**TRAFFITIZER-EMERGENCY RESPONSE SYSTEM**

A Project Report

*Submitted in partial fulfillment of the requirements for*

*the award of the degree of*

**BACHELOR OF TECHNOLOGY**

*in*

**COMPUTER SCIENCE & ENGINEERING**

*from*

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY**

**HYDERABAD**

By

**SHIVARAM.K**(16C11A05B0)

**SAMRAJ.M**(16C11A05A2)

**VIVEK KUMAR REDDY.K**(16C11A05E6)

**GOPINATH.G**(17C1A0502)

Under the guidance of

**Mr. K.VIJAY KUMAR M.Tech**.

Assistant Professor



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**ANURAG ENGINEERING COLLEGE**

**AUTONOMOUS**

(Affiliated to JNTU-Hyderabad & Approved by AICTE-New Delhi)

ANANTHAGIRI (V) (M), SURYAPETA (Dt), TELANGANA -508206.

www.anurag.ac.in

**2018-2019**

**ANURAG ENGINEERING COLLEGE**

**AUTONOMOUS**

(Affiliated to JNTU-Hyderabad & Approved by AICTE-New Delhi )

ANANTHAGIRI (V) (M), SURYAPETA (D), TELANGANA -508206.

Ph: 08683- 272555, 272456, 272454

www.anurag.ac.in



***CERTIFICATE***

*This is to certify that theproject work entitled “****TRAFFITIZER-EMERGENCY RESPONSE SYSTEM****” is a bonafide work done by “****SHIVARAM.K (16C11A05B0), SAMRAJ.M (16C11A05A2), VIVEK KUMAR REDDY (16C11A05E6) & GOPINATH.G (17C1A0502)****” in the partial fulfillment for the award of Bachelor of Technology in Computer Science & Engineering from JNTU, Hyderabad during the year* ***2019- 2020****.*

*This work has not been submitted to any other university or institute or organization for the award of any degree or diploma.*

(Mr. P. SANDEEP REDDY) (Dr. M. MURUGESAN Ph.D., M.Tech)

**SUPERVISOR H.O.D., C.S.E**

(Dr. M.V. SIVA PRASAD Ph.D., M.Tech) **EXTERNAL EXAMINER PRINCIPAL**

**ACKNOWLEDGEMENTS**

This report will certainly not be completed without due acknowledgements paid to all those who helped us during this project work.

We would like to express our gratefulness to the project trainer **Mr.RAKESH REDDY GADDAM** and **UNAVO, HYDERABAD** for their valuable guidance in carrying out this project work.

We express our sincere thanks to our supervisor **Mr. P. SANDEEP REDDY,** for giving me moral support, kind attention and valuable guidance throughout this project work.

It is our privilege to thank **Dr. M. MURUGESAN***,* Head of the Departmentfor his encouragement during the progress of this project work.

We are thankful to both Teaching and non-teaching staff members of **CSE DEPARTMENT** for their kind cooperation and all sorts of help in bringing out this project work successfully.

We derive great pleasure in expressing our sincere gratitude to the principal **Dr. M.V. SIVA PRASAD** for his timely suggestions, which helped us to complete this work successfully.

We are thankful to the **ANURAG ENGINEERING COLLEGE** for providing required facilities during the project work.

We would like to thank our parents and friends for being supportive all the time, and we are very much obliged to them.

**SHIVARAM.K**

**SAMRAJ.M**

**VIVEK KUMAR REDDY.K**

**GOPINATH.G**

**CONTENTS**

Table of Contents i

ABSTRACT ii

LIST OF FIGURES iii

LIST OF TABLES v

LIST OF ACRONYMS vi

**I. INTRODUCTION**

1.1 Objective 03

1.2 Problem Statement 03

1.3 Existing System 03

1.4 Proposed System 04

**II. SYSTEM ANALYSIS**

2.1 Literature Survey 06

2.2 Requirements Specification 06

2.3 Feasibility study 07

**III. SYSTEM DESIGN**

3.1 Modules 09 3.2 Design Representation 13

**IV. IMPLEMENTATION**

4.1 Technologies 19

4.2 Sample Code 20

**V. RESULTS** 29

**VI. TESTING**

5.1 Test Levels 35

5.2 Test Cases 36

**VII. CONCLUSION**  42

**VIII. FUTURE ENHANCEMENTS** 44

**REFERENCES** 46

**BIBLIOGRAPHY** 47

**ABSTRACT**

Agriculture is vital to human survival and continues to be a major driver of several economies around the world, especially in underdeveloped and developing economies. with rising demand for food and cash crops due to a growing global population and the challenges raised by climate change, there is an urgent need to raise farm yields while incurring minimal costs. With the help of conventional machine learning models like K-Nearest Neighbor (KNN) and Support Vector Machine (SVM). Plant seedlings classification is having less accuracy, less speed and more time to train. With the help of Convolutional Neural Network (CNN), a deep learning model, can classify the plant seedlings with more accuracy, faster and less training time.

A Kaggle Competition Plant-Seedlings dataset (Dyrmann, 2017) that contains 4,275 images of approximately 960 unique plants belonging to 12 species at several growth stages are used to compare the performances of two traditional algorithms (SVM and KNN) and a Convolutional Neural Network (CNN), The performance show that CNN driven seedling classification is more accurate than SVM and KNN based classification.

**LIST OF FIGURES**

**Figure Description Page**

3.1.1.8 Max pooling 2X2 11

3.1.1.9 ReLU activation function 12

3.2.1 CNN framework 13

3.2.2 Use case diagram 14

3.2.3 Model framework 15

3.2.4 Activity diagram 16

3.2.5 Flow chart 17

3.2.6 Sequence diagram 30

5.1 Result 30

5.2Result 31

5.3Result 31

5.4 Accuracy comparison 32

5.5 Accuracy of CNN 32

5.6 Accuracy of SVM 32

5.7 Accuracy of KNN 33

6.2.3.1 Unit testing 39

6.2.3.2 Unit testing 39

6.2.3.3 Unit testing 40

6.2.3.4 Unit testing 40

6.2.5.1 Validation testing 41

6.2.5.2 Validation testing 41

**LIST OF TABLES**

**Table Description Page**

2.1 Accuracy of different classification algorithms 6

6.2.1White box testing 36

6.2.2Blackbox testing 38

**LIST OF ACRONYMS**

**CNN** :Convolutional Neural Network

**CONV** : Convolutional

**CPU**  : Central Processing Unit

**FC**  : Fully Connected

**HTML** : Hypertext Markup Language

**KNN**  : K-Nearest Neighbors

**ReLU** : Rectified Linear Unit

**SVM**  : Support Vector Machine

**CHAPTER 1**

**INTRODUCTION**

In the last few decades, the development of image and video processing algorithms and approaches has received significant attention from the scientific community. This development is a consequence of the increased hardware and sensing capabilities and the increased quality of the digitally recorded materials. The increase of quality of the recorded materials and the increased computing capabilities have allowed the development of more complex and more powerful approaches.

The large interest in the development of such algorithms is mainly caused by the large number of potential applications that could use them. The number of industries that could use the advancements of image processing and machine learning for the purpose of object recognition or segmentation from images is quite large. Some of the prospect applications are medical applications, agriculture, ecology, quality inspection in automated production for machine parts, the military industry and many others. The ultimate goal is to build applications that could use machine learning and vision for process automation that could increase the productivity and ease the burden of the everyday life.

There are multiple tasks at hand that could use image processing and machine vision. Some of them are: Object recognition in images, image segmentation to extract regions of interest, image database search etc. All of these tasks are equally important for the development of the computer sciences and especially for the development of robotics and intelligent systems where these approaches could be beneficial.

Plant species recognition is a new and popular topic. There are a large number of challenges when applying machine vision and learning algorithms for plant species recognition. In recent years, there is a significant expansion and increment in the development of such approaches that could be useful for plant species recognition and plant segmentation, detection of water level in the fields, weed detection in plantations etc. The research in these topics has the final goal of full automation of the agricultural production from the phase of planting the seeds to the phase of collecting the crops. While we are still a long way from this final goal, there are significant advancements in the recognition of plants, weed detection and detection of plant deceases and water levels. The proposed algorithms should always be very precise and the margin of error should be very little since a wrong detection could be very damaging to the agricultural production

**1.1 OBJECTIVE :-**

The objective of this project is to classify the Plant-seedlings belonging to the 12 different species by using deep learning model. This project uses one of Kaggle Competition Plant-Seedlings dataset (Dyrmann, 2017), this dataset containing images of approximately 960 unique plants belonging to 12 species at several growth stages. With the help of this plant-seedlings dataset (Dyrmann, 2017) the model is trained and tested.

**1.2 PROBLEM STATEMENT :-**

The target of this project is to classify a crop seeding of 12 different plant species, hence, it’s a multi-classification problem. As the image of a crop seedling is taken and output the correspondent specie from our 12 classes. CNN model is used to classify the plants. As mentioned below. CNN is widely used in the field of computer vision for doing complicated tasks, hence it’ll be using for classification.

**1.3 EXISTING SYSTEM :-**

Support Vector Machines and K-Nearest Neighbours To perform baseline tests, no neural network techniques is used: Support Vector Machines (SVMs) and K-Nearest Neighbours classifiers. To find optimal parameters for each model, grid search is performed using a combination of parameters. For the KNN model, using values for the number of neighbours parameter, n\_neighbours, ranging from [1,p,n], where n = input size, grid search is used to find our best model to have, number of neighbours = 5 with uniform weights and accuracy of 56.84%.

For our SVM classifier, using similar grid search technique, the optimal parameters find to be: penalty parameter of the error term, C=5, kernel=linear and gamma value=auto which uses 1 / n features as the kernel coefficient. Our accuracy for this model is 61.2%.

**1.3.1 DRAWBACKS :-**

🡪Lower accuracy

🡪Higher training time

🡪Takes more time to classify (produce outputs)

🡪Complex design and theory

**1.4 PROPOSED SYSTEM :-**

The neural network architecture has 6 convolutional layers. Each is followed with a Rectified Linear Unit (ReLU). The first two convolutional layers have 64 filters, the next has 128 while the last one has 256. Each convolutional layer has zero padding. After each pair of convolutional layer, max pooling layer is used for dimensionality reduction and a 10% dropout to prevent over-fitting. At the end of the six convolutional layers are 3 fully connected layers. The last fully connected layer has a softmax activation function which outputs probability distribution for each of the 12 classes. Adam optimizer is used with a batch size of 32 for each step and a weighted cross-entropy loss, to handle the imbalanced number of pixels for each class. The accuracy acquired is around 80.21.

**1.4.1 ADVANTAGES :-**

🡪Higher accuracy

🡪Lower training time

🡪Simple design

🡪Classifies image very fast (rapid output)

**CHAPTER 2**

**SYSTEM ANALYSIS**

**2.1 LITERATURE SURVEY :-**

With the help of existing Machine Learning algorithms and the given data set it is observed that lower accuracy (75-80) is achieved. It is also observed that it takes more time to train the model. Support Vector Machines and K-Nearest Neighbours To perform baseline tests, usage of non-neural network techniques: Support Vector Machines (SVMs) and K-Nearest Neighbours classifiers. To find optimal parameters for each model, grid search is performed using a combination of parameters. For the KNN model, using values for the number of neighbours parameter, n\_neighbours, ranging from [1,p,n], where n = input size, grid search is used to find our best model to have, number of neighbours = 5 with uniform weights and accuracy of 56.84%. For our SVM classifier, using similar grid search technique, optimal parameters are to be: penalty parameter of the error term, C=5, kernel=linear and gamma value=auto which uses 1 / n\_features as the kernel coefficient. Our accuracy for this model is 61.2%.

|  |  |
| --- | --- |
| ALGORITHM | ACCURACY |
| KNN | 56.84 |
| SVM | 61.20 |
| CNN | 80.21 |

**Table 2.1** table showing accuracy of different classification algorithms

**2.2 REQUIREMENTS SPECIFICATION :-**

The following are the hardware and software requirements that have used to implement the proposed system

**2.2.1 hardware requirements:**

Processor : Intel i7 7th generation processor

RAM : 16GB recommended

Operating system : Microsoft windows 10

**2.2.2 software requirements :-**

Operating System : Windows 10

Programming Language : python 3.7.0

IDE : Jupyter 6.0.1, pycharm 2019.2.3

Scripting Languages : python 3.7.0

Scripting Tools : HTML 4

Web Server : FLASK 1.0.2

Web Browser : CHROME

**2.3 FEASIBILITY STUDY :-**

Sometimes a feasibility study is done as part of a systems development lifecycle, in order to drive precision for the implementation of technologies. Engineers might look at a five-point model called TELO — this includes the following components:

* Technical
* Economic
* Legal
* Operational

**2.3.1 Technical :-**

The technology that is used to develop this project is technically feasible to system and user in all the ways.

**2.3.2 Economic :-**

Since the project is rendered with the help of open source libraries economically it is feasible to develop and use the project.

**2.3.3 Legal :-**

This project does not have any legal issues and it is free to implement and use, since open source technologies and dataset are being used.

**2.3.4 Operational :-**

A person with little knowledge on computer can be able to use the model. It is very precise.

**CHAPTER 3**

**SYSTEM DESIGN**

**3.1 MODULES :-**

**3.1.1 CNN module :-**

In general, Neural networks accept a single vector as input, transform it to a series of hidden layers, which in turn is made up of set of neurons that are fully connected to all neurons in the previous layer. Neurons of the same layer are independent and do not share any connections. After the hidden layers, is the last fully connected layer which is also called the ‘output layer’, where each node outputs score for each class. The downside of regular neural network is that they don’t scale well to full images. It's mainly because with images of decent size, the number of neurons and weights that the network must accommodate becomes unmanageable. This is where Convolutional Neural Network comes to rescue with its neurons arranged in 3 dimensions (width, height, depth).Each of the layer in CNN accepts 3D input volume and transforms it into 3D output volume. Following is a simple visualization of how CNN arranges its neurons in 3dimensions (width, height, depth)

**3.1.1.1 CONV layer :-**

The CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. For eg, if 32 filters are used, CONV layer will output a volume equal to (256x256x3) Activation functions Activation layer will apply an element wise activation functions, leaving the volume unchanged.

**3.1.1.2 POOL layer :-**

Pool layer performs down sampling operation along the spatial dimensions (width, height), outputting a reduced volume than the previous layer. For eg,(128x128x32)

**3.1.1.3 Fully-Connected Layer :-**

FC layer, also called as the dense layer, each neuron will be connected to all the neurons of the previous layer. FC layer when used as the output layer results in much reduced volume of size [1x1xm], where m is the number of categories that are to be predicted. Each of the nodes of fully connected layer outputs a score corresponding to a class score.

**3.1.1.4 Dropout layer :-**

Dropout layer is used as a method of regularization to combat over-fitting of the training set. It ‘drops’ neurons at random (depending on the probability mentioned) while calculating the forward prop and backward prop, resulting in a simpler version of the CNN for each iteration and hence giving the model a hard time to overfit the training set.

**3.1.1.5 CNN algorithm :-**

consists of several convolution (CNV) operations followed of the image sequentially which is followed by pooling operation (PL) to generate the neurons feed into fully connected (FC) layer. I used the Keras Sequential API, where you have just to add one layer at a time, starting from the input.

**3.1.1.6 Input :-**

CNV is typically 2D image data

**3.1.1.7 The first is the convolutional (Conv2D) layer** :-

It is like a set of learnable filters. I chose to set 32 filters for the two firsts conv2D layers and 64 filters for the 2nd convolutional layer and 128 filters for the two last ones. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image. The CNN can isolate features that are useful everywhere from these transformed images (feature maps).

**3.1.1.8 The second important layer in CNN is the pooling :-**

(MaxPool2D) layer. This layer simply acts as a down sampling filter. It looks at the 2 neighbouring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. Choosing the pooling size (i.e the area size pooled each time) more the pooling dimension is high, more the down sampling is important. This figure below illustrate Max pooling with a 2x2 filter and stride = 2:



**Figure 3.1.1.8** Max pooling 2X2

Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

Dropout is a regularization method, where a proportion of nodes in the layer are randomly ignored (setting their weights to zero) for each training sample. This drops randomly a proportion of the network and forces the network to learn features in a distributed way. This technique also improves generalization and reduces the overfitting.

**3.1.1.9 ReLU Layer :-**

ReLU is the abbreviation of Rectified Linear Units. This layer applies the non-saturating activation function. *f*(*x*) = *x*+ = *max*(0, *x*) It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer. Figure below illustrate ReLU function: Plot of the rectifier (blue) and softplus(green) functions near x = 0



**Table 3.1.1.9** ReLU Activation function

**3.1.1.10 The Flatten layer :-**

It is used to convert the final feature maps into a one single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers. In the end features are used in two fully-connected (Dense) layers which is just artificial neural networks (ANN) classifier. In the last layer (Dense(10,activation="softmax")) the net outputs distribution of probability of each class.

**3.1.1.11 Fully Connected (FC) Layer :-**

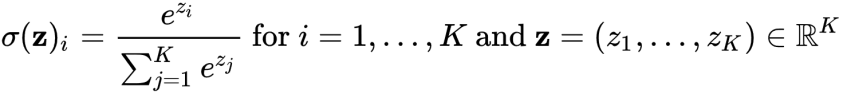
This layer will reduce the size of input data to the size of classes that the CNN is trained for by combining output of CNV layer with different weights. Each neuron at the output of the CNV layer will be connected to all other neurons after weighted properly, Similar to CNV layer, weight of these taps in FC layer is found though backpropagation algorithm.

**3.1.1.12 Classification Layer (CL) :-**

This is the final layer of the CNN that converts the output of FC to probability of each object being in a certain class. Typically, soft-max type of algorithms are used in this layer.

**3.1.1.13 Softmax activation function :-**

In mathematics, the softmax function, also known as soft argmaxor normalized exponential function, is a function that takes as input a vector of *K* real numbers and normalizes it into a probability distribution consisting of *K* probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval {\displaystyle (0,1)}, and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities. Softmax is often used in neural networks to map the non-normalized output of a network to a probability distribution over predicted output classes. The standard (unit) softmax function {\displaystyle \sigma :\mathbb {R} ^{K}\to \mathbb {R} ^{K}}is defined by the formula



**3.2 DESIGN REPRESENTATION :-**

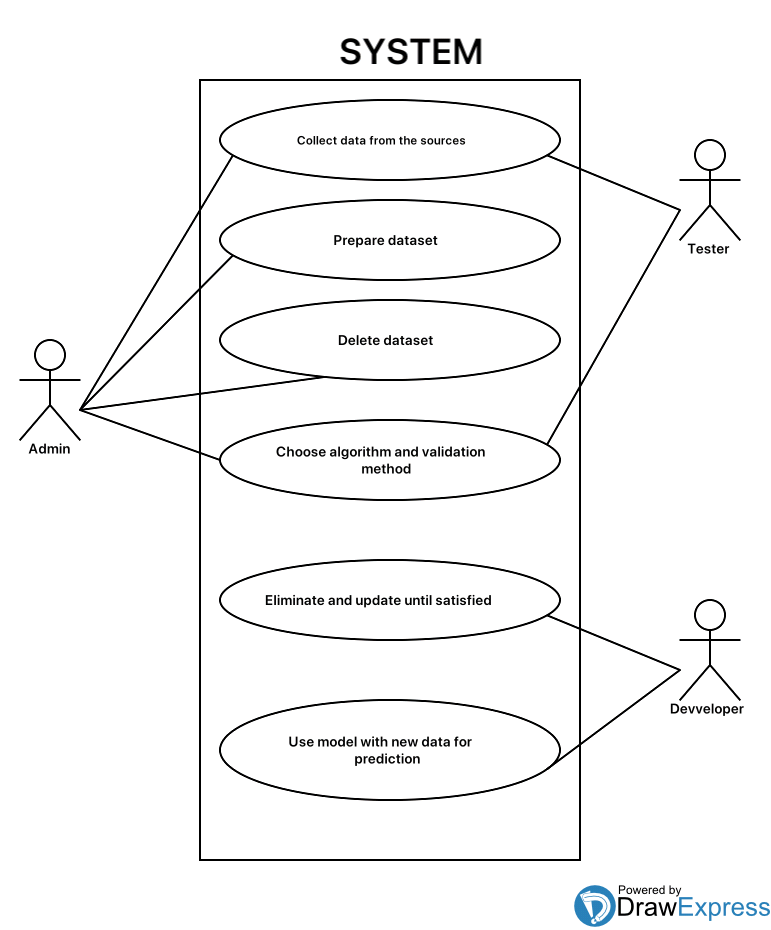
**3.2.1 CNN framework :-**

**A close up of a map

Description automatically generated**

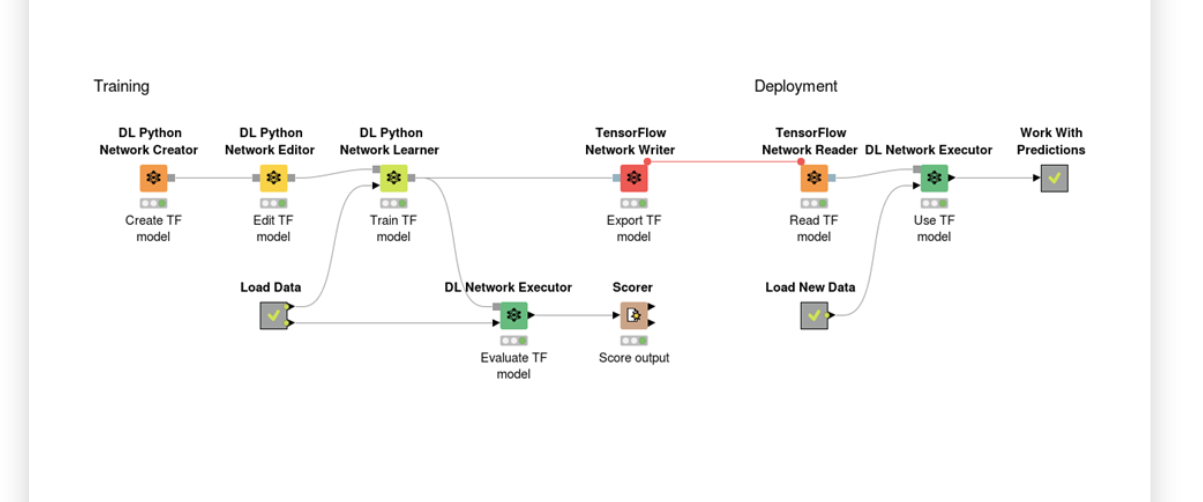
**Figure 3.2.1** CNN framework

**3.2.2 Use case diagram :-**



**Figure 3.2.2** Use case diagram

**3.2.3 Model framework :-**

****

**Figure 3.2.3** Model framework

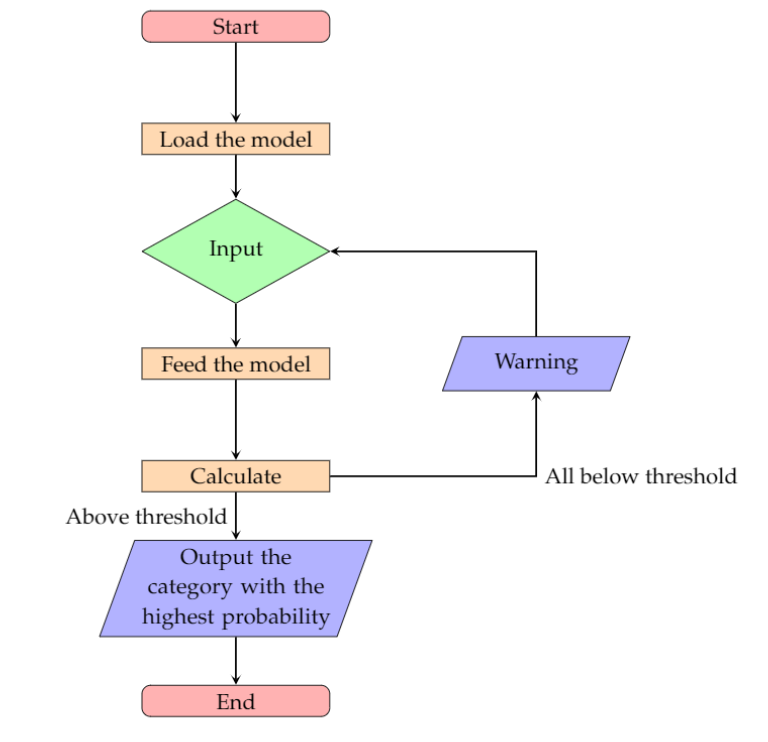
**3.2.4 Activity diagram :-**

**A screenshot of a cell phone

Description automatically generated**

**Figure 3.2.4** Activity diagram

**3.2.5 Flow chart :-**

****

**Figure 3.2.5** Flow chart

**3.2.6 Sequence diagram :-**

**A screenshot of a cell phone

Description automatically generated**

**Figure 3.2.6** Sequence diagram

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 TECHNOLOGIES :-**

**4.1.1 Jupyter notebook :-**

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

**4.1.2 python :-**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991. Python’s design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed  and garbage-collected. It supports multiple programming paradigms including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library .

**4.1.3 Keras :--**

Keras is a high-level neural networks API, written in Python and capable of running on top of Tensorlow, CNKT, or Theano. It was developed with a focus on enabling fast experimentation.

Use of Keras if needed a Deep Learning library that:

🡪Allows for easy and fast prototyping (through user friendliness, modularity and extensibility).

🡪Supports both convolutional networks and recurrent networks, as well as combinations of the two.

🡪Runs seamlessly on CPU and GPU.

**4.2 SAMPLE CODE :-**

**4.2.1 PSC.py :-**

//PSC.PY

from keras.models import Sequential

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.layers.core import Activation, Flatten, Dense

from sklearn.model\_selection import GridSearchCV

from keras.layers import BatchNormalization

from keras.wrappers.scikit\_learn import KerasClassifier

from keras.preprocessing.image import ImageDataGenerator, array\_to\_img, img\_to\_array, load\_img

from keras.optimizers import Adam, SGD, RMSprop, Adagrad, Adadelta

from sklearn.model\_selection import train\_test\_split

import numpy as np

import random

import os

import sys

import cv2

from keras.utils import to\_categorical

import matplotlib

from subprocess import check\_output

def classes\_to\_int(label):

# label = classes.index(dir)

label = label.strip()

if label == "Black-grass": return 0

if label == "Charlock": return 1

if label == "Cleavers": return 2

if label == "Common Chickweed": return 3

if label == "Common wheat": return 4

if label == "Fat Hen": return 5

if label == "Loose Silky-bent": return 6

if label == "Maize": return 7

if label == "Scentless Mayweed": return 8

if label == "Shepherds Purse": return 9

if label == "Small-flowered Cranesbill": return 10

if label == "Sugar beet": return 11

print("Invalid Label", label)

return 12

def int\_to\_classes(i):

if i == 0: return "Black-grass"

elif i == 1: return "Charlock"

elif i == 2: return "Cleavers"

elif i == 3: return "Common Chickweed"

elif i == 4: return "Common wheat"

elif i == 5: return "Fat Hen"

elif i == 6: return "Loose Silky-bent"

elif i == 7: return "Maize"

elif i == 8: return "Scentless Mayweed"

elif i == 9: return "Shepherds Purse"

elif i == 10: return "Small-flowered Cranesbill"

elif i == 11: return "Sugar beet"

print("Invalid class ", i)

return "Invalid Class"

#1.Read the images and generate the train and test dataset (5 points)

NUM\_CLASSES = 12

# we need images of same size so we convert them into the size

WIDTH = 128

HEIGHT = 128

DEPTH = 3

inputShape = (WIDTH, HEIGHT, DEPTH)

# initialize number of epochs to train for, initial learning rate and batch size

EPOCHS = 25

INIT\_LR = 1e-3

BS = 32

def readTrainData(trainDir):

data = []

labels = []

# loop over the input images

dirs = os.listdir(trainDir)

for dir in dirs:

absDirPath = os.path.join(os.path.sep,trainDir, dir)

images = os.listdir(absDirPath)

for imageFileName in images:

# load the image, pre-process it, and store it in the data list

imageFullPath = os.path.join(trainDir, dir, imageFileName)

#print(imageFullPath)

img = load\_img(imageFullPath)

arr = img\_to\_array(img) # Numpy array with shape (233,233,3)

arr = cv2.resize(arr, (HEIGHT,WIDTH)) #Numpy array with shape (HEIGHT, WIDTH,3)

#print(arr.shape)

data.append(arr)

label = classes\_to\_int(dir)

labels.append(label)

return data, labels

def createModel():

model = Sequential()

# first set of CONV => RELU => POOL layers

# The CONV layer will learn 20 convolution filters, each of which are 5×5.

model.add(Conv2D(32, (3,3), padding="same", input\_shape=inputShape))

#model.add(BatchNormalization())

model.add(Activation("ReLU"))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

# second set of CONV => RELU => POOL layers

model.add(Conv2D(64, (3,3), padding="same"))

#model.add(BatchNormalization())

model.add(Activation("ReLU"))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

model.add(Conv2D(128, (3,3), padding="same"))

#model.add(BatchNormalization())

model.add(Activation("ReLU"))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

model.add(Flatten())

model.add(Dense(units=500))

model.add(Activation("ReLU"))

model.add(Dense(units=12))

model.add(Activation("softmax"))

opt = Adam(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)

# use binary\_crossentropy if there are two classes

model.compile(loss="categorical\_crossentropy", optimizer=opt, metrics=["accuracy"])

return model

#2.Divide the data set into Train and validation data sets

random.seed(10)

allLabels = os.listdir("C:\\Users\\shivaramakrishna\\Desktop\\updated project\\train\\train") # list of subdirectories and files

print("Loading images...")

sys.stdout.flush()

X, Y = readTrainData("C:\\Users\\shivaramakrishna\\Desktop\\updated project\\train\\train")

# scale the raw pixel intensities to the range [0, 1]

X = np.array(X, dtype="float") / 255.0

Y = np.array(Y)

# convert the labels from integers to vectors

Y = to\_categorical(Y, num\_classes=12)

print("Parttition data into 75:25...")

sys.stdout.flush()

# partition the data into training and testing splits using 75% training and 25% for validation

(trainX, valX, trainY, valY) = train\_test\_split(X,Y,test\_size=0.25, random\_state=10)

#3. Initialize & build the model (10 points)

#construct the image generator for data augmentation

print("Generating images...")

sys.stdout.flush()

aug = ImageDataGenerator(rotation\_range=30, width\_shift\_range=0.1, \

height\_shift\_range=0.1, shear\_range=0.2, zoom\_range=0.2,\

horizontal\_flip=True, fill\_mode="nearest")

# initialize the model

print("compiling model...")

sys.stdout.flush()

model = createModel()

# train the network

print("training network...")

sys.stdout.flush()

H = model.fit\_generator(aug.flow(trainX, trainY, batch\_size=BS), \

validation\_data=(valX, valY), \

steps\_per\_epoch=len(trainX) // BS, epochs=EPOCHS, verbose=1)

# save the model to disk

print("Saving model to disk")

sys.stdout.flush()

model.save("C:\\Users\\shivaramakrishna\\Desktop\\updated project\\model.h5")

**4.2.2 app.py :-**

from flask import Flask

from flask import request

from flask import jsonify

from flask import render\_template

from flask import send\_from\_directory

import test

import json

import numpy as np

# @app.route('/result/',)

# from flask import render\_template, Flask

# app=Flask(\_\_name\_\_,)

app = Flask(\_\_name\_\_,template\_folder='templates', static\_folder='static')

model=test.loading\_model()

@app.route('/',methods=['GET','POST'])

def index():

return render\_template("index.html")

@app.route('/prediction',methods=['GET','POST'])

def prediction():

if request.method == 'POST':

link = request.form['in']

# url = request.args.get('d')

out= test.predcit(link,model)

# return jsonify(int(out))

return render\_template('index.html',pred=out)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**4.2.3 test.py :-**

import os

import sys

import numpy as np

from PIL import Image

import requests

from io import BytesIO

from keras.models import load\_model

import tensorflow as tf

graph = tf.get\_default\_graph()

from keras.utils import CustomObjectScope

from keras.initializers import glorot\_uniform

def loading\_model():

# model=load\_model('model.h5')

with CustomObjectScope({'GlorotUniform': glorot\_uniform()}):

model = load\_model('model.h5')

print('model loaded')

return model

def predcit(url,model):

response=requests.get(url)

img=Image.open(BytesIO(response.content))

img=np.array(img.resize((128,128)))

img=img.reshape(1,128,128,3)

img = np.array(img, dtype="float") / 255.0

print(img)

with graph.as\_default():

b=model.predict(img)

label=np.argmax(b)

label\_name = ['Black -grass', 'Charlock','Cleavers' ,'Common Chickweed','Common Wheat' ,'Fat Hen','Loose Silky-bent' ,'Maize','Scentless Mayweed','Shepherds Purse','Small -floweres cranesbill' ,'Sugar beet']

#return label

return label\_name[label]

**CHAPTER 5**

**RESULTS**

**5.1 Home page :-**

Home page of the project has a text field and a submit button.

A screenshot of a computer screen

Description automatically generated

**Figure 5.1** Home page

**5.2 Providing input :-**

The URL of the image need to classify is to be pasted in text field and hit the submit button.

A screenshot of a computer

Description automatically generated

**Figure 5.2** Providing input

**5.3 Output :-**

After the image is classifies by the backend CNN model the output will be shown on the screen.

A screenshot of a computer screen

Description automatically generated

**Figure 5.3** Output

5.4 **Accuracy comparison :-**

The accuracy of the three models is being compared and plotted in the form of graph. This comparison shows CNN model is having highest accuracy(~81%) than KNN and SVM models.

**Figure 5.4** accuracy comparison

**5.5 Accuracy and loss of CNN :-**

The accuracy and loss graph is plotted of CNN model is plotted. The graph shows that the model acquired gives around 80.21% accuracy and 19.79% loss.

A picture containing kite, flying, sky

Description automatically generated

**Figure 5.5** accuracy and loss of model

**5.6 Accuracy of SVM :-**

The accuracy and loss graph of KNN is plotted. From the graph we can observe the accuracy achieved is around 61% and loss is 49%.

A close up of a map

Description automatically generated

**Figure 5.6** Accuracy of SVM model

**5.7 Accuracy of KNN :-**

The accuracy and loss graph of SVM is plotted. From the graph we can observe that accuracy achieved using SVM is around 51%.

A close up of a map

Description automatically generated

**Figure 5.7** Accuracy of KNN model

**CHAPTER 6**

**TESTING**

**6.1 TEST LEVELS :-**

**6.1.1 Whitebox testing :-**

There are 12 kinds of species that are needed to classify. Here testing is done against all these 12 kinds of species so that the code executes with all the conditions and loops with boundaries. Correct output is resulted.

**6.1.2 Blackbox testing :-**

The model finds trouble while classification is done between loose silky bent and grass. When the image of loose silky bent is passed to model it show result as black grass. It is due to margin of accuracy.

**6.1.3 Unit testing :-**

With the help of this unit testing every small internal logic of the code is checked and executed perfectly. It extracts the image and supply it to the model. The model evaluates and produce outputs and that output is popped on the screen very efficiently.

**6.1.4 Integration testing :-**

By this testing it is observed that the project is having the modules that are properly integrated and there is perfect switching between the modules in the project.

**6.1.5 Validation testing :-**

In order to run project the system needs internet connection. Unless there is no internet connection it pops out an exception. The input must be a path specifying location of an image that is need to be classified, unless the path contains no image it shows an exception.

**6.2 TEST CASES :-**

**6.2.1 WHITEBOX TESTING :-**

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Expected Output | Actual Output | Result |
| A picture containing ground, outdoor, grass  Description automatically generated | Black-grass | Black-grass | OK |
| A close up of food  Description automatically generated | Charlock | Charlock | OK |
| A close up of food  Description automatically generated | Cleavers | cleavers | OK |
| A close up of food  Description automatically generated | Common chickweed | Common chickweed | OK |
| A picture containing food  Description automatically generated | Common wheat | Common wheat | OK |
| A picture containing food, indoor  Description automatically generated | Fat hen | Fat hen | OK |
| A close up of food  Description automatically generated | Loose silky bent | Loose silky bent | OK |
| A close up of a rock  Description automatically generated | Maize | Maize | OK |
| A close up of food  Description automatically generated | Scentless maywood | Scentless maywood | OK |
| A close up of food  Description automatically generated | shepherds purse | shepherds purse | OK |
| A close up of food  Description automatically generated | Small flowered cranesbill | Small flowered cranesbill | OK |
| A close up of food  Description automatically generated | Sugar beet | Sugar beet | OK |

**Table 6.2.1** white box testing

**6.2.2 BLACKBOX TESTING :-**

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Result |
| A picture containing food, wall  Description automatically generated | Loose silky bent | Black-grass | Fail |

**Table 6.2.2** Blackbox testing

**6.2.3 UNIT TESTING :-**

**6.2.3.1 RUNNING PROJECT IN COMMAND PROMPT :-**

A screenshot of a computer

Description automatically generated

**Figure 6.2.3.1** running project in command prompt

**6.2.3.2 RUNNING PROJECT ON BROWSER :-**

A screenshot of a computer screen

Description automatically generated

**Figure 6.2.3.2** running project on browser

**6.2.3.3 GIVING INPUT TO THE MODEL :-**

A screenshot of a computer

Description automatically generated

**Figure 6.2.3.3** giving input to the model

**6.2.3.4 OUTPUT SCREEN :-**

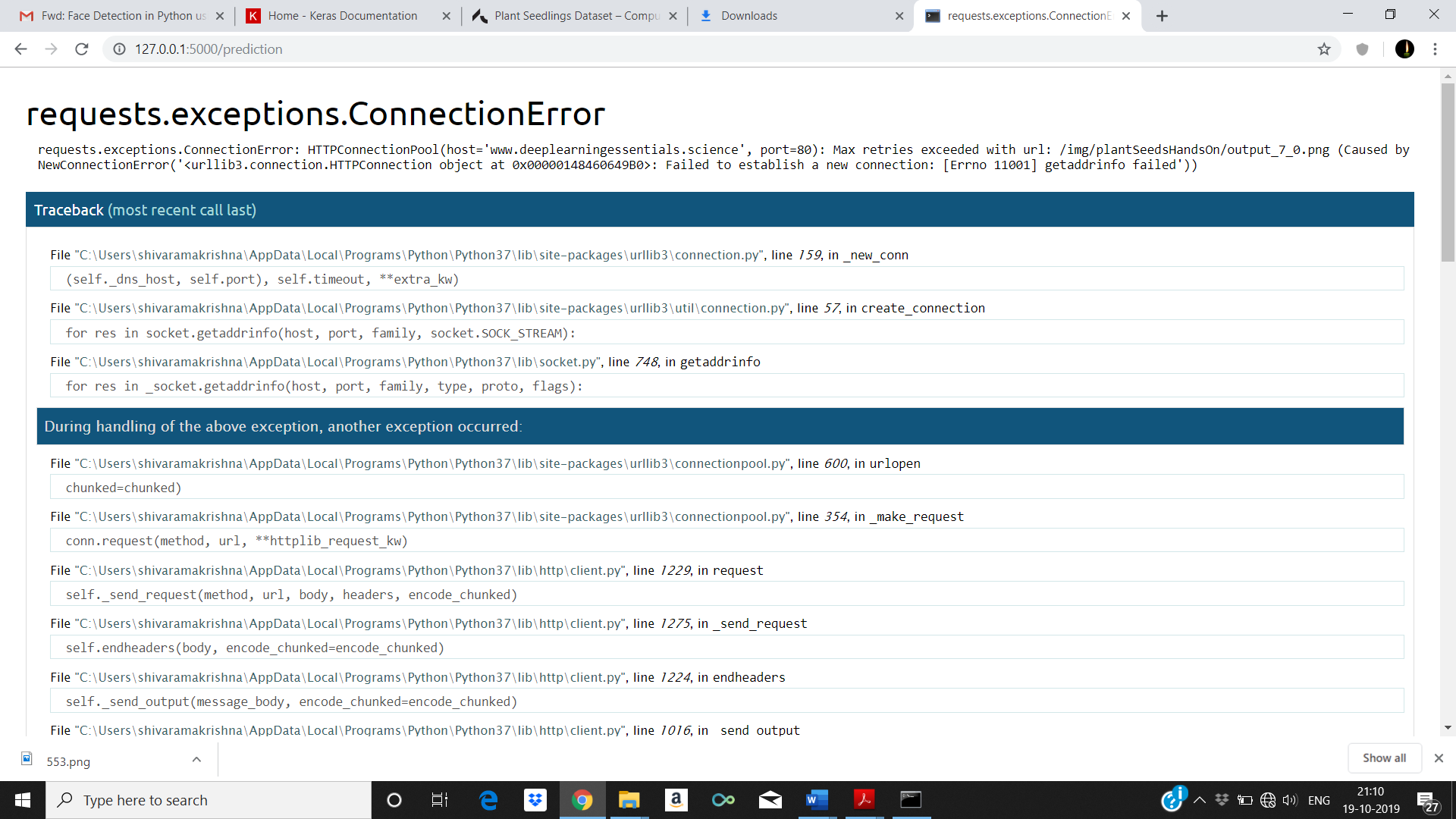
A screenshot of a computer screen

Description automatically generated

**Figure 6.2.3.4** output screen

**6.2.5 VALIDATION TESTING :-**

**6.2.5.1 RUNNING PROJECT WITHOUT INTERNET CONNECTION :-**



**Figure 6.2.5.1** connection error screen

**6.2.5.2 PATH CONTAINING NO IMAGE :-**

A screenshot of a computer

Description automatically generated

**Figure 6.2.5.2** OS error

**CHAPTER 7**

**CONCLUSION**

Firstly, CNN model is developed by using 6 layers, different filters and activation functions. The plant seedlings classification dataset (Dyrmann, 2017) extracted from Kaggle website is being used to train and test the model. The trained model is saved and used for classification of plant seedlings. By observing the results and accuracy it is proved that the CNN model outperformed the accuracy and performance of Machine Learning models like KNN and SVM.

**CHAPTER 8**

**FUTURE ENHANCEMENTS**

As an improvement and future work, data masking on the training set is to be done. Noise from the background of the images can be cancelled by masking images. It is believe that without the background noise and restricting the visibility to the green leaves, the model can be trained better, and this may notice significant improvement in the performance. Another implementation that can be tried is data augmentation. As the dataset is highly unbalanced, augmenting data to the under-represented classes might give a good boost to the total number of training images yielding a well-balanced dataset. Training the model on such dataset may give us significant improvement in the performance.

**REFERENCES:**

[1] T. Giselsson, R. Jørgensen, P. Jensen, M. Dyrmann, and H. Midtiby, A public image database for

benchmark of plant seedling classification algorithms, CoRR, abs/1711.05458, 2017.

[2] S. Lee, C. Chan, S. Mayo, and P. Remagnino, How deep learning extracts and learns leaf features

for plant classification, Pattern Recognition, vol. 71, pp. 1-13, 2017.

[3] Jeon, Wang-Su, and Sang-Yong Rhee, Plant leaf recognition using a convolution neural network,

International Journal of Fuzzy Logic and Intelligent Systems 17, no. 1, pp 26-34. 2017.

[4] Y. Sun, Y. Liu, G. Wang, and H. Zhang, Deep learning for plant identification in natural

environment, Computational Intelligence and Neuroscience, 2017.

[5] A. Milioto, P. Lottes, and C. Stachniss, Real-time semantic segmentation of crop and weed

for precision agriculture robots leveraging background knowledge in CNNs, cs.CV, 2018. URL:arXiv:1709.06764v2.

# Bibliography

[Dyrmann, M. (2017, november 15).](C:\\Users\\shivaramakrishna\\Downloads\\Dyrmann, M. (2017, november 15). kaggle. Retrieved from kaggle: https:\\arxiv.org\\abs\\1711.05458) *[kaggle.](C:\\Users\\shivaramakrishna\\Downloads\\Dyrmann, M. (2017, november 15). kaggle. Retrieved from kaggle: https:\\arxiv.org\\abs\\1711.05458)* [Retrieved from kaggle: https://arxiv.org/abs/1711.05458](C:\\Users\\shivaramakrishna\\Downloads\\Dyrmann, M. (2017, november 15). kaggle. Retrieved from kaggle: https:\\arxiv.org\\abs\\1711.05458)