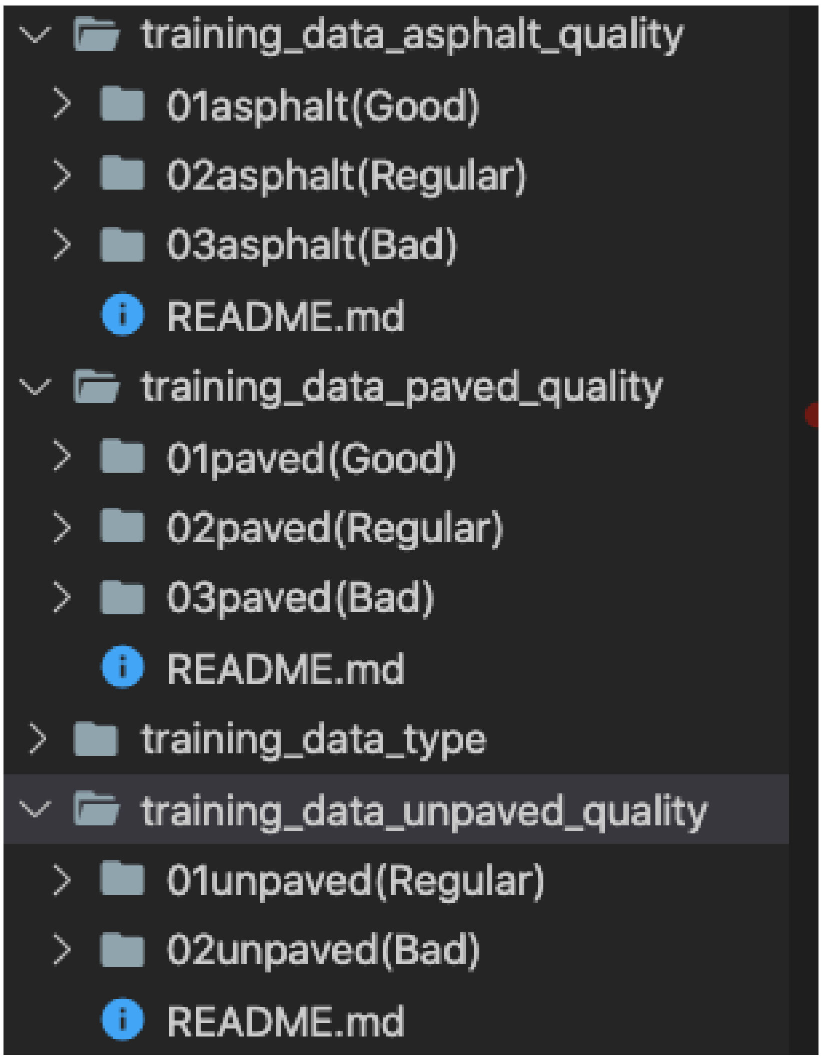


```python

        if index == 0:
            label = 'Asphalt'
            prob = str("{0:.2f}".format(value))
            color = (0, 0, 0)
        elif index == 1:
            label = 'Paved'
            prob = str("{0:.2f}".format(value))
            color = (153, 102, 102)
        elif index == 2:
            label = 'Unpaved'
            prob = str("{0:.2f}".format(value))
            color = (0, 153, 255)

```

FIGURE 19 TEST.PY CODE PART 2

- **Parking Surface and Quality Classification**

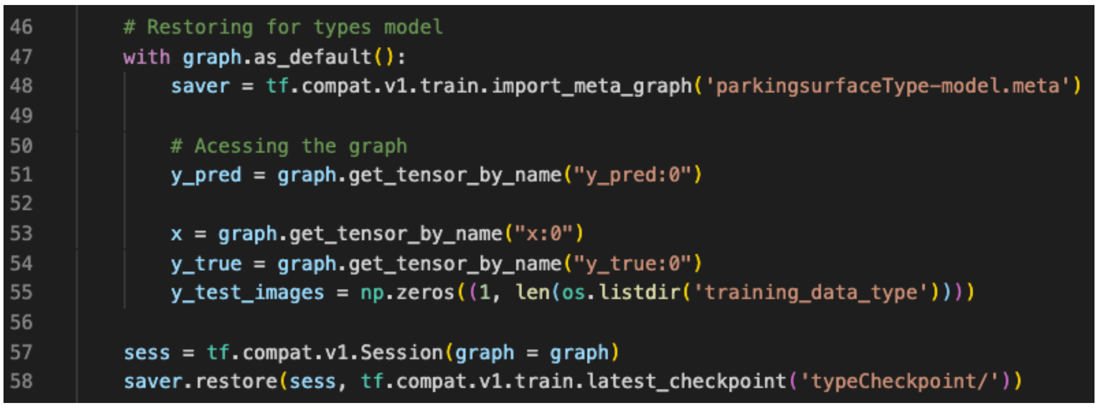
When classifying the quality of each surface type I used the same technic used in the above model which is classifying the surface type. For this work, all training images categorized according to the (Figure 20). Then trained three models for each surface types qualities.



```
training_data_asphalt_quality
    01asphalt(Good)
    02asphalt(Regular)
    03asphalt(Bad)
    README.md
training_data_paved_quality
    01paved(Good)
    02paved(Regular)
    03paved(Bad)
    README.md
training_data_type
training_data_unpaved_quality
    01unpaved(Regular)
    02unpaved(Bad)
    README.md
```

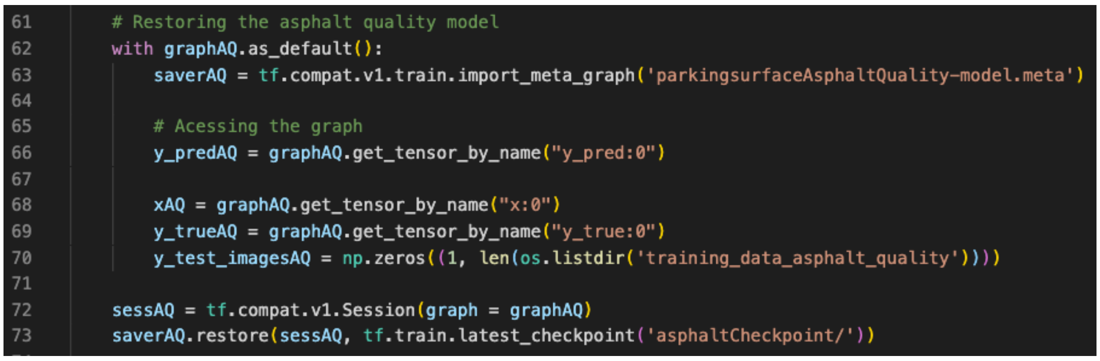
FIGURE 20 SURFACE QUALITY CLASSIFICATION FOLDER STRUCTURE

Then I used the surface type classification model to identify the surface type and the used other three surface quality models to identify the quality of each surface.



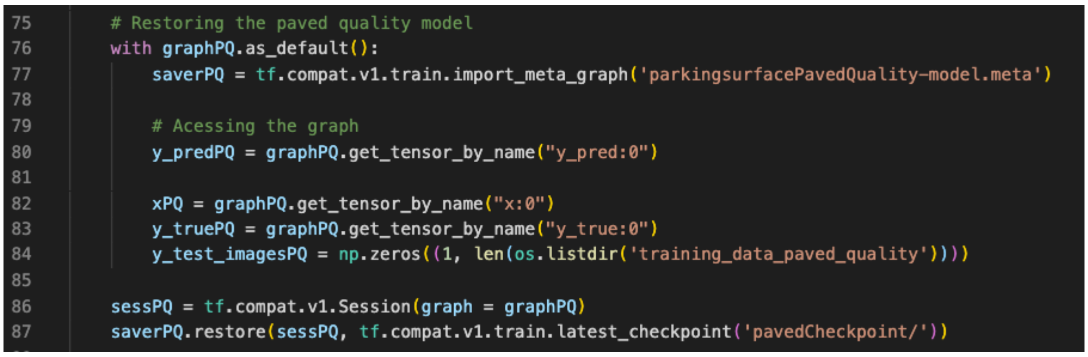
```python
    # Restoring for types model
    with graph.as_default():
        saver = tf.compat.v1.train.import_meta_graph('parkingsurfaceType-model.meta')

        # Acessing the graph
        y_pred = graph.get_tensor_by_name("y_pred:0")

        x = graph.get_tensor_by_name("x:0")
        y_true = graph.get_tensor_by_name("y_true:0")
        y_test_images = np.zeros((1, len(os.listdir('training_data_type'))))

    sess = tf.compat.v1.Session(graph = graph)
    saver.restore(sess, tf.compat.v1.train.latest_checkpoint('typeCheckpoint/'))
```

FIGURE 21 RESTORING THE SURFACE TYPE MODEL



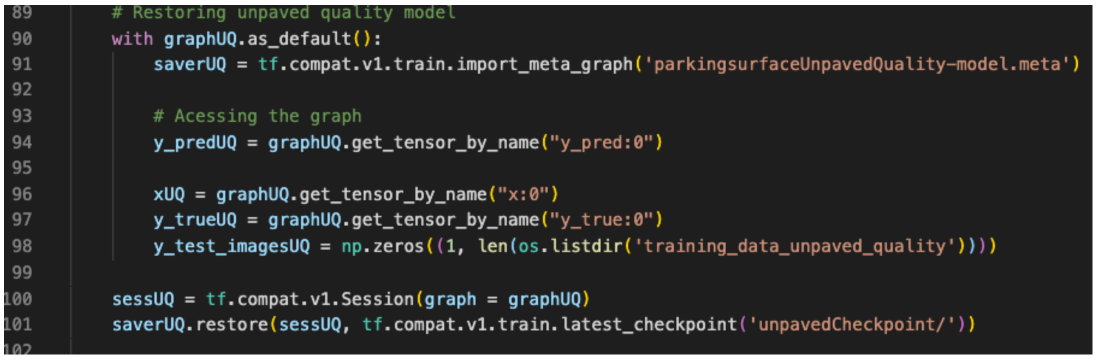
```python
    # Restoring the asphalt quality model
    with graphAQ.as_default():
        saverAQ = tf.compat.v1.train.import_meta_graph('parkingsurfaceAsphaltQuality-model.meta')

        # Acessing the graph
        y_predAQ = graphAQ.get_tensor_by_name("y_pred:0")

        xAQ = graphAQ.get_tensor_by_name("x:0")
        y_trueAQ = graphAQ.get_tensor_by_name("y_true:0")
        y_test_imagesAQ = np.zeros((1, len(os.listdir('training_data_asphalt_quality'))))

    sessAQ = tf.compat.v1.Session(graph = graphAQ)
    saverAQ.restore(sessAQ, tf.train.latest_checkpoint('asphaltCheckpoint/'))
```

FIGURE 22 RESTORING THE ASPHALT QUALITY MODEL



```python
# Restoring the paved quality model
with graphPQ.as_default():
    saverPQ = tf.compat.v1.train.import_meta_graph('parkingsurfacePavedQuality-model.meta')

    # Acessing the graph
    y_predPQ = graphPQ.get_tensor_by_name("y_pred:0")

    xPQ = graphPQ.get_tensor_by_name("x:0")
    y_truePQ = graphPQ.get_tensor_by_name("y_true:0")
    y_test_imagesPQ = np.zeros((1, len(os.listdir('training_data_paved_quality'))))

sessPQ = tf.compat.v1.Session(graph = graphPQ)
saverPQ.restore(sessPQ, tf.compat.v1.train.latest_checkpoint('pavedCheckpoint/'))
```

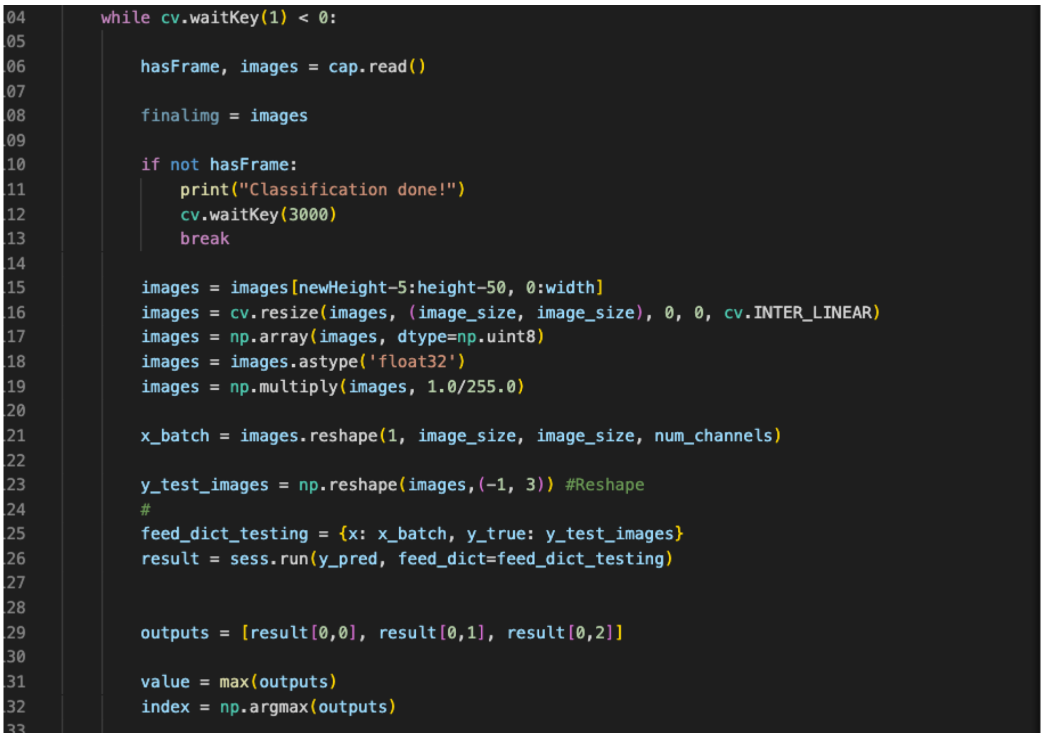
FIGURE 23 RESTORING THE PAVED QUALITY MODEL



```python
# Restoring unpaved quality model
with graphUQ.as_default():
    saverUQ = tf.compat.v1.train.import_meta_graph('parkingsurfaceUnpavedQuality-model.meta')

    # Acessing the graph
    y_predUQ = graphUQ.get_tensor_by_name("y_pred:0")

    xUQ = graphUQ.get_tensor_by_name("x:0")
    y_trueUQ = graphUQ.get_tensor_by_name("y_true:0")
    y_test_imagesUQ = np.zeros((1, len(os.listdir('training_data_unpaved_quality'))))

sessUQ = tf.compat.v1.Session(graph = graphUQ)
saverUQ.restore(sessUQ, tf.compat.v1.train.latest_checkpoint('unpavedCheckpoint/'))
```

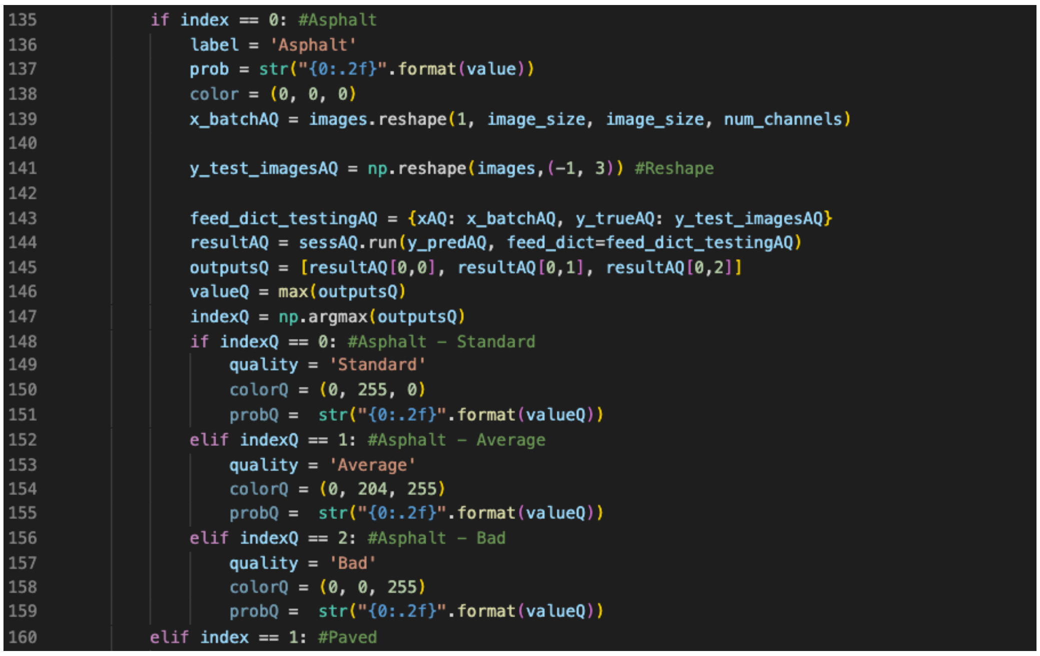
FIGURE 24 RESTORING THE UNPAVED QUALITY MODEL

Using the input image data this script now classifying the surface type and its relevant quality in just a few seconds. Once after the classification we return the all the classified information as object.



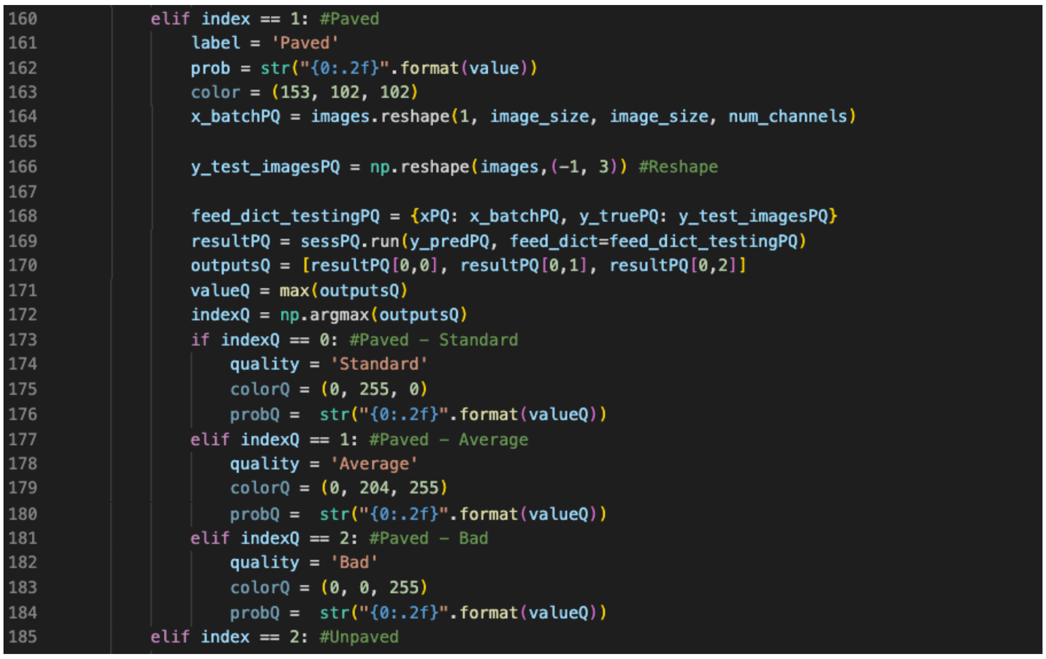
```
    while cv.waitKey(1) < 0:

        hasFrame, images = cap.read()

        finalimg = images

        if not hasFrame:
            print("Classification done!")
            cv.waitKey(3000)
            break

        images = images[newHeight-5:height-50, 0:width]
        images = cv.resize(images, (image_size, image_size), 0, 0, cv.INTER_LINEAR)
        images = np.array(images, dtype=np.uint8)
        images = images.astype('float32')
        images = np.multiply(images, 1.0/255.0)

        x_batch = images.reshape(1, image_size, image_size, num_channels)

        y_test_images = np.reshape(images,(-1, 3)) #Reshape
        #
        feed_dict_testing = {x: x_batch, y_true: y_test_images}
        result = sess.run(y_pred, feed_dict=feed_dict_testing)


        outputs = [result[0,0], result[0,1], result[0,2]]

        value = max(outputs)
        index = np.argmax(outputs)
```

FIGURE 25 MAIN.PY CODE PART 1



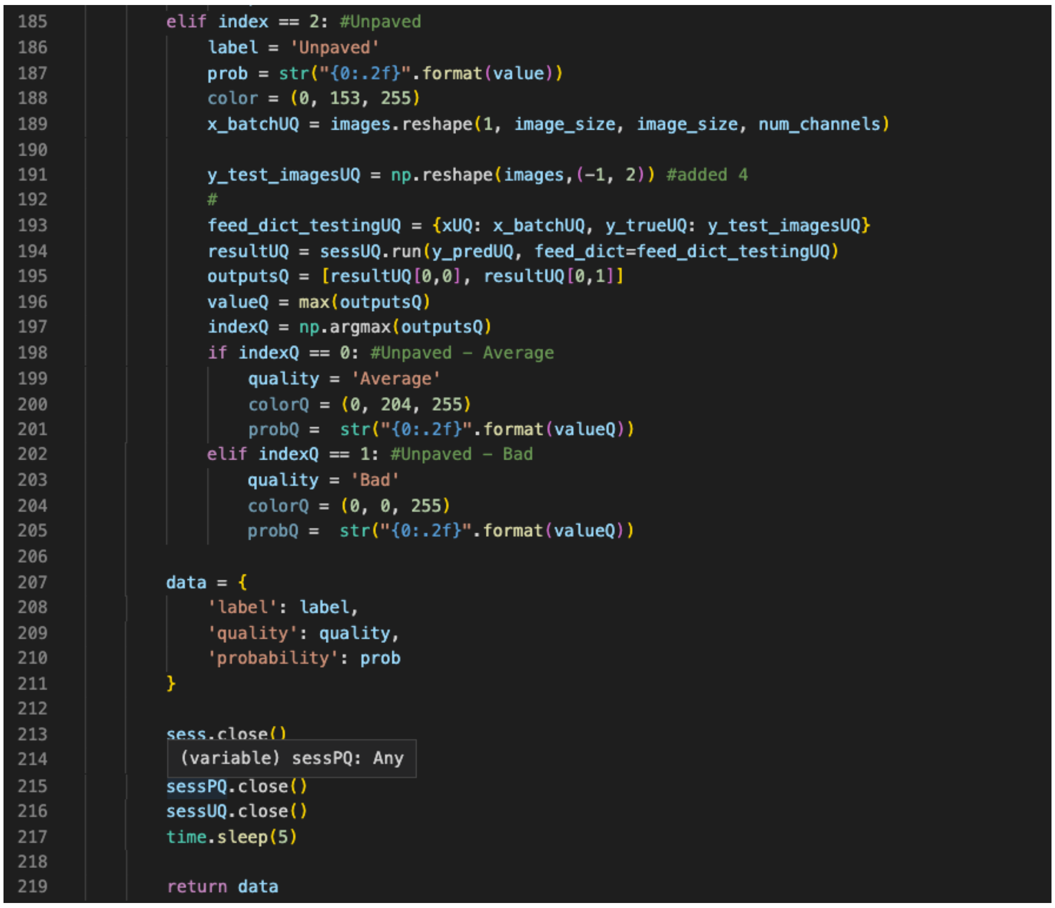
```
        if index == 0: #Asphalt
            label = 'Asphalt'
            prob = str("{0:.2f}".format(value))
            color = (0, 0, 0)
            x_batchAQ = images.reshape(1, image_size, image_size, num_channels)

            y_test_imagesAQ = np.reshape(images,(-1, 3)) #Reshape

            feed_dict_testingAQ = {xAQ: x_batchAQ, y_trueAQ: y_test_imagesAQ}
            resultAQ = sessAQ.run(y_predAQ, feed_dict=feed_dict_testingAQ)
            outputsQ = [resultAQ[0,0], resultAQ[0,1], resultAQ[0,2]]
            valueQ = max(outputsQ)
            indexQ = np.argmax(outputsQ)
            if indexQ == 0: #Asphalt - Standard
                quality = 'Standard'
                colorQ = (0, 255, 0)
                probQ =  str("{0:.2f}".format(valueQ))
            elif indexQ == 1: #Asphalt - Average
                quality = 'Average'
                colorQ = (0, 204, 255)
                probQ =  str("{0:.2f}".format(valueQ))
            elif indexQ == 2: #Asphalt - Bad
                quality = 'Bad'
                colorQ = (0, 0, 255)
                probQ =  str("{0:.2f}".format(valueQ))
        elif index == 1: #Paved
```

FIGURE 26 MAIN.PY CODE PART 2



```
        elif index == 1: #Paved
            label = 'Paved'
            prob = str("{0:.2f}".format(value))
            color = (153, 102, 102)
            x_batchPQ = images.reshape(1, image_size, image_size, num_channels)

            y_test_imagesPQ = np.reshape(images,(-1, 3)) #Reshape

            feed_dict_testingPQ = {xPQ: x_batchPQ, y_truePQ: y_test_imagesPQ}
            resultPQ = sessPQ.run(y_predPQ, feed_dict=feed_dict_testingPQ)
            outputsQ = [resultPQ[0,0], resultPQ[0,1], resultPQ[0,2]]
            valueQ = max(outputsQ)
            indexQ = np.argmax(outputsQ)
            if indexQ == 0: #Paved - Standard
                quality = 'Standard'
                colorQ = (0, 255, 0)
                probQ =  str("{0:.2f}".format(valueQ))
            elif indexQ == 1: #Paved - Average
                quality = 'Average'
                colorQ = (0, 204, 255)
                probQ =  str("{0:.2f}".format(valueQ))
            elif indexQ == 2: #Paved - Bad
                quality = 'Bad'
                colorQ = (0, 0, 255)
                probQ =  str("{0:.2f}".format(valueQ))
        elif index == 2: #Unpaved
```

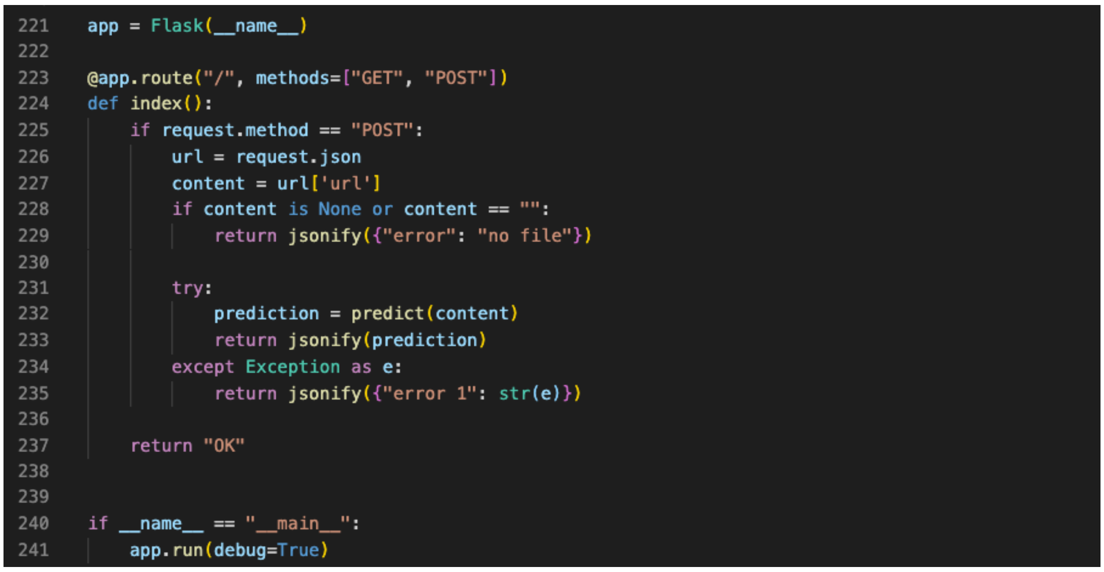
Figure 27 Main.py code part 3



```
        elif index == 2: #Unpaved
            label = 'Unpaved'
            prob = str("{0:.2f}".format(value))
            color = (0, 153, 255)
            x_batchUQ = images.reshape(1, image_size, image_size, num_channels)

            y_test_imagesUQ = np.reshape(images,(-1, 2)) #added 4
            #
            feed_dict_testingUQ = {xUQ: x_batchUQ, y_trueUQ: y_test_imagesUQ}
            resultUQ = sessUQ.run(y_predUQ, feed_dict=feed_dict_testingUQ)
            outputsQ = [resultUQ[0,0], resultUQ[0,1]]
            valueQ = max(outputsQ)
            indexQ = np.argmax(outputsQ)
            if indexQ == 0: #Unpaved - Average
                quality = 'Average'
                colorQ = (0, 204, 255)
                probQ =  str("{0:.2f}".format(valueQ))
            elif indexQ == 1: #Unpaved - Bad
                quality = 'Bad'
                colorQ = (0, 0, 255)
                probQ =  str("{0:.2f}".format(valueQ))

        data = {
            'label': label,
            'quality': quality,
            'probability': prob
        }

        sess.close()
         (variable) sessPQ: Any
        sessPQ.close()
        sessUQ.close()
        time.sleep(5)

        return data
```

FIGURE 28 MAIN.PY CODE PART 4

Finally, we have created flask API using the above created classification script to use in the pay as you park web application surface quality identification before the parking place registration (Figure 29)



```
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        url = request.json
        content = url['url']
        if content is None or content == "":
            return jsonify({"error": "no file"})

        try:
            prediction = predict(content)
            return jsonify(prediction)
        except Exception as e:
            return jsonify({"error 1": str(e)})

    return "OK"


if __name__ == "__main__":
    app.run(debug=True)
```

FIGURE 29 FLASK IMPLEMENTATION

4.2. Testing

- **Parking Surface Classification**

For the training purposes, more than 6000+ frames were used from our dataset to train our model for parking surface classification, which were manually divided into three categories as asphalt, paved, and unpaved. All training data were divided into two parts such as 80% for training and 20% for validation. The number of frames chosen for each class was proportionate to the amount of data available. Around 70% of the frames are asphalt, 20% are paved, and 10% are unpaved.

Model Name	Training Accuracy	Validation Accuracy	Validation Loss
Surface Type Model	100%	93.8%	0.146

TABLE 2 PARKING SURFACE TYPE MODEL ACCURACY

- **Parking Surface and Quality Classification**

When consider the surface quality, this method used three models for the surface quality classification which is used to identify the quality of each surface type. In the training stage, we have used a few frames: 1500+ frames for the asphalt quality model, 600+ frames for the paved quality model, and 500+ frames for the unpaved quality model. We separated 80 percent of the data for training, and the remaining 20 % was picked at random for validation.

Quality Models	Training Accuracy	Validation Accuracy	Validation Loss
Asphalt	100%	88.4%	0.012
Paved	100%	96.8%	0.180
Unpaved	100%	92.8%	0.205

TABLE 3 PARKING SURFACE QUALITY MODEL ACCURACY

5 RESULT AND DISCUSSION

5.1 Results

In the research I have conducted, two types of models were created to accomplish the pay as you park requirement. After the research I have developed four models called Surface Type Model, Asphalt Quality Model, Paved Quality Model and Unpaved Quality Model. Using those models, it can identify;

- Parking yard surface category identification
    - Asphalt surface
    - Paved surface (concreate surfaces, marble surfaces etc)
    - Unpaved surface (gravel surfaces, sandy surfaces etc)

- Parking yard quality identification
    - Standard surface detection
        - Quality surface
        - Standard parking yard
    - Average surface detection
        - Pothole detection
        - Surface damages detection
        -
    - Bad surface detection
        - Pothole detection
        - Surface damages detection

After the testing of the classification model, I have printed those results in the images itself as the visual output (Figure 30,31,32).

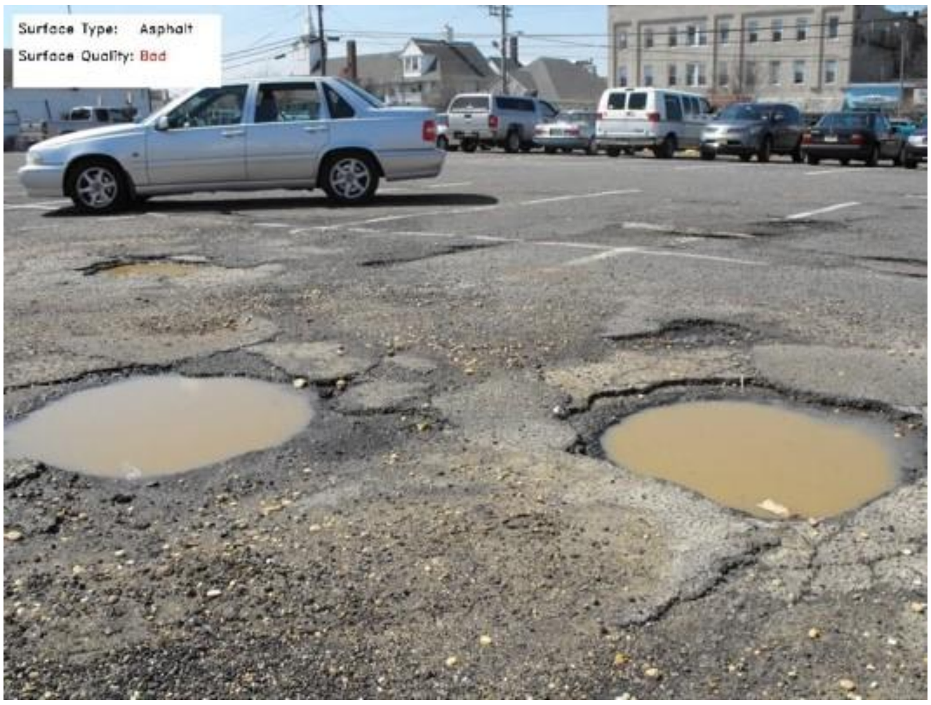


FIGURE 30 OUTPUT (ASPHALT - BAD)

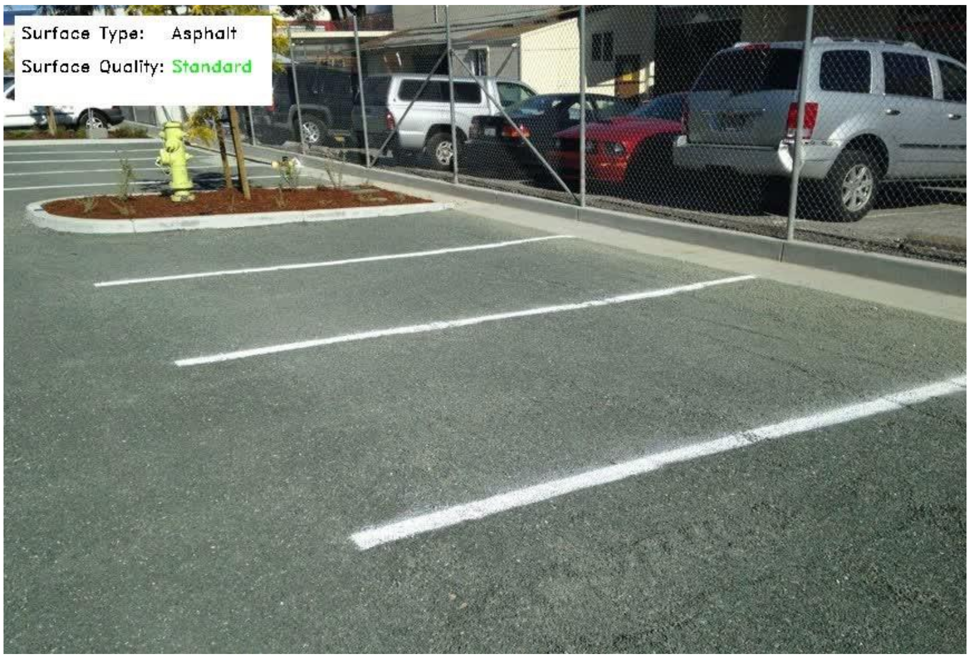


FIGURE 31 OUTPUT (ASPHALT - STANDARD)

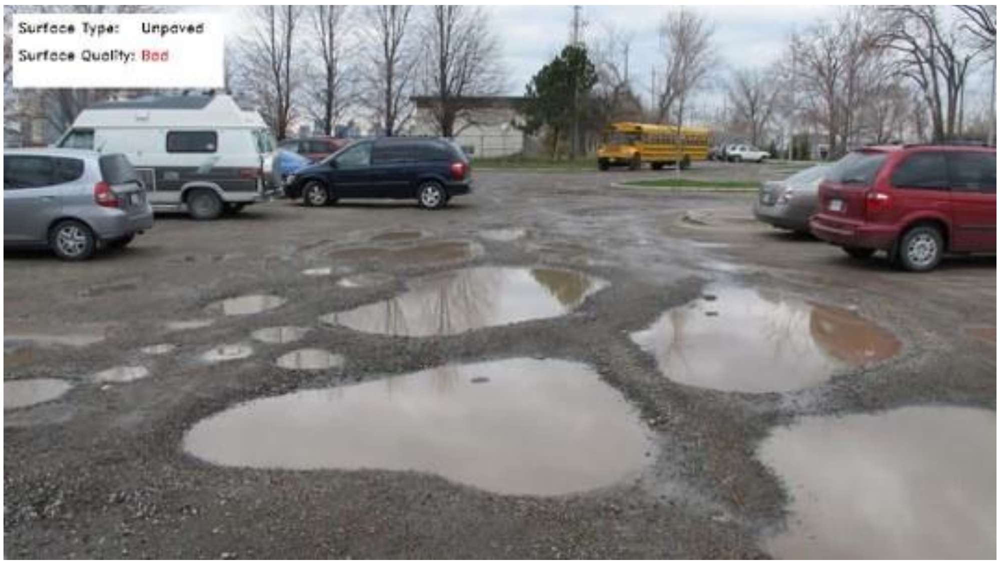


FIGURE 32 OUTPUT (UNPAVED- BAD)

5.2 Research Findings

My research based on the classification of the parking surface which used to identify the parking area standard before registering yard to the system. As described in the literature survey, I have researched on different techniques used by different authors. In those different scenarios they have come up with different approaches. We needed to cover up those all scenarios such as surface identification, crack detection, potholes detection and many more related standards of a parking yard. After researching those I have identified Convolutional Neural Network (CNN) structure as the most suitable approach for this kind of scenario which we can used to classify bunch of constraint at the same time (Figure 33).

Using those four models' methodology it was easy to accomplish the requirement on the "Pay as you park" owner registration without having troubles of virtuality of the parking application even with the current situation.

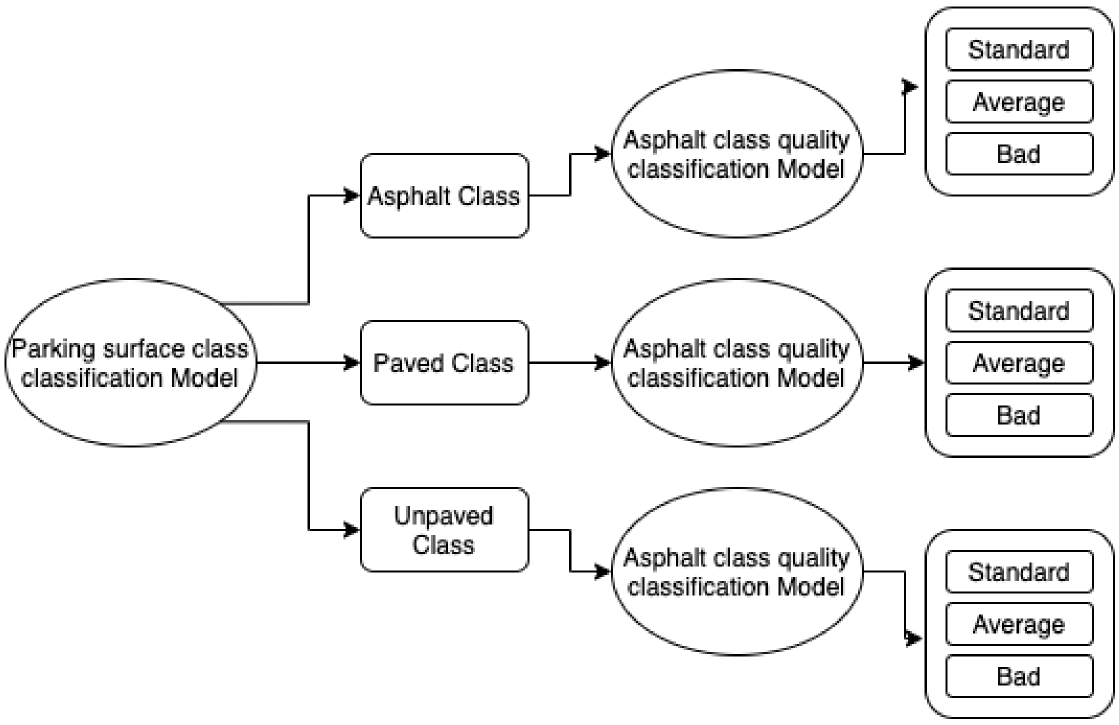

Parking surface class classification Model
Asphalt Class
Paved Class
Unpaved Class
Asphalt class quality classification Model
Asphalt class quality classification Model
Asphalt class quality classification Model
Standard
Average
Bad
Standard
Average
Bad
Standard
Average
Bad

FIGURE 33 MODEL TREE

5.3 Discussion

In this research my primary goal was to provide a method to register the new parking yards with a proper verification procedure for the users who have their own land and garages. There are many factors to consider when registering a new place as a parking yard to systematically.

To coverup all those factors I come up with solution which is very fast and smooth process with caring the concerns of the yard owners and the drivers. Also, I was focused on the real time web application development which provide the end users (Parking yard owner) a better UI and UX with considering mobile responsive web development. And to evaluate the key factors impact on parking I have used haversine and CNN model as it gives the accurate result with covering all the constraint of parking yard standard which we should consider when registering the yard to our new system. We focused on using latest tech stack evolving in the industry which would give advantage to the application level. When it comes to parking yard Standarization and registration, I got attention on what the most users want and what are the problems they are faced I their parking vehicle and the renting their places as a commercial purpose. As per the response of users gained by the users, it clearly shows that parking yards are not well organized and parking unavailability of the urban areas causes for many reasons. So, I focused more on the problems and fact cause the problems and implemented new functionality to add owners to their places for ranting purposes and standardize those all places with less human interaction in real time using Convolutional Neural Network (CNN) to perform the surface types classification between: asphalt, paved and unpaved algorithm. Use of flask framework I created the model API to retrieve the Realtime image classification result for the parking yard quality evaluation. Also, we choose the node.js as server application where we gained the endpoints to our web application. Node.js API application implemented in Express.js framework which connect with Mongo DB and perform database operations. Mongo DB plays key role in this application since it provides high availability, accuracy and efficiency. Through the solution that I have implemented for parking yard registration and validation process; we can use this component in final product to provide users a better parking experience which may result in positive way to environment and society.

6 SUMMARY OF THE STUDENT CONTRIBUTION

Software Implementation and Testing

- Identifying the major problems in parking vehicles

At the beginning of the research, I have conducted a survey to find out the major problems of the urban parking yards. After the survey I have identified there is lack of urban areas. And also, we research about some existing parking applications features, but none of them focusing to increase parking places in urbans areas. So, I come up with new idea of registering new private parking places and land to the system to overcome unavailability of parking places urban areas. To accomplish this feature in our system we need to verify those places before releasing to the system for commercial use.

- Identifying best algorithms and techniques to use to measure the standards

When considering the procedure of parking yard registration and validation, I had to find a method to cover up the registration process by validating the parking place in optimal method with satisfaction of the end user by ending the process as fast as possible. Identifying the parking place quality assurance factors and using a machine learning methodology to whether checkup those factors available or not in those registering new lands and granges was a challenge throughout the research. Anyway, as the given above results chosed algorithm and mythology gave us the satisfied results compared to other methods suggested by other papers.

- UX/UI designing in “Pay as you park” web application

At the final stage of the research, I had to find a best way to design the parking registration process to complete the whole process in ease way to the end user. Using the React I have done the attractive move to our web application by providing fully configured parking yard management to the yard owner. Comparing to other application this web application will be more useful because none of the application provide a yard management options in application itself.

7 CONCLUSION

Parking in urban areas is more difficult for the divers in current situation. People are looking for smart solution with current trends to overcome such situations. “Pay as you park” allows the user to grab the best opportunity with many trends features. Divers and parking yards owners can use this application to have a great experience of smart parking solution by earning some incomes and rewards. Features like registering peoples’ own lands and garages except existing parking yards will expand the business case and availability of the current services to end users. To fast forward this process with expanding the market with highest growth, it is mandatory to provide a reliable service continuously by winning the trust of users. In this content describe the process of standardizing the parking places before registering to the system.

Using the latest trending technology, we have conducted this research to overcome our end user problems by giving solutions. The ability to train a model to conduct image classification, automating the parking surface type identification task, and also the quality of that surface has become more practical and faster due to deep learning developments in recent years. In this study, we used a dataset of surface images with various sorts of surfaces, both damaged and undamaged. In addition, the photos from the collection were found on the internet as a free open-source dataset. In this paper, we demonstrated our research using a basic Convolutional Neural Network (CNN) to differentiate between asphalt, paved, and unpaved surfaces. We got good results from a significant quantity of data in a variety of datasets. Also, we have created a structure to classify the surface quality based on the surface type using another three models. After having the good result for the first surface type classification model we used the same methodology to build other quality models. We have obtained good results as well as the first model we build use of images and deep learning.

8 REFERENCES

[1]. *Lin, Trista & Rivano, Herve & Le Mouël, Frédéric. (2017). A Survey of Smart Parking Solutions. IEEE Transactions on Intelligent Transportation Systems. 18. 3229 - 3253. 10.1109/TITS.2017.2685143.*

[2]. *Awari, Mahesh babu & Babu, Mahesh. (2017). Study of Urban Cities Traffic Problems Due to Delay and Overcrowding. International Journal of Latest Engineering and Management Research (IJLEMR). 03. 1-8.*

[3]. *Ibrahim, Hossam El-Din, Car Parking Problem in Urban Areas, Causes and Solutions (November 25, 2017). 1st International Conference on Towards a Better Quality of Life, 2017, Available at SSRN: https://ssrn.com/abstract=3163473 or http://dx.doi.org/10.2139/ssrn.3163473*

[4]. *M. M. Forrest, Z. Chen, S. Hassan, I. O. Raymond and K. Alinani, "Cost Effective Surface Disruption Detection System for Paved and Unpaved Roads," in IEEE Access, vol. 6, pp. 48634-48644, 2018, doi: 10.1109/ACCESS.2018.2867207.*

[5]. *Nienaber, S & Booysen, M.J. (Thinus) & Kroon, Rs. (2015). Detecting Potholes Using Simple Image Processing Techniques and Real-World Footage. 10.13140/RG.2.1.3121.8408.*

[6]. *Taluja, Chandan & Thakur, Ritula. (2018). An Intelligent Model for Indian Soil Classification using various Machine Learning Techniques. 2250-3005.*

[7]. *Lee, T.; Yoon, Y.; Chun, C.; Ryu, S. CNN-Based Road-Surface Crack Detection Model That Responds to Brightness Changes. Electronics2021, 10, 1402. https://doi.org/10.3390/electronics10121402*

[8]. *Varona, B., Monteserin, A., & Teyseyre, A. (2019). A deep learning approach to automatic road surface monitoring and pothole detection. Personal and Ubiquitous Computing. doi:10.1007/s00779-019-01234-z*

[9]. *Nienaber, S & Booysen, M.J. (Thinus) & Kroon, Rs. (2015). Detecting Potholes Using Simple Image Processing Techniques and Real-World Footage. 10.13140/RG.2.1.3121.8408.*

[10]. *Taluja, Chandan & Thakur, Ritula. (2018). An Intelligent Model for Indian Soil Classification using various Machine Learning Techniques. 2250-3005.*

[11]. *Mingxing Gao, Xu Wang, Shoulin Zhu, Peng Guan, "Detection and Segmentation of Cement Concrete Pavement Pothole Based on Image Processing Technology", Mathematical Problems in Engineering, vol. 2020, Article ID 1360832, 13 pages, 2020. https://doi.org/10.1155/2020/1360832*

[12]. *Mahmoodi-Eshkaftaki, M., Haghighi, A., & Houshyar, E. (2019). Land Suitability Evaluation using Image Processing based on Determination of Soil Texture-Structure and Soil Features. Soil Use and Management. doi:10.1111/sum.12572*

[13]. *Rouini, Abdelghani. (2020). Re: Is there any dataset of potholes on roads available?. Retrieved from: https://www.researchgate.net/post/Is_there_any_dataset_of_potholes_on_roads_available/5f5be694532e3871c05d4e96/citation/download.*

[14]. *Crum, J. R. and H. P. Collins. 1995. KBS Soils. Kellogg Biological Station Long-term Ecological Research Special Publication. Zenodo.*

[15]. *Dai, Jinpeng (2020), "Cement and Concrete", Mendeley Data, V1, doi: 10.17632/m96hc626hg.1*