Garbage Classification Using Deep Learning

Mohammed Aaththish M

Guide: Mrs. K. Priyadharshini , Assistant Professor

Department of Computer Science and Engineering

Master of Engineering

Sethu Institute of Technology

Pulloor, Kariapatti-626 115

ABSTRACT

Garbage disposal is a critical responsibility in maintaining a healthy and green environment. The recycling business is thriving as inhabitants of India become more aware of the necessity of a clean environment in terms of reducing natural resource use and rubbish disposal. The amount of trash generated in daily life has an impact on land, water, and air, posing a major threat to aquatic animals and their habitats, as well as humans, if not properly managed. Conventional garbage disposal systems are on the rise, requiring accurate and efficient segmentation and recognition processes. This demand corresponds to an increase in the computing capacity of modern computer systems as well as more efficient picture recognition methods. To address this issue, the garbage categorization process is automated by using an image classifier with a convolutional neural network (CNN), which reduces waste segregation time and costs. The goal of automating the process is to reduce human intervention and increase productivity in the waste segregation process. In this paper, three distinct models are tested for greater accuracy: Simple CNN, ResNet50, and VGG16 are trained on various image datasets and used to extract features from images, which are then fed into a classifier for dump/trash categorization. The learned models are then loaded into the mobile application.

CHAPTER 1
INTRODUCTION

Object detection is an important and challenging field in computer vision, one which has been the subject of extensive research [1]. The goal of object detection is to detect all objects and class the objects. It has been widely used in autonomous driving [2], pedestrian detection [3], medical imaging [4], industrial detection [5], robot vision [6], intelligent video surveillance [7], remote sensing images [8], etc. In recent years, deep learning techniques have been applied in object detection [9]. Deep learning uses low-level features to form more abstractive high-level features, and to hierarchically represent the data in order to improve object detection [10]. Compared with traditional detection algorithms, the deep learning-based object detection method based has better performance in terms of robustness, accuracy and speed for multi-classification tasks. Object detection methods based on deep learning mainly include region proposal-based methods, and those based on a unified pipeline framework. The former type of method firstly generates a series of region proposals from an input image, and then uses a convolution neural network to extract features from the generated regions and construct a classifier for object classes. The region-based convolution neural network (R-CNN) method [11] is the earlier method used to introduce convolution neural networks into the field of object detection. It uses the selection search method to generate region proposals from the input images, and uses a convolution neural network to extract features from the generated region proposals. The extracted features are used to train the support vector machine. Based on the R-CNN method, Fast R-CNN [12] and Faster R-CNN [13] were also proposed to reduce training time and improve the mean average precision. Although region proposal-based methods have higher detection accuracy, the structure of the method is complex, and object detection is time-consuming. The latter type of method (based on a unified pipeline framework) directly predicts location information and class probabilities of objects with a single-feed forward convolution neural network from the whole image, and does not require the generation of region proposals and post-classification. Therefore, the structure of the unified pipeline framework approach is simple and can detect objects quickly; however, it is less accurate than the region proposal-based approach. The two kinds of methods have different advantages and are suitable for different applications. In this paper, we mainly discuss the unified pipeline framework-based approach. Researchers have proposed a wide range of unified pipeline framework-based methods in recent years, one of which is the You Only Look Once v2 (YOLOv2) method [14]. YOLOv2 uses batch normalization to improve convergence and prevent overfitting, and anchor boxes to predict bounding boxes, in order to increase the recall. Other innovations include a high-resolution classifier, direct location prediction, dimension cluster, and multi-scale training, all of which lend greater detection accuracy. Pedoeem and Huang recently proposed a shallow real-time detection method for Non-GPU computers based on the YOLOv2 method [15]; their method reduces the size of input image by half in order to speed up the detection speed, and removes the batch normalization of shallow layers in order to reduce the amount of model parameters. Shafiee et al. have proposed the Fast YOLO method, whereby YOLOv2 can be applied to embedded devices [16]: this employs an evolutionary deep intelligence framework to generate an optimized network architecture. The optimized network architecture can be used in the motion-adaption inference framework to speed up the detection process and thus reduce the energy consumption of the embedded device. Simon et al. have developed a complex-YOLO method [17], which uses a specific complex regression strategy to estimate multi-class 3D boxes in Cartesian space for detecting RGB images; the authors report a significant improvement in the speed of 3D object detection. Liu et al. have developed the Single Shot MultiBox Detector (SSD) method [18], which generates multi-scale feature maps in order to detect objects of different sizes. This method strikes a careful balance between speed and accuracy of detection, but the expression ability of the feature map is insufficient in the shallow layer. In order to enhance the expression ability of shallow feature maps, Fu et al. have proposed the De-convolutional Single Shot Detector (DSSD) method [19], which uses the ResNet extraction network (generating better features) [20], a deconvolution layer and skip connection in order to improve the expression ability of shallow feature maps. In order to improve the detection accuracy of the SSD method for small objects, Qin et al. have proposed a new SSD method based on the feature pyramid [21]. Their method applies a deconvolution network in the high-level of the feature pyramid in order to extract semantic information, and expands the convolution network so as to learn low-level position information. Their method constructs a multi-scale detection structure so as to improve the detection accuracy of small objects. Redmon and Farhadi have proposed applying the YOLOv3 method for using binary cross-entropy loss for class predictions [22], which employs scale prediction to predict boxes at different scales, and thus improves the detection accuracy with regard to small objects

CHAPTER 2
LITERATURE SURVEY

2.1 YOLO v3-Tiny: Object Detection and Recognition using one stage improved model, 2020

—Object detection has seen many changes in algorithms to improve performance both on speed and accuracy. By the continuous effort of so many researchers, deep learning algorithms are growing rapidly with an improved object detection performance. Various popular applications like pedestrian detection, medical imaging, robotics, self-driving cars, face detection, etc. reduces the efforts of humans in many areas. Due to the vast field and various state-of-the-art algorithms, it is a tedious task to cover all at once. This paper presents the fundamental overview of object detection methods by including two classes of object detectors. In two stage detector covered algorithms are RCNN, Fast RCNN, and Faster RCNN, whereas in one stage detector YOLO v1, v2, v3, and SSD are covered. Two stage detectors focus more on accuracy, whereas the primary concern of one stage detectors is speed. We will explain an improved YOLO version called YOLO v3-Tiny, and then its comparison with previous methods for detection and recognition of object is described graphically

ADVANTAGE:

- RCNN, Fast RCNN, and Faster RCNN,
- comparison with previous methods for detection and recognition of object

2.2 Automated Image Capturing System for Deep Learning-based Tomato Plant Leaf Disease Detection and Recognition, 2018

Smart farming system using necessary infrastructure is an innovative technology that helps improve the quality and quantity of agricultural production in the country including tomato. Since tomato plant farming take considerations from various variables such as environment, soil, and amount of sunlight, existence of diseases cannot be avoided. The recent advances in computer vision made possible by deep learning has paved the way for camera-assisted disease diagnosis for tomato. This study developed the innovative solution that provides efficient disease detection in tomato plants. A motor-controlled image capturing box was made to capture four sides of every tomato plant to detect and recognize leaf diseases. A specific breed of tomato which is Diamante Max was used as the test subject. The system was designed to identify the diseases namely Phoma Rot, Leaf Miner, and Target Spot. Using dataset of 4,923 images of diseased and healthy tomato plant leaves collected under controlled conditions, we train a deep convolutional neural network to identify three diseases or absence thereof. The system used Convolutional Neural Network to identify which of the tomato diseases is present on the monitored tomato plants. The F-RCNN trained anomaly detection model produced a confidence score of 80 % while the Transfer Learning disease recognition model achieves an accuracy of 95.75 %. The automated image capturing system was implemented in actual and registered a 91.67 % accuracy in the recognition of the tomato plant leaf diseases

ADVANTAGES:

- It has more efficiency in term of embedding capacity

DISADVANTAGES:

- One of the problems is that the low Accuracy

2.3 Design and Implementation of High Speed Background Subtraction Algorithm for Moving Object Detection, 2018

Object detection is important and challenging task in computer vision applications such as surveillance, vehicle navigation, and human tracking. Video surveillance is a key technology to fight against terrorism and public safety management. In video surveillance, detection of moving objects from a video is important for object detection and behaviour understanding. Detection of moving objects in video streams is important process of revelation and background subtraction is popular approach for foreground segmentation. In this paper high speed background subtraction algorithm for moving object detection is proposed. The video is first converted to streams and then applied to convolution filter which removes high frequency noise components to obtain smoothened images. The smoothened images are then applied to background subtraction algorithm with adaptive threshold which gives detected object present in background image. The detected object is then applied to convolution filter to remove the spurious distorted pixels which improves the quality of image. The proposed architecture was designed using VHDL language and implemented using Spartan-6 (XC6SLX45-2csg324) FPGA kit. It is observed that the proposed technique is better compared to existing method in terms of image quality and speed of operations.

ADVANTAGES:

- It requires Low Cost
- It is Low Object detection Time

DISADVANTAGES

- It has low prediction quality.

2.4 Multi-View 3D Object Detection Network for Autonomous Driving

This paper aims at high-accuracy 3D object detection in autonomous driving scenario. We propose Multi-View 3D networks (MV3D), a sensory-fusion framework that takes both LIDAR point cloud and RGB images as input and predicts oriented 3D bounding boxes. We encode the sparse 3D point cloud with a compact multi-view representation. The network is composed of two subnetworks: one for 3D object proposal generation and another for multi-view feature fusion. The proposal network generates 3D candidate boxes efficiently from the bird's eye view representation of 3D point cloud. We design a deep fusion scheme to combine region-wise features from multiple views and enable interactions between intermediate layers of different paths. Experiments on the challenging KITTI benchmark show that our approach outperforms the state-of-the-art by around 25% and 30% AP on the tasks of 3D localization and 3D detection. In addition, for 2D detection, our approach obtains 14.9% higher AP than the state-of-the-art on the hard data among the LIDAR-based methods.

DISADVANTAGES:

- Object detection Time is high.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

It uses the k-means cluster method to estimate the initial width and height of the predicted bounding boxes. With this method, the estimated width and height are sensitive to the initial cluster centres, and the processing of large-scale datasets is time-consuming. In order to address these problems, a new cluster method for estimating the initial width and height of the predicted bounding boxes has been developed. Firstly, it randomly selects a couple of width and height values as one initial cluster centre separate from the width and height of the ground truth boxes. Secondly, it constructs Markov chains based on the selected initial cluster and uses the final points of every Markov chain as the other initial centres. In the construction of Markov chains, the intersection-over-union method is used to compute the distance between the selected initial clusters and each candidate point, instead of the square root method. Finally, this method can be used to continually update the cluster centre with each new set of width and height values, which are only a part of the data selected from the datasets.

DISADVANTAGES:

- Object Detection Time is High

3.2 PROPOSED SYSTEM:

In proposed system, we have to identify and localize objects, there exist many methods with a trade-off in speed performance and accuracy of result. But yet we can't say any single algorithm is best over others. One can always select the method that suits the requirement at best. In a short period, object detection applications got much popularity and still a lot to cover in this area because of its vast scope of research[15] [18] [26]. This paper presents the comparison of various algorithms to identify and localize objects based on accuracy, time, and parameter values with varying sizes of the input image. We have identified a new methodology of single stage model for improving speed without sacrificing much accuracy. The comparison results show that YOLO v3-Tiny increases the speed of object detection while ensures the accuracy of the result. We can also extend object localization and recognition from static pictures to a video containing the dynamic sequence of images.

ADVANTAGES:

- It performs the good Detection.
- It enhance the performances.

CHAPTER 4
SYSTEM REQUIREMENTS

The system requirement is the first step in the requirements analysis process. It lists the requirements of a particular software system including functional, performance and security requirements. The requirements also provide usage scenarios from a user, an operational and an administrative perspective. The purpose of software requirements specification is to provide a detailed overview of the software project, its parameters and goals. This describes the project target audience and its user interface, hardware and software requirements. It defines how the client, team and audience see the project and its functionality.

4.1 HARDWARE SPECIFICATION:

- System : Pentium IV 2.4 GHz
- Hard Disk : 200 GB
- Mouse : Logitech.
- Keyboard : 110 keys enhanced
- Ram : 4GB
-

4.2 SOFTWARE SPECIFICATION:

- O/S : Windows 7.
- Language : Python
- Front End : Anaconda Navigator - Spyder

4.3 SOFTWARE DESCRIPTION

4.3.1 Python

Python is one of those rare languages which can claim to be both *simple* and powerful. You will find yourself pleasantly surprised to see how easy it is to concentrate on the solution to the problem rather than the syntax and structure of the language you are programming in. The official introduction to Python is Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. I will discuss most of these features in more detail in the next section.

4.3.2 Features of Python

- **Simple**

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English, although very strict English! This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the language itself.

- **Easy to Learn**

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax, as already mentioned.

- **Free and Open Source**

Python is an example of a *FLOSS* (Free/Libré and Open Source Software). In simple terms, you can freely distribute copies of this software, read its source code, make changes to it, and use pieces of it in new free programs. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and is constantly improved by a community who just want to see a better Python.

- **High-level Language**

When you write programs in Python, you never need to bother about the low-level details such as managing the memory used by your program, etc.

- **Portable**

Due to its open-source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features.

You can use Python on GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC!

You can even use a platform like Kivy to create games for your computer *and* for iPhone, iPad, and Android.

- **Interpreted**

This requires a bit of explanation.

A program written in a compiled language like C or C++ is converted from the source language i.e. C or C++ into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it.

Python, on the other hand, does not need compilation to binary. You just *run* the program directly from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your computer and then runs it. All this, actually, makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc. This also makes your Python programs much more portable, since you can just copy your Python program onto another computer and it just works!

- **Object Oriented**

Python supports procedure-oriented programming as well as object-oriented programming. In *procedure-oriented* languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In *object-oriented* languages, the program is built around objects which combine data and functionality. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

- **Extensible**

If you need a critical piece of code to run very fast or want to have some piece of algorithm not to be open, you can code that part of your program in C or C++ and then use it from your Python program.

- **Embeddable**

You can embed Python within your C/C++ programs to give *scripting* capabilities for your program's users.

- **Extensive Libraries**

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), and other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the *Batteries Included* philosophy of Python.

Besides the standard library, there are various other high-quality libraries which you can find at the Python Package Index.

CHAPTER 5
SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE:

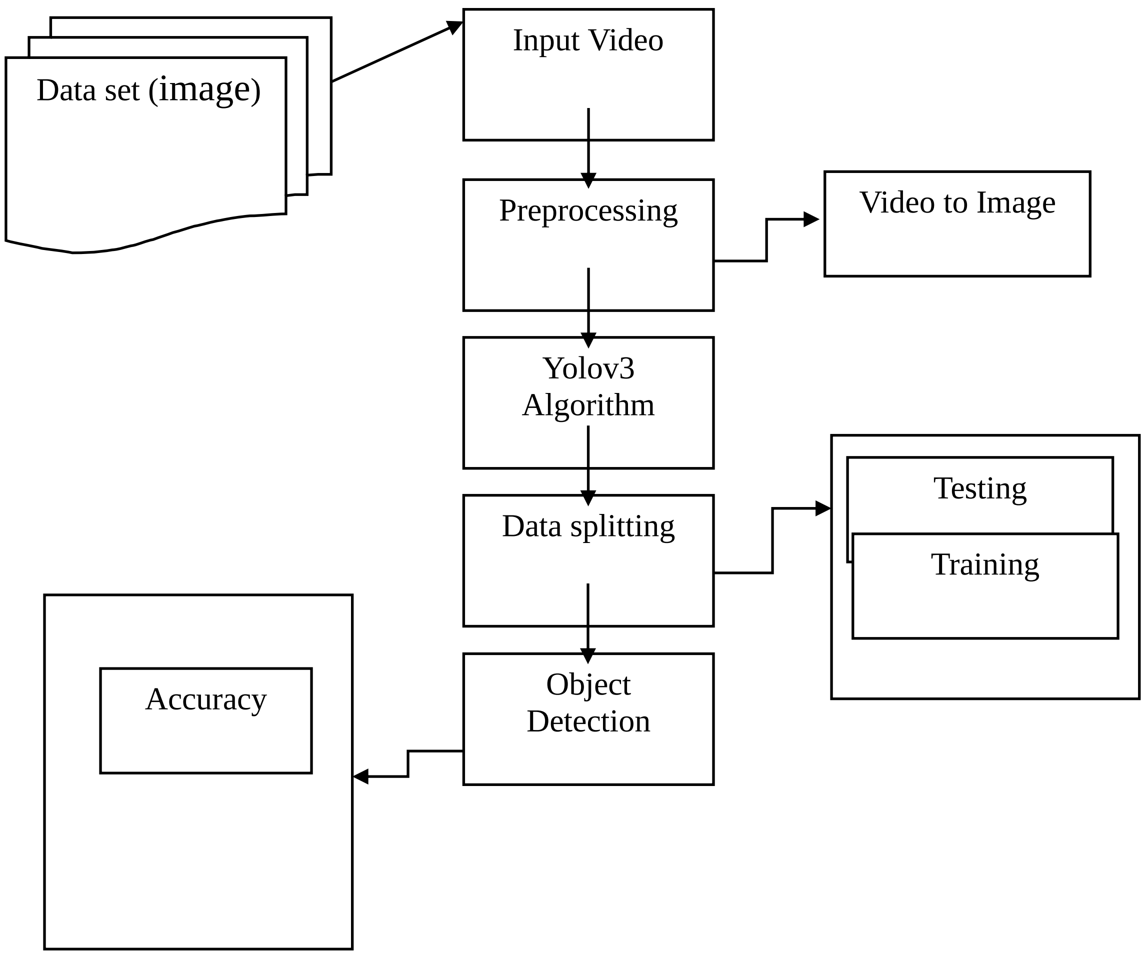

Figure 5.1: System Architect

5.2 FLOW DIAGRAM:

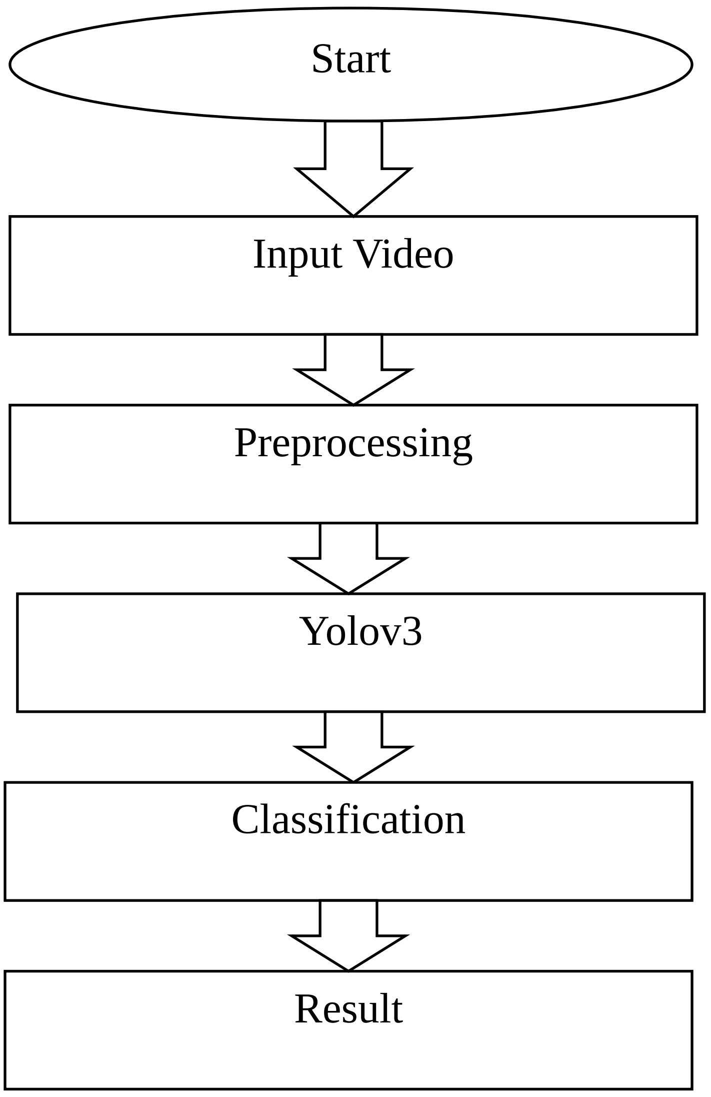

Figure 5.2: Flow Diagram

5.3 USE CASE DIAGRAM:

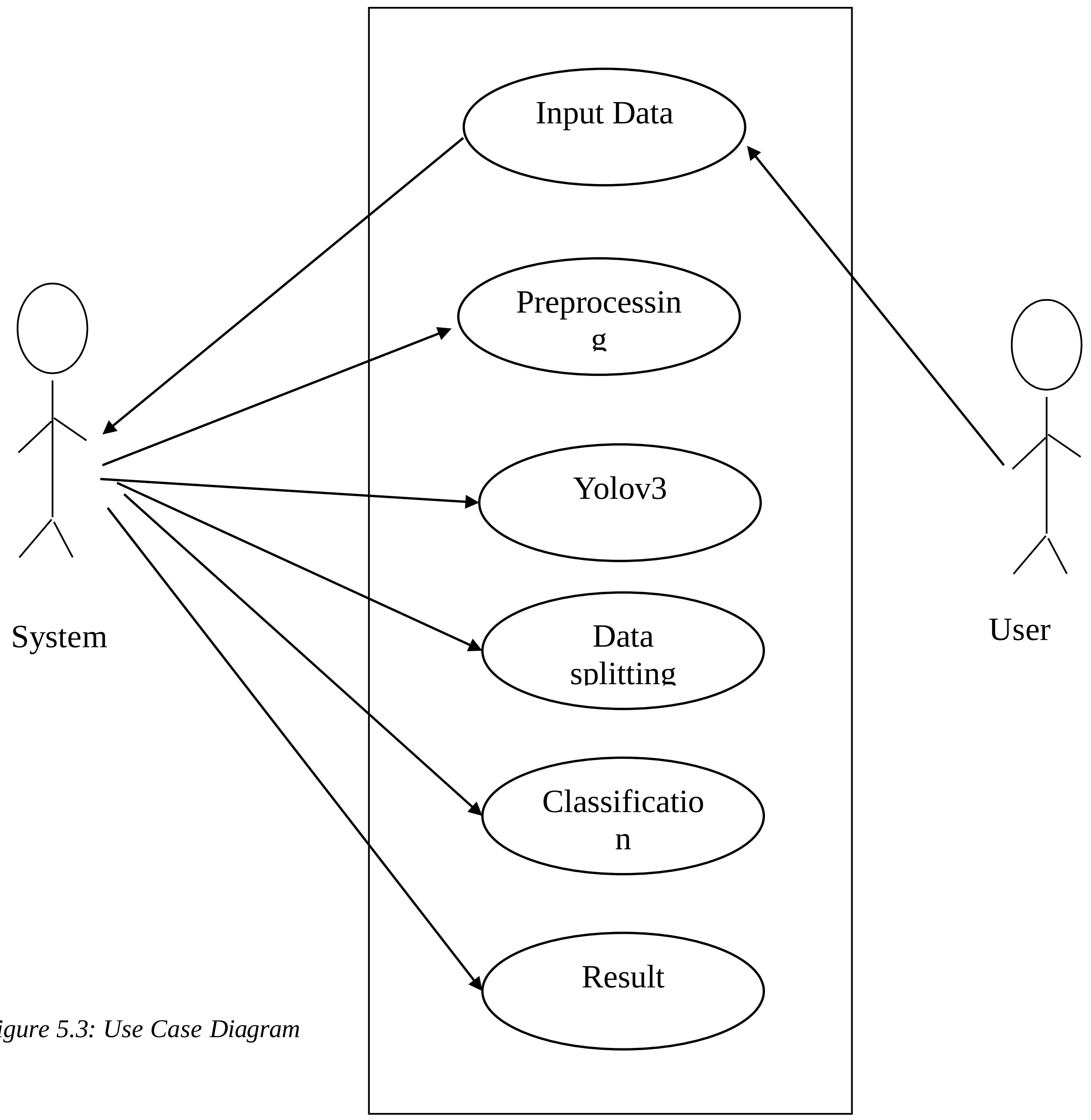

Figure 5.3: Use Case Diagram

5.4 ACTIVITY DIAGRAM:

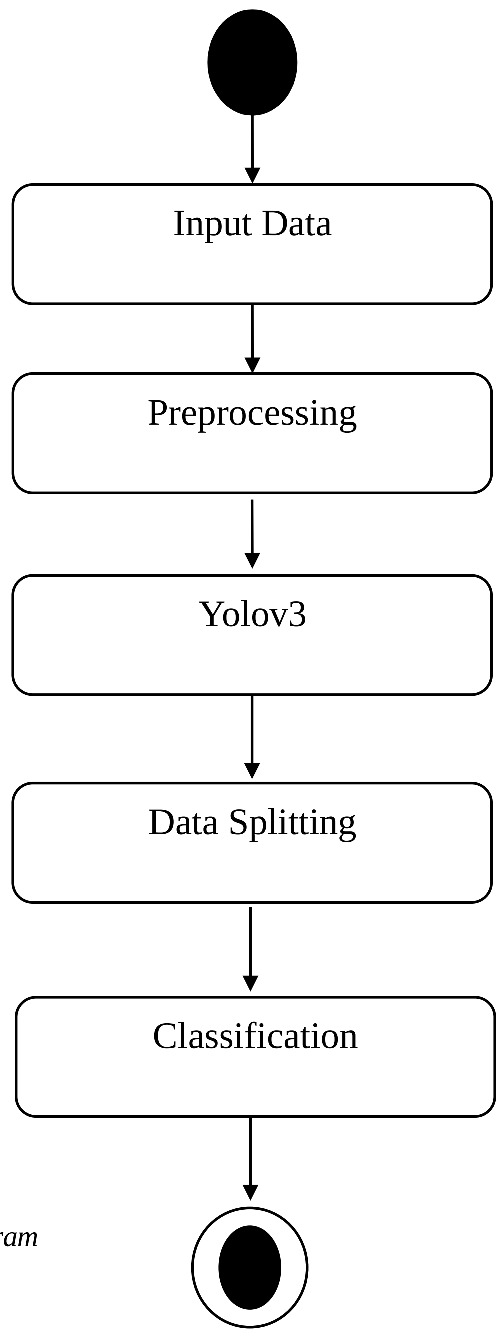

Figure 5.4: Activity Diagram

5.5 CLASS DIAGRAM:

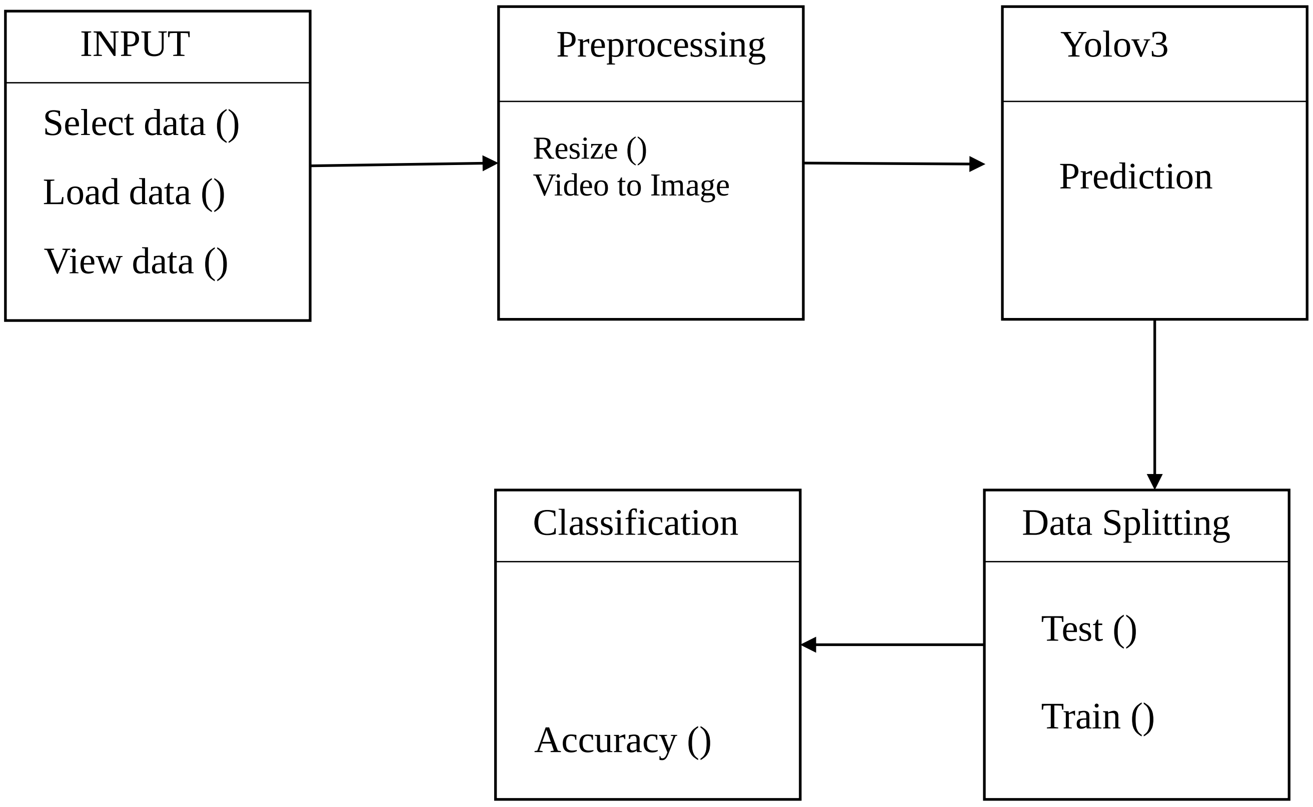

Figure 5.5: Class Diagram

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 04 | April - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

5.6 SEQUENCE DIAGRAM:

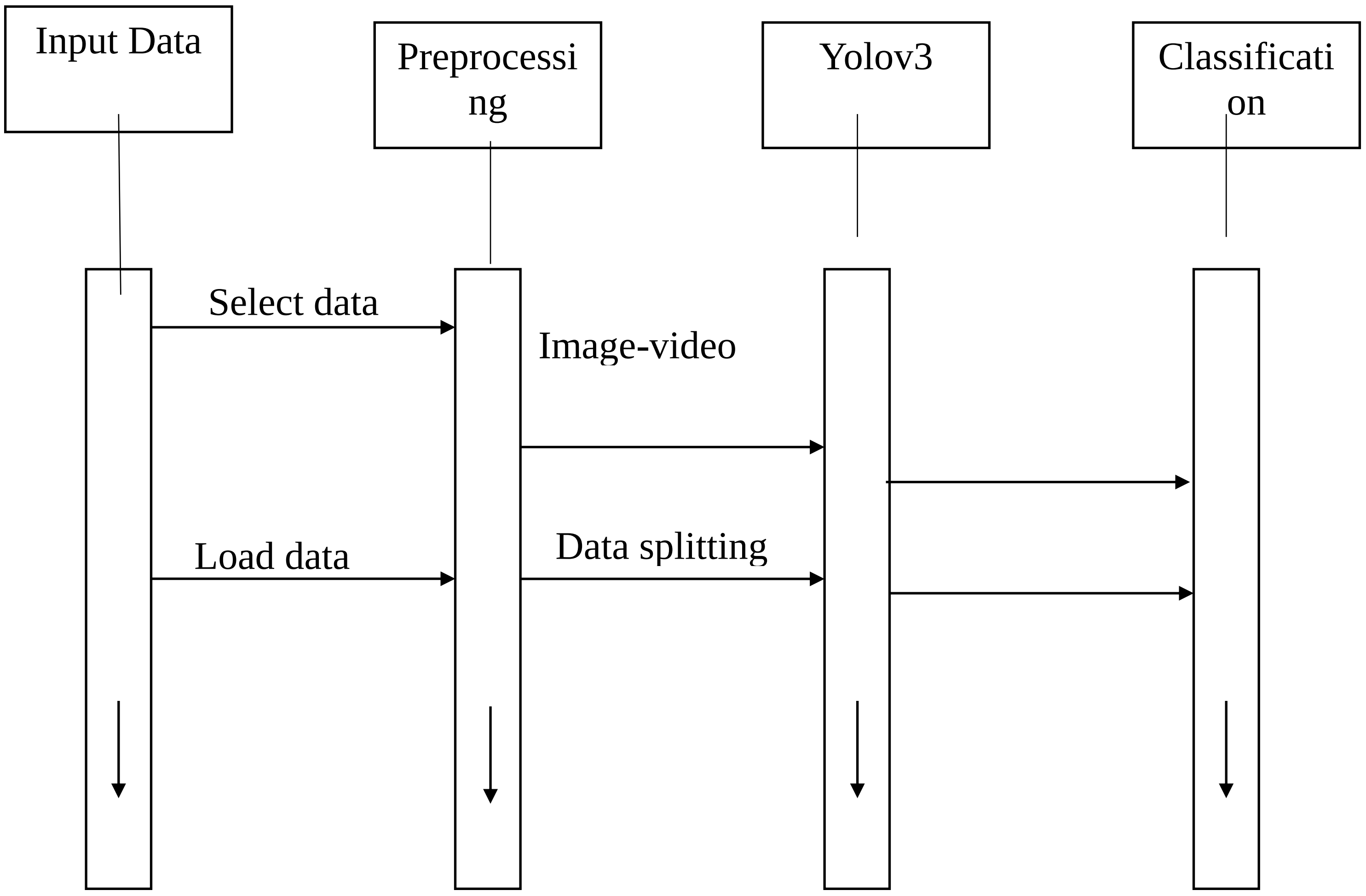

Figure 5.6: Sequence Diagram

[Type here]

5.7 ER DIAGRAM:

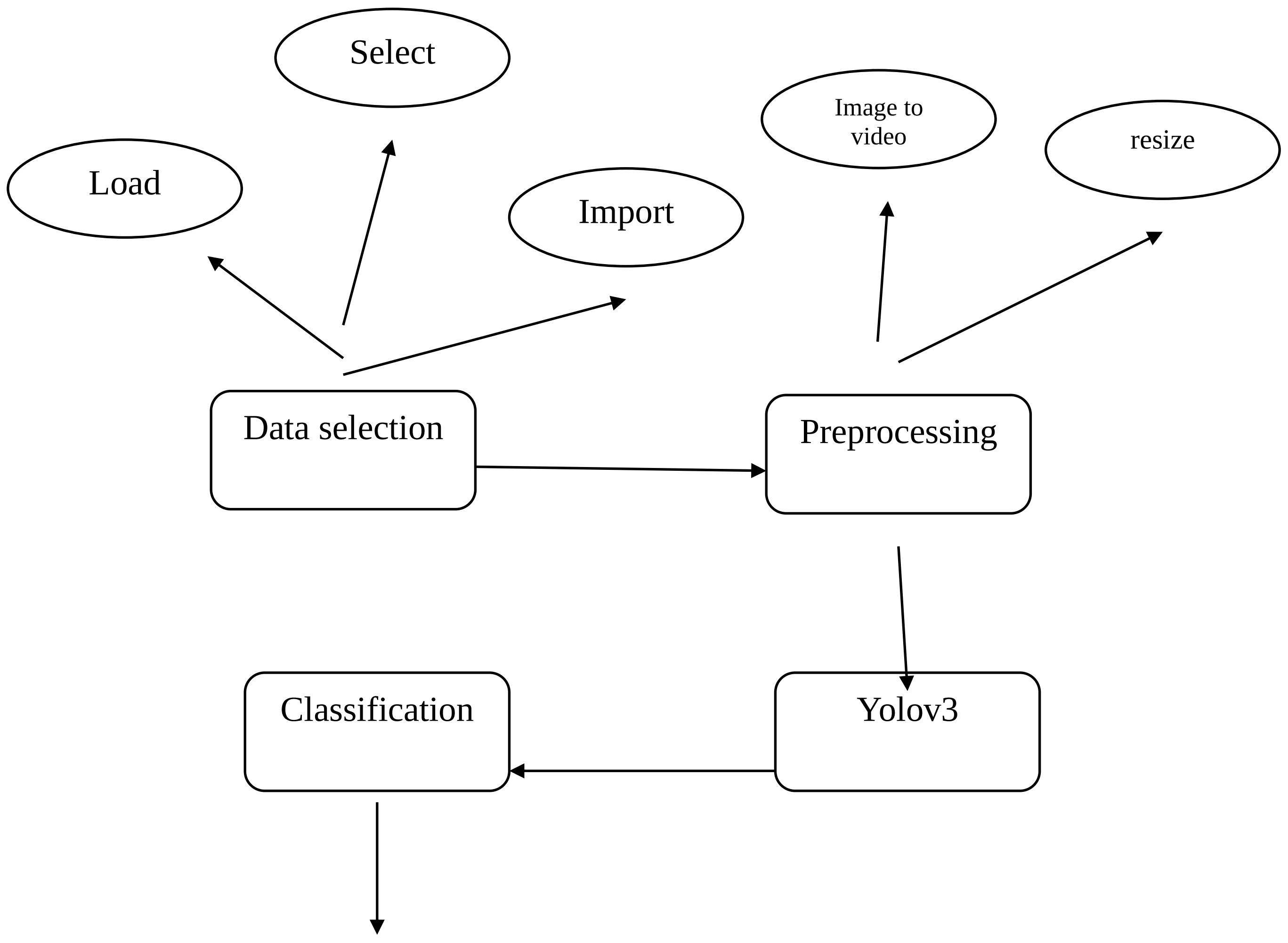

Figure 5.7: ER Diagram

CHAPTER 6
SYSTEM IMPLEMENTATION

6.1 MODULES:

- Data selection
- Data preprocessing
- Data splitting
- Yolov3
- Deep learning
- Result Generation

6.1 MODULE DESCRIPTION

6.2.1 DATA SELECTION:

- The data selection is the process of selecting and loading the input images from dataset.
- The dataset is used to video

6.2.2 PREPROCESSING:

- The data selection is the process of selecting and loading the input images from dataset.
- The dataset is used to video convert to image
- To read the image with the help of imread() function.

6.2.3 Yolov3:

- Object detection is used in many fields of human life, for example, health and education, etc. As all these fields are growing rapidly, so to match their requirements, one stage models also need improvement. The next advanced variant of YOLO is version 3 that uses logistic regression to compute the targetness score. It gives the score for all targets in each boundary box.
- YOLO v3 can give the multilabel classification because it uses a logistic classifier for each class in place of the softmax layer used in YOLO v2. YOLO v3 uses darknet 53. It has fifty-three layers of convolution. These layers are more in-depth compared to darknet 19 used in YOLO v2. Darknet-53 contains mainly 3x3 and 1x1 filters along with bypass links [8] [9] [10]. The formulas given below explain the transformation of network output for obtaining bounding box prediction

6.2.4 DATA SPLITTING:

- Data splitting is the act of partitioning available data into two portions, usually for cross-validator purposes.
- One Portion of the data is used to develop a predictive model and the other to evaluate the model's performance.
- Separating data into training and testing sets is an important part of evaluating data mining models.
- Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing.

6.2.5 DEEP LEARNING:

- In classification process, we are using ANN algorithm for better performance.
- An is configured for a specific application, such as pattern recognition or data classification, through a learning process.
- Learning largely involves adjustments to the synaptic connections that exist between the neurons.

6.2.6 RESULT GENERATION:

- While using collaborative filtering algorithm, the recommend Movie for particular user id.
- The Final Result will get generated based on the overall classification and prediction.
- The performance of this proposed approach is evaluated using some measures like,
- **Accuracy**

Accuracy of classifier refers to the ability of classifier. It predicts the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.

CHAPTER 7
SYSTEM TESTING

System testing is the stage of implementation, which aimed at ensuring that system works accurately and efficiently before the live operation commence. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an error. A successful test is one that answers a yet undiscovered error. Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. . A series of tests are performed before the system is ready for the user acceptance testing. Any engineered product can be tested in one of the following ways. Knowing the specified function that a product has been designed to from, test can be conducted to demonstrate each function is fully operational. Knowing the internal working of a product, tests can be conducted to ensure that “al gears mesh”, that is the internal operation of the product performs according to the specification and all internal components have been adequately exercised.

7.1 UNIT TESTING:

Unit testing is the testing of each module and the integration of the overall system is done. Unit testing becomes verification efforts on the smallest unit of software design in the module. This is also known as ‘module testing’. The modules of the system are tested separately. This testing is carried out during the programming itself. In this testing step, each model is found to be working satisfactorily as regard to the expected output from the module. There are some validation checks for the fields. For example, the validation check is done for verifying the data given by the user where both format and validity of the data entered is included. It is very easy to find error and debug the system.

7.2 INTEGRATION TESTING:

Data can be lost across an interface, one module can have an adverse effect on the other sub function, when combined, may not produce the desired major function. Integrated testing is systematic testing that can be done with sample data. The need for the integrated test is to find the overall system performance. There are two types of integration testing. They are:

i) Top-down integration testing. ii) Bottom-up
integration testing.

7.3 TESTING TECHNIQUES/STRATEGIES:

7.3.1 WHITE BOX TESTING:

White Box testing is a test case design method that uses the control structure of the procedural design to drive cases. Using the white box testing methods, we Derived test cases that guarantee that all independent paths within a module have been exercised at least once.

7.3.2 BLACK BOX TESTING:

- Black box testing is done to find incorrect or missing function
- Interface error
- Errors in external database access
- Performance errors.
- Initialization and termination errors

In 'functional testing', is performed to validate an application conforms to its specifications of correctly performs all its required functions. So this testing is also called 'black box testing'. It tests the external behaviour of the system. Here the engineered product can be tested knowing the specified function that a product has been designed to perform, tests can be conducted to demonstrate that each function is fully operational.

7.4 SOFTWARE TESTING STRATEGIES

7.4.1 VALIDATION TESTING:

After the culmination of black box testing, software is completed assembly as a package, interfacing errors have been uncovered and corrected and final series of software validation tests begin validation testing can be defined as many, but a single definition is that validation succeeds when the software functions in a manner that can be reasonably expected by the customer

7.4.2 USER ACCEPTANCE TESTING:

User acceptance of the system is the key factor for the success of the system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system at the time of developing changes whenever required.

7.5 OUTPUT TESTING:

After performing the validation testing, the next step is output asking the user about the format required testing of the proposed system, since no system could be useful if it does not produce the required output in the specific format. The output displayed or generated by the system under consideration. Here the output format is considered in two ways. One is screen and the other is printed format. The output format on the screen is found to be correct as the format was designed in the system phase according to the user needs. For the hard copy also output comes out as the specified requirements by the user. Hence the output testing does not result in any connection in the system.

CHAPTER 8
CONCLUSION

To identify and localize objects, there exist many methods with a trade-off in speed performance and accuracy of result. But yet we can't say any single algorithm is best over others. One can always select the method that suits the requirement at best. In a short period, object detection applications got much popularity and still a lot to cover in this area because of its vast scope of research[15] [18] [26]. This paper presents the comparison of various algorithms to identify and localize objects based on accuracy, time, and parameter values with varying sizes of the input image. We have identified a new methodology of single stage model for improving speed without sacrificing much accuracy. The comparison results show that YOLO v3-Tiny increases the speed of object detection while ensures the accuracy of the result. We can also extend object localization and recognition from static pictures to a video containing the dynamic sequence of images.

APPENDIX
SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras import Input
from tensorflow.keras.layers import Dense, Dropout, Conv2D, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.utils import to_categorical
from numpy import argmax
from sklearn.metrics import confusion_matrix, classification_report
from mlxtend.plotting import plot_confusion_matrix
import easygui

#==============================Input Data======================================
path = 'Dataset/'
```

```
# categories
categories = ['cardboard','glass','metal','paper','plastic','trash']

# let's display some of the pictures
for category in categories:
    fig, _ = plt.subplots(3,4)
    fig.suptitle(category)
    fig.patch.set_facecolor('xkcd:white')
    for k, v in enumerate(os.listdir(path+category)[:12]):
        img = plt.imread(path+category+'/'+v)
        plt.subplot(3, 4, k+1)
        plt.axis('off')
        plt.imshow(img)
    #     cv2.imshow(' Image',img)
    #     cv2.waitKey(0)
    #     cv2.destroyAllWindows()
    # plt.show()

shape0 = []
shape1 = []



data = []
labels = []
imagePaths = []
HEIGHT = 65
WIDTH = 65
N_CHANNELS = 3

# grab the image paths and randomly shuffle them
for k, category in enumerate(categories):
    for f in os.listdir(path+category):
        imagePaths.append([path+category+'/'+f, k])

import random
random.shuffle(imagePaths)
print(imagePaths[:10])

# loop over the input images
for imagePath in imagePaths:
# load the image, resize the image to be HEIGHT * WIDTH pixels (ignoring aspect ratio) and store the image in the data list
    image = cv2.imread(imagePath[0])
    image = cv2.resize(image, (WIDTH, HEIGHT))  # .flatten()
    data.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath[1]
    labels.append(label)

# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)

# Let's check everything is ok
fig, _ = plt.subplots(4,5)
fig.suptitle("Sample Input")
fig.patch.set_facecolor('xkcd:white')
for i in range(20):
    plt.subplot(4,5, i+1)
    plt.imshow(data[i])
    plt.axis('off')
#    cv2.imshow(' Image',data[1])
#    cv2.waitKey(0)
#    cv2.destroyAllWindows()
# #    plt.title(categories[labels[i]])
# plt.show()

# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2, random_state=42)
# Preprocess class labels
trainY =to_categorical(trainY, 6)

print(trainX.shape)
print(testX.shape)
print(trainY.shape)
print(testY.shape)

#================================Classification===============================
'''DENSENET121'''

def build_densenet():
    densenet = DenseNet121(weights='imagenet', include_top=False)

    input = Input(shape=(HEIGHT, WIDTH, N_CHANNELS))
    x = Conv2D(3, (3, 3), padding='same')(input)

    x = densenet(x)

    x = GlobalAveragePooling2D()(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

    # multi output
    output = Dense(6,activation = 'softmax', name='root')(x)
    # model
    model = Model(input,output)

    model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
    model.summary()

    return model

model = build_densenet()

model.fit(trainX, trainY, batch_size=32, epochs=10, verbose=1)

history=model.history.history
#Plotting the accuracy
train_loss = history['loss']
train_acc = history['accuracy']
    # Performance graph
plt.figure()
plt.plot(train_loss, label='Loss')
plt.plot(train_acc, label='Accuracy')
plt.title('Performance Plot')
plt.legend()
plt.show()

print("Accuracy of the CNN is:",model.evaluate(trainX,trainY)[1]*100, "%")
#============================Analytic Results================================
pred = model.predict(testX)
predictions = argmax(pred, axis=1)
print('Classification Report')
cr=classification_report(testY, predictions,target_names=categories)
print(cr)
print('Confusion Matrix')
```

```
cm = confusion_matrix(testY, predictions)
print(cm)
#Confusion Matrix Plot
plt.figure()
plot_confusion_matrix(cm,figsize=(15,15), class_names = categories,
                show_normed = True);
plt.title( "Model confusion matrix")
plt.style.use("ggplot")
plt.show()



#================================Prediction=================================
from tkinter import filedialog
test_data=[]
Image = filedialog.askopenfilename()
head_tail = os.path.split(Image)
fileNo=head_tail[1].split('.')
test_image_o = cv2.imread(Image)
test_image = cv2.resize(test_image_o, (WIDTH, HEIGHT))
#test_data.append(test_image)
# scale the raw pixel intensities to the range [0, 1]
test_data = np.array(test_image, dtype="float") / 255.0
test_data=test_data.reshape([-1,65, 65, 3])
pred = model.predict(test_data)
predictions = argmax(pred, axis=1) # return to label
print ('Prediction : '+categories[predictions[0]])
#Imersing into the plot
fig = plt.figure()
fig.patch.set_facecolor('xkcd:white')
plt.title(categories[predictions[0]])
plt.imshow(test_image_o)
#==========================================================================
```

27

REFERENCES

[1].A. Krizhevsky, I. Sutskever, and G. E.Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” NIPS, 2012, doi: 10.1201/9781420010749.

[2] R. L. Galvez, A. A. Bandala, E. P. Dadios, R. R. P. Vicerra, and J. M. Z. Maningo, “Object Detection Using Convolutional Neural Networks,” IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON, vol. 2018- October, no. October, pp. 2023–2027, 2019, doi: 10.1109/TENCON.2018.8650517.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., pp. 580–587, 2014, doi: 10.1109/CVPR.2014.81.

[4] R. Girshick, “Fast R-CNN,” Proc. IEEE Int. Conf. Comput. Vis., vol. 2015 International Conference on Computer Vision, ICCV 2015, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.

[5] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards RealTime Object Detection with Region Proposal Networks,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[6] P. Dong and W. Wang, “Better region proposals for pedestrian detection with R-CNN,” 30th Anniv. Vis. Commun. Image Process., pp. 3–6, 2016, doi: 10.1109/VCIP.2016.7805452.

[7] W. Liu, D. Anguelov, D. Erhan, and C. Szegedy, “SSD: Single Shot MultiBox Detector,” ECCV, vol. 1, pp. 21–37, 2016, doi: 10.1007/978- 3-319-46448-0.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real- time object detection,” IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[9] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR, vol. 2017-Janua, pp. 6517–6525, 2017, doi: 10.1109/CVPR.2017.690.

[10] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” arXiv Prepr., 2018.

S. Ding, F. Long, H. Fan, L. Liu, and Y. Wang, “A novel YOLOv3-tiny network for unmanned airship obstacle detection,” IEEE 8th Data Driven Control Learn. Syst. Conf. DDCLS, pp. 277–281, 2019, doi: 10.1109/DDCLS.2019.8908875.

[12] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” IEEE CVPR, vol. 1, pp. 886–893, 2005, doi: 10.1109/CVPR.2005.177.

[13] C. Szegedy, W. Liu, Y. Jia, and P. Sermanet, “Going Deeper with Convolutions,” CVPR, 2015, doi: 10.1108/978-1-78973-723- 320191012.

[14] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, “Selective search for object recognition,” Int. J. Comput. Vis., vol. 104, no. 2, pp. 154–171, 2013, doi: 10.1007/s11263-013-0620- 5. [15] Z. Q.

Zhao, P. Zheng, S. T. Xu, and X. Wu, “Object Detection with Deep Learning: A Review,” IEEE Trans. Neural Networks Learn. Syst., vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

[16] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” ECCV, pp. 346–361, 2014, doi: 10.1023/B:KICA.0000038074.96200.69.

[17] R. Nabati and H. Qi, “RRPN: RADAR REGION PROPOSAL NETWORK FOR OBJECT DETECTION IN AUTONOMOUS VEHICLES,” IEEE Int. Conf. Image Process., pp. 3093–3097, 2019.

[18] L. Jiao et al., “A Survey of Deep Learning-Based Object Detection,” IEEE Access, vol. 7, pp. 128837–128868, 2019, doi: 10.1109/access.2019.2939201.

[19] D. Wang, C. Li, S. Wen, X. Chang, S. Nepal, and Y. Xiang, “Daedalus: Breaking Non-Maximum Suppression in Object Detection via Adversarial Examples,” arXiv Prepr., 2019.

[20] C. Ning, H. Zhou, Y. Song, and J. Tang, “Inception Single Shot MultiBox Detector for object detection,” IEEE Int. Conf. Multimed. Expo Work. ICMEW, no. July, pp. 549–554, 2017, doi: 10.1109/ICMEW.2017.8026312.

[21] Z. Chen, R. Khemmar, B. Decoux, A. Atahouet, and J. Y. Ertaud, “Real time object detection, tracking, and distance and motion estimation based on deep learning: Application to smart mobility,” 8th Int. Conf. Emerg. Secur. Technol. EST, pp. 1–6, 2019, doi: 10.1109/EST.2019.8806222.

[22] D. Xiao, F. Shan, Z. Li, B. T. Le, X. Liu, and X. Li, “A Target Detection Model Based on Improved Tiny-Yolov3 Under the Environment of Mining Truck,” IEEE Access, vol. 7, pp. 123757–123764, 2019, doi: 10.1109/access.2019.2928603.

[23] Q. C. Mao, H. M. Sun, Y. B. Liu, and R. S. Jia, “Mini-YOLOv3: RealTime Object Detector for Embedded Applications,” IEEE Access, vol. 7, pp. 133529–133538, 2019, doi: 10.1109/ACCESS.2019.2941547.

[24] W. Fang, L. Wang, and P. Ren, “Tinier-YOLO: A Real-time Object Detection Method for Constrained Environments,” IEEE Access, vol. 8, pp. 1935–1944, 2019, doi10.1109/ACCESS.2019.2961959.

[25] R. Huang, J. Pedoeem, and C. Chen, “YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers,” IEEE Int. Conf. Big Data, Big Data, pp. 2503–2510, 2019, doi: 10.1109/BigData.2018.8621865.

[26] J. Huang et al., “Speed/accuracy trade-offs for modern convolutional object detectors,” 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR, vol. 2017-Janua, pp. 3296–3305, 2017, doi: 10.1109/CVPR.2017.351.

[27] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, and K. Ouni, “Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3,” 1st Int. Conf. Unmanned Veh. Syst., pp. 1–6, 2019, doi: 10.1109/UVS.2019.8658300.