WILD PLANTS EDIBILITY PREDICTION USING IBM WATSON

IOMP PROJECT REPORT

Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD

In partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

THAVISHI VAMSHI	**19UK1A05M4**
ALUVALA SAI PRIYA	**20UK5A0501**
ERRABELLY SAI CHANDU	**19UK1A05M3**
GAMINENI AJAY	**18UK1A05K0**

Under the esteemed guidance of

Ms .V.MADHAVI

(Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VAAGDEVI ENGINEERING COLLEGE

(Affiliated to JNTUH, Hyderabad)

Bollikunta, Warangal – 506005

. (2019-2023)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VAAGDEVI ENGINEERING COLLEGE

BOLLIKUNTA, WARANGAL – 506005

. **(2019-2023)**

CERTIFICATE OF COMPLETION

IOMP PROJECT

This is to certify that the IOMP Project entitled **"WILD PLANTS EDIBILITY PREDICTION USING IBM WATSON STUDIO"** being submitted by **T. VAMSHI (H.NO:19UK1A05M4), A. SAIPRIYA (H.NO:20UK5A0501), E.SAICHANDHU (H.NO:19UK1A05M3), G.AJAY (H.NO.18UK1A05K0)** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** to **Jawaharlal Nehru Technological University Hyderabad** during the academic year **2022-23**, is a record of work carried out by them under the guidance and supervision.

Project Guide	**Head of the Department**
Ms .V.MADHAVI	**Dr. R. Naveen Kumar**
(Assistant Professor)	(Professor)

ACKNOWLEDGEMENT

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved **Dr.P.PRASAD RAO**, Principal, Vaagdevi Engineering College for makingus available all the required assistance and for his support and inspiration to carry out thisIOMP Project in the institute.

We extend our heartfelt thanks to **Dr.R.NAVEEN KUMAR**, Head of the Department of CSE, Vaagdevi Engineering College for providing us necessary infrastructure and there by giving us freedom to carry out the IOMP Project.

We express heartfelt thanks to Smart Bridge Educational Services Private Limited, for their constant supervision as well as for providing necessary information regarding the UG Project Phase-1 and for their support in completing the IOMP Project .

We express heartfelt thanks to the guide, **Ms.V.MADHAVI** (Assistant professor), Department of CSE for her constant support and giving necessary guidance for completion of this IOMP Project .

Finally, we express our sincere thanks and gratitude to my family members, friends for their encouragement and outpouring their knowledge and experience throughout the thesis.

T. VAMSHI	**19UK1A05M4**
A.SAI PRIYA	**20UK5A0501**
E.SAI CHANDU	**19UK1A05M3**
G. AJAY	**18UK1A05K0**

ABSTRACT

Plant identification based on deep learning received many attention and effort from the research community with many promising results. It becomes an active trend in the recent years. We apply Convolutional Neural Network (CNN) to recognize Vietnamese medicinal plant images in this paper. Different frameworks are evaluated such as: VGG16, Resnet50, InceptionV3, DenseNet121, Xception and MobileNet. The highest accuracy reached by Xception with 88.26%. We might think this approach will greatly contribute to the discovery and conservation of valuable medicinal plants.

Wild edible plants (WEPs) can be defined as native species that grow and reproduce naturally in their natural habitat without being cultivated. Humans have gathered WEPs since ancient times, and they have become part of the human diet and traditional food systems. WEPs still play an important role when food crops are scarce, ensuring food sovereignty and food security, and they potentially contribute to well-being in vulnerable households.

WEPs can also be central to efforts to empower local market actors and reduce the distance between consumers and producers, thereby diminishing the overreliance on globalized value chains [1]. Although the current global food system is capable of providing enough food for mankind, many still experience hunger or do not have access to a nutritious diet.

On the other hand, the increased consumptionof highly processed foods can negatively affect human health. Malnutrition (including over and undernutrition) is considered to be, in addition to climate change, a global threat, indicating an urgent need for a healthier and more sustainable food system.

WEPs can therefore play an important role as an essential component of people’s diets in some regions of the world and provide greater dietary diversity for those who rely on them. In some cases, food plants are also eaten for their health-giving properties, and many species are commonly used as herbal medicines in folk phytotherapy for the treatment of several ailments [2]. Due to their clearly positive influence on health, they are often identified as functional foods, thanks to their higher contents of vitamins, phenols, flavonoids, antioxidants, microelements, and fiber than in cultivated crops.

Wild plants are also perceived as a healthy alternative to cultivated vegetables that might be rich in pesticides and other chemicals. Therefore, wild species may have great potential as sources of unusual colors and flavors, bioactive compounds, and of dietary supplements.

TABLE OF CONTENTS

.

1.INTRODUCTION

1.1 Overview

Agriculture plays a critical role in the global economy. Unfortunately, it is currently pressured by a growing population that requires immediate attention. When technology was first integrated into agriculture more than one century ago, with the first tractor in 1913 (Santos et al., 2020), there was a large increase in food production. Nowadays, mechanical technology has dramatically impacted agriculture, improving its efficiency (Schmitz and Moss, 2015). However, it may not be enough to sustain the increasing global population predicted to reach 9.6 billion by 2050, increasing by approximately 1.8 billion people. This massive increase will require an immense demand for new food (Gikunda and Jouandeau, 2019), making further agricultural advances essential. Several studies on a concept known as smart farming use a combination of technologies, such as the Internet of Things (IoT), Big Data, and robotics, and plays a crucial step in increasing the sustainability and reliability of food production while simultaneously reducing its environmental impact (Walter et al., 2017). This modern approach assists in tackling the key issues facing agriculture and expands future agricultural advances. It focuses on DL tools, specifically CNNs, for identifying patterns in images, which have now become fundamental in agriculture (Liakos et al., 2018), aiding in the identification of plants and diseases through image recognition. However, the applications of smart farming are limited. It solely focuses on existing farms and currently consumed crops, despite the existence of over 50,000 edible plants worldwide, with only 15 of them providing 90% of the world's food energy intake (National Geographic Headquarters, Undated). Examining other edible flora and considering them as a resource for daily consumption can prove beneficial, assisting in the limitations of future food provisions. Furthermore, with the continued decline in the number of botany students (Drea, 2011; Lauer, 2015), the world's understanding of plants may diminish, demanding the need for new methods to understand the plants around us.

This paper presents a method to efficiently identify edible plants in the wild, using CNN architectures. The purpose of this project is to expand the use of DL within horticulture, enabling inexperienced individuals the ability to swiftly recognise wild edible plants within the vast expanse of flora in the world. Opening the opportunity for scientists and individuals interested in horticulture to extract, experiment, and reproduce new types of edible vegetation. Additionally, encouraging future research that concentrates on highlighting and obtaining a deeper understanding of plants.

Classifying plants can be challenging due to the wide variety of similarities between them. What distinguishes plants from one another are their features, such as colour, shape, style, and unique properties, e.g. ground ivy has whiskers, and coneflowers have cones on top of their petals. For botany experts, the difficulty with classifying plants lies in finding their distinct features. Unfortunately, this is the same for CNNs, except they identify the features through numeric pixel values.

1.2 Purpose

With the world population continuing to increase, additional food sources are required.This project aimed to successfully classify different classes of wild edible plants and understand each of their distinct features, allowing easy identification within a large expanse of vegetation. Three CNN architectures were implemented to investigate the unique elements of each plant. The process to achieve this can be divided into multiple objectives.

Objective 1: maximise each model's classification accuracy by creating a dataset containing 35 classes of wild edible plants, where each category has a minimum of 400 images per one. Additionally, employ data augmentation techniques onto the images, increasing the diversity of the dataset.

Objective 2: examine, select, and perform classification with three different CNN architectures and apply transfer learning to each one, bolstering the classification performance to achieve an accuracy of over 80%.

2.LITERATURE SURVEY

2.1 Existing Problems

The global population continues to increase daily, causing an expected increase in food demand, where an estimate of 70% more food needs to be produced by 2050 (Conijn et al., 2011). Unfortunately, the resolution of this food increase requires more than farming additional land for increased fresh produce. With the challenges of resource scarcity and climate change, addressing this food demand becomes extremely difficult. Place et al. (2013) explains that resource scarcity requires the efficient and sustainable use of natural resources to ensure food security. They explain that successfully mitigating resource scarcity requires available resources to be optimized through its availability, accessibility, utilization, and stability. Additionally, it requires a need for new technologies that assist in increasing physical production and accounting for the sustainable use of resources.

In a report by Adams et al. (1998), they write about the effects of global climate change on agriculture and explain that climate is a critical determinant of agricultural productivity, causing it to affect food production by influencing crops and livestock. Some of the core factors produced by climate change include temperature changes,precipitation, droughts, floods, windstorms, crop and livestock pests, and soil erosion, varying in frequency and severity. Moreover, Aydinalp and Cresser (2008) express the cause of climate change to be greenhouse gases that get released into the atmosphere, where today's agricultural facilities contribute to approximately 20% of the annual increase in greenhouse gas emissions through carbon dioxide (CO_2), methane (CH_4) and nitrous oxide (N_2O). With recent developments in DL, agriculture will continue to thrive and alleviate a global food crisis.

2.2 Proposed Solution

In recent years, the field of DL has taken the world by storm and has been implemented into various industries, predominantly scientific fields. Some of these fields include bioinformatics (Li et al., 2019; Min et al., 2017), biochemistry (Cova and Paris, 2019; Richardson et al., 2016), medicine (Ching et al., 2018; Esteva et al., 2019), food security (Mohanty et al., 2016; Ramcharan et al., 2017) and robotics (Pierson and Gashler, 2017; Sünderhauf et al., 2018). DL is a sub-field of Machine Learning (ML) that focuses on the creation of computer models (neural networks) that can learn without being strictly programmed (LeCun et al., 2015). When considering the factors in food production, it is no surprise that agriculture requires multiple disciplines to be working in tandem to create an effective yield of crops. In a report written by Santos et al. (2020), they review 43 papers, where they discover a minimum of 14 domains that use DL in agriculture. Some of the most popular are disease identification, plant recognition, and land cover.

3. THEORETICAL ANALYSIS

3.1 Block Diagram

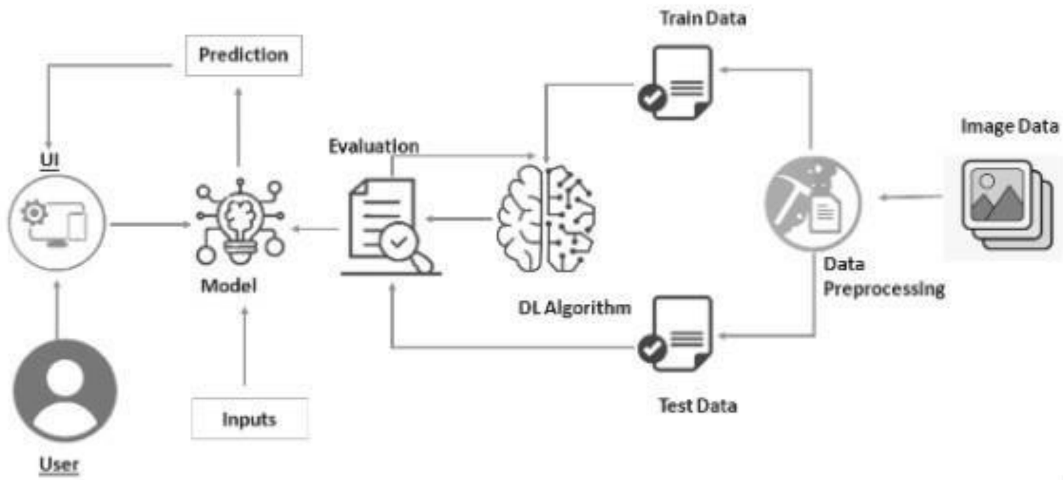

Fig 1: Diagrammatic Overview of the Project

3.2 Hardware and software Requirements

The artefact created utilises the Python programming language, specifically version 3.9.2, and has its functionality extended with the assistance of 8 packages (libraries). While the latest Python version is recommended, it is possible to run the artefact with a minimum of version 3.7. However, it is critical to ensure that the libraries installed utilise the same versions described in this section. The 8 libraries used include Flask, NumPy, gevent, Tensorflow, Werkzeug, Pandas, Scikitlearn, and Scikit-plot.

Another core piece of software used for creating the artefact is virtual environments. When developing software, it is common for developers to have packages and libraries already set up with the latest package versions. Unfortunately, it can be timeconsuming and troublesome to switch between package versions when working on multiple projects. However, virtual environments help to mitigate this issue by preventing conflicting package versions. Anaconda provides a platform to manage Python environments and enables accessibility through a Python kernel, integrated seamlessly with Jupyter Notebooks.

Name	Description	Version
Flask	Flask is a micro-framework in Python which is extensively used to deploy ML models on the web	2.0.2
NumPy	A package used for scientific computing, providing multidimensional array objects and the ability to perform mathematical computations on them.	1.18.5
Gevent	Gevent 1.3 is an important update for performance, debugging and monitoring, and platform support. It introduces an (optional) libuv loop implementation and supports PyPy on Windows.	21.8.0
Tensorflow	TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.	2.3.0
Werkzeug	Werkzeug is a comprehensive WSGI web application library. It began as a simple collection of various utilities for WSGI applications and has become one of the most advanced WSGI utility libraries.	2.0.2
Pandas	A package dedicated to data analysis that uses dataframe objects to view and manipulate data within a tabular format. Required for presenting the model results in a tabular format.	1.2.2
Scikitlearn	A package built on top of NumPy, SciPy, and Matplotlib that provides functionality for efficiently creating ML models and calculating their performance. Required for creating confusion matrices.	0.2.4
Scikit-plot	An extension of Scikit-learn specific to plotting performance metrics. Required for creating the confusion matrix and ROC curve plots.	0.3.7

Table 1: Software required in the project

CNNs can take a long time to train and tune when only using a CPU. To increase this, NIVIDA has a parallel computing toolkit that allows the transfer of data onto GPUs, called CUDA. Fortunately, PyTorch has CUDA functionality built into its library that automatically keeps track of the available GPUs on a system and the tensors allocated to them. When experimenting with different CUDA versions, CUDA toolkit version 10.1 proved fundamental and is required to run the artefact within this project to minimise the training, tuning and evaluation computation speed.

The system hardware used to create the artefact is as follows:

– Intel Core i7-4790k CPU 4.00GHz

– 16GB of RAM

– NVIDIA GeForce GTX 970 (4GB) GPU

– Windows 11 Operating System

It is recommended, but not required, to have similar hardware specifications to run the artefact to maximise its efficiency. If a GPU is not available, it is possible to run the artefact on a typical CPU. However, the tuning time taken to complete the 36 model variants on a GPU took approximately two days. Using a CPU, the time taken will increase by a minimum of 4x as long.

4.EXPERIMENTAL INVESTIGATIONS

4.1 Analysis of the Project

4.1.1 Import Libraries and Initialize The Model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add () method.

4.1.2 Add CNN Layers

We will be adding three layers for CNN

- Convolution layer
- Pooling layer
- Flattening layer
- Full Connection

4.1.2.1 Adding Convolution Layer

In the convolution2D function we given arguments like, 64,(3,3), that means we are applying 64 filters of 3x3 matrix filter, and input_shape is the input image shape with rgb, here 128x128 is the size and 3 represent the channel, rgb color images.

And Activation function defines the output of input or set of inputs or in other terms defines node of the output of node that is given in inputs. They basically decide to deactivate neurons or activate them to get the desired output.

4.1.2.2 Adding Max Pooling Layer

Pooling reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer. It keeps only the necessary details. Pooling is a technique in CNN which helps us to avoid over fitting of data, spatial invariance and distortion. After applying max pooling we will get another feature map called Pooled Feature Map.

4.1.2.3 Adding Flatten Layers

Now the pooled feature map from the pooling layer will be converted into a one single dimension matrix or map, where each pixel in one single column, nothing but flattening. Flattening layer converts multi dimension matrix to one single dimension layer.

4.1.3 Adding Dense Layers

4.1.3.1 Adding Dense Layers

The name suggests that layers are fully connected (dense) by the neurons in a network layer. Each neuron in a layer receives an input from all the neurons present in the previous layer. Dense is used to add the layers.

4.1.3.2 Adding Hidden layers

This step is to add a dense layer (hidden layer). We flatten the feature map and convert it into a vector or single dimensional array in the Flatten layer. This vector array is feed it as an input to the neural network and applies an activation function, such as sigmoid or other, and returns the output.

Understanding the model is very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

4.1.4 Configuring The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to training phase. Loss function is used to find error or deviation in the learning process. Keras requires loss function during model compilation process.

- Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Here we are using adam optimizer

- Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process

4.1.5 Train And Save The Model

4.1.5.1 Train the Model

Now let us train our model with our image dataset. fit_generator functions used to train a deep learning neural network.

- **steps_per_epoch:** It specifies the total number of steps taken from the generator as soon as one epoch is finished and next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your training folder divided by the batch size.

- **Epochs:** an integer and number of epochs we want to train our model for.

- **Validation_data** can be either an inputs and targets list a generator an inputs, targets, and sample_weights list which can be used to evaluate. The loss and metrics for any model after any epoch has ended.

- **Validation_steps:** Only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

4.1.5.2 Save the Model

The model is saved with .h5 extension as follows:

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

4.1.6 Test The Model

The last and final step is to make use of our saved model to do predictions. For that we have a class in keras called load_model. Load_model is used to load our saved model h5 file (edible- non.h5).

4.1.7 Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has uploads an image . The uploaded image is given to the saved model and prediction is showcased on the UI.

This section has the following tasks:

- Building HTML Pages
- Building server-side script

4.1.8 Build Python Code

We will be using python for server side scripting. Let's see step by step process for writing backend code.

4.1.8.1 Importing Libraries

Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of current module (_name_) as argument Pickle library to load the model file.

4.1.8.2 Routing to the html Page

Here we will be using declared constructor to route to the html page which we have created earlier.

4.1.8.3 Showcasing prediction on UI

When the image is uploaded,it predicts the category of uploaded the image is either 'Asparagus_edible','Blue Vervain_edible','Cattail_edible','Chicory_edible_non edible', 'Fireweed_edible_non edible', 'green castor bean_non edible'. If the image predicts value as 0, then it is displayed as "Left Bundle Branch". Similarly, if the predicted value is 1, it displays "Normal" as output and so on.

4.1.8.4 Main Function

This is used to run the application in local host.

4.1.9 Run The App

- Open anaconda prompt from start menu.
- Navigate to the folder where your app.py resides.
- Now type "python app.py" command.
- It will show the local host where your app is running on http://127.0.0.1.5000/
- Copy that local host URL and open that URL in browser. It does navigate you to the where you can view your web page.
- Enter the values, click on predict button and see the result/prediction on web page.

5.FLOWCHART

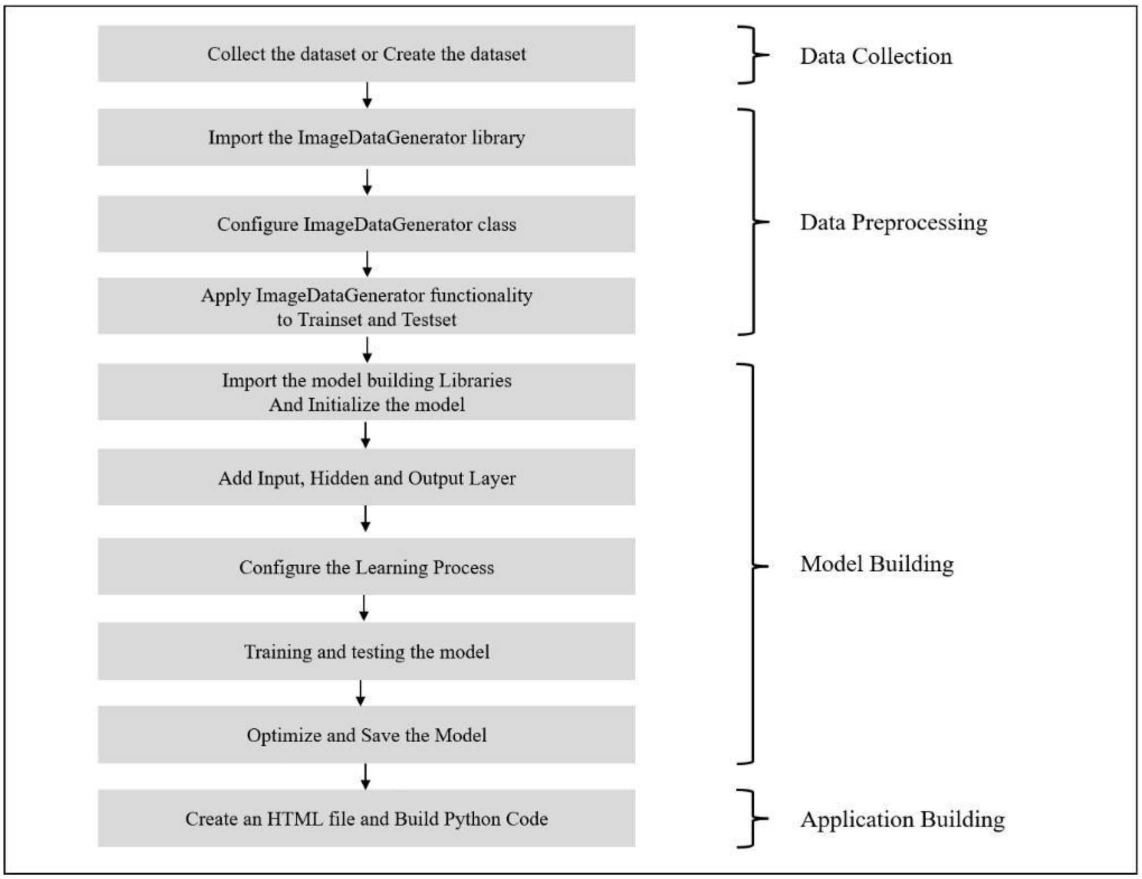

Fig 2: Flowchart showing the control flow of the solution

6.RESULT

6.1 Final findings (Output) of the project

```
In [14]: # view the structure of the model
         model.summary()

Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 8)                 200712
=================================================================
Total params: 14,915,400
Trainable params: 200,712
Non-trainable params: 14,714,688
_________________________________________________________________
```

Fig 3: Structure of the Model

```
In [16]: # fit the model
         # Run the cell. It will take some time to execute
         r = model.fit_generator(
           training_set,
           validation_data=test_set,
           epochs=10,
           steps_per_epoch=len(training_set),
           validation_steps=len(test_set)
         )

C:\Users\HP\anaconda3\envs\tf1\lib\site-packages\tensorflow\python\keras\engine\training.py:1844: UserWarning: `Model.fit_generator` is deprecated a
nd will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/10
20/20 [==============================] - 106s 5s/step - loss: 2.3926 - accuracy: 0.2255 - val_loss: 1.5907 - val_accuracy: 0.4349
Epoch 2/10
20/20 [==============================] - 111s 6s/step - loss: 1.2632 - accuracy: 0.5757 - val_loss: 1.1966 - val_accuracy: 0.5948
Epoch 3/10
20/20 [==============================] - 190s 10s/step - loss: 0.7801 - accuracy: 0.7463 - val_loss: 1.1466 - val_accuracy: 0.5948
Epoch 4/10
20/20 [==============================] - 125s 6s/step - loss: 0.6110 - accuracy: 0.8346 - val_loss: 1.0180 - val_accuracy: 0.6543
Epoch 5/10
20/20 [==============================] - 117s 6s/step - loss: 0.4902 - accuracy: 0.8536 - val_loss: 0.9159 - val_accuracy: 0.6654
Epoch 6/10
20/20 [==============================] - 119s 6s/step - loss: 0.3820 - accuracy: 0.8943 - val_loss: 0.8916 - val_accuracy: 0.6914
Epoch 7/10
20/20 [==============================] - 122s 6s/step - loss: 0.3773 - accuracy: 0.9026 - val_loss: 0.9658 - val_accuracy: 0.6394
Epoch 8/10
20/20 [==============================] - 125s 6s/step - loss: 0.2706 - accuracy: 0.9369 - val_loss: 0.9008 - val_accuracy: 0.6729
Epoch 9/10
20/20 [==============================] - 131s 7s/step - loss: 0.2278 - accuracy: 0.9502 - val_loss: 0.8269 - val_accuracy: 0.6952
Epoch 10/10
20/20 [==============================] - 130s 7s/step - loss: 0.1871 - accuracy: 0.9670 - val_loss: 0.8079 - val_accuracy: 0.7175
```

Fig 4: fit the Model

In [26]:

```
# Training the model



#res = model.fit_generator(x_train,steps_per_epoch=14,validation_steps=5,epochs=40,validation_data=x_test)
# Training the model
history = model.fit_generator(x_train,steps_per_epoch=624//32,
                              validation_data=x_test,
                              epochs=100,validation_steps=269//32)
#history = model.fit_generator(x_train,steps_per_epoch=14,validation_data=x_test,epochs=40,validation_steps=5)
```

```
C:\Users\HABIN\Anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py:1940: UserWarning: `Model.fit_generator` is deprecated and wil
l be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/100
19/19 [==============================] - 19s 937ms/step - loss: 2.0787 - accuracy: 0.1841 - val_loss: 2.0795 - val_accuracy: 0.1250
Epoch 2/100
19/19 [==============================] - 12s 600ms/step - loss: 2.0734 - accuracy: 0.1875 - val_loss: 2.0958 - val_accuracy: 0.1289
Epoch 3/100
19/19 [==============================] - 10s 558ms/step - loss: 2.0608 - accuracy: 0.1909 - val_loss: 2.0743 - val_accuracy: 0.1367
Epoch 4/100
19/19 [==============================] - 11s 563ms/step - loss: 2.0595 - accuracy: 0.1875 - val_loss: 2.0534 - val_accuracy: 0.1367
Epoch 5/100
19/19 [==============================] - 10s 524ms/step - loss: 2.0343 - accuracy: 0.1943 - val_loss: 2.1383 - val_accuracy: 0.1289
Epoch 6/100
19/19 [==============================] - 10s 513ms/step - loss: 2.0049 - accuracy: 0.2111 - val_loss: 2.1873 - val_accuracy: 0.1875
Epoch 7/100
19/19 [==============================] - 10s 544ms/step - loss: 1.9797 - accuracy: 0.2179 - val_loss: 2.0045 - val_accuracy: 0.2031
Epoch 8/100
19/19 [==============================] - 10s 535ms/step - loss: 1.9355 - accuracy: 0.2162 - val_loss: 1.9810 - val_accuracy: 0.2227
Epoch 9/100
19/19 [==============================] - 10s 533ms/step - loss: 1.9255 - accuracy: 0.2230 - val_loss: 1.9342 - val_accuracy: 0.2383
Epoch 10/100
19/19 [==============================] - 11s 585ms/step - loss: 1.8957 - accuracy: 0.2568 - val_loss: 1.9158 - val_accuracy: 0.2461
Epoch 11/100
19/19 [==============================] - 10s 506ms/step - loss: 1.9237 - accuracy: 0.2669 - val_loss: 1.9252 - val_accuracy: 0.2344
Epoch 12/100
19/19 [==============================] - 10s 557ms/step - loss: 1.8751 - accuracy: 0.2466 - val_loss: 1.9045 - val_accuracy: 0.2461
Epoch 13/100
19/19 [==============================] - 10s 518ms/step - loss: 1.8728 - accuracy: 0.2365 - val_loss: 1.8732 - val_accuracy: 0.2617
Epoch 14/100
19/19 [==============================] - 9s 490ms/step - loss: 1.8546 - accuracy: 0.2804 - val_loss: 1.8821 - val_accuracy: 0.2539
Epoch 15/100
19/19 [==============================] - 11s 565ms/step - loss: 1.8419 - accuracy: 0.2838 - val_loss: 1.9056 - val_accuracy: 0.2578
Epoch 16/100
19/19 [==============================] - 10s 536ms/step - loss: 1.8113 - accuracy: 0.3057 - val_loss: 1.8719 - val_accuracy: 0.2617
Epoch 17/100
19/19 [==============================] - 11s 565ms/step - loss: 1.7720 - accuracy: 0.3328 - val_loss: 2.0896 - val_accuracy: 0.2422
Epoch 18/100
19/19 [==============================] - 10s 520ms/step - loss: 1.8619 - accuracy: 0.2686 - val_loss: 1.9870 - val_accuracy: 0.2461
Epoch 19/100
19/19 [==============================] - 10s 551ms/step - loss: 1.8377 - accuracy: 0.2956 - val_loss: 1.8999 - val_accuracy: 0.2852
Epoch 20/100
19/19 [==============================] - 10s 534ms/step - loss: 1.7979 - accuracy: 0.3091 - val_loss: 1.8486 - val_accuracy: 0.2969
Epoch 21/100
19/19 [==============================] - 10s 527ms/step - loss: 1.7666 - accuracy: 0.3159 - val_loss: 1.8865 - val_accuracy: 0.2695
Epoch 22/100
19/19 [==============================] - 10s 508ms/step - loss: 1.7866 - accuracy: 0.3074 - val_loss: 1.8390 - val_accuracy: 0.3086
Epoch 23/100
19/19 [==============================] - 9s 471ms/step - loss: 1.7355 - accuracy: 0.3260 - val_loss: 1.8860 - val_accuracy: 0.3047
Epoch 24/100
19/19 [==============================] - 10s 516ms/step - loss: 1.6874 - accuracy: 0.3328 - val_loss: 1.8879 - val_accuracy: 0.3125
Epoch 25/100
19/19 [==============================] - 10s 522ms/step - loss: 1.6850 - accuracy: 0.3470 - val_loss: 1.8946 - val_accuracy: 0.3242
Epoch 26/100
19/19 [==============================] - 10s 505ms/step - loss: 1.6730 - accuracy: 0.3615 - val_loss: 1.8329 - val_accuracy: 0.3047
Epoch 27/100
19/19 [==============================] - 10s 537ms/step - loss: 1.6531 - accuracy: 0.3801 - val_loss: 1.8303 - val_accuracy: 0.3398
Epoch 28/100
19/19 [==============================] - 10s 537ms/step - loss: 1.6526 - accuracy: 0.3564 - val_loss: 1.8013 - val_accuracy: 0.3320
Epoch 29/100
19/19 [==============================] - 10s 526ms/step - loss: 1.6525 - accuracy: 0.3598 - val_loss: 1.8096 - val_accuracy: 0.2969
Epoch 30/100
19/19 [==============================] - 10s 541ms/step - loss: 1.7077 - accuracy: 0.3553 - val_loss: 1.7425 - val_accuracy: 0.3164
Epoch 31/100
19/19 [==============================] - 11s 567ms/step - loss: 1.6670 - accuracy: 0.3514 - val_loss: 1.7974 - val_accuracy: 0.3320
Epoch 32/100
19/19 [==============================] - 10s 554ms/step - loss: 1.6531 - accuracy: 0.3581 - val_loss: 1.7740 - val_accuracy: 0.3516
Epoch 33/100
19/19 [==============================] - 10s 533ms/step - loss: 1.6236 - accuracy: 0.3834 - val_loss: 1.8221 - val_accuracy: 0.3320
Epoch 34/100
19/19 [==============================] - 9s 497ms/step - loss: 1.6438 - accuracy: 0.3598 - val_loss: 1.7963 - val_accuracy: 0.3242
Epoch 35/100
19/19 [==============================] - 10s 502ms/step - loss: 1.5962 - accuracy: 0.3868 - val_loss: 1.8251 - val_accuracy: 0.3320
Epoch 36/100
19/19 [==============================] - 11s 578ms/step - loss: 1.6053 - accuracy: 0.3598 - val_loss: 1.8691 - val_accuracy: 0.2930
Epoch 37/100
19/19 [==============================] - 10s 544ms/step - loss: 1.6785 - accuracy: 0.3564 - val_loss: 1.7154 - val_accuracy: 0.3320
Epoch 38/100
19/19 [==============================] - 9s 483ms/step - loss: 1.6276 - accuracy: 0.3868 - val_loss: 1.7663 - val_accuracy: 0.3438
Epoch 39/100
19/19 [==============================] - 9s 497ms/step - loss: 1.5788 - accuracy: 0.3699 - val_loss: 1.8653 - val_accuracy: 0.3477
Epoch 40/100
19/19 [==============================] - 11s 560ms/step - loss: 1.5391 - accuracy: 0.4037 - val_loss: 1.7509 - val_accuracy: 0.3633
Epoch 41/100
19/19 [==============================] - 10s 518ms/step - loss: 1.5600 - accuracy: 0.3936 - val_loss: 1.8432 - val_accuracy: 0.3555
Epoch 42/100
19/19 [==============================] - 10s 520ms/step - loss: 1.6172 - accuracy: 0.3936 - val_loss: 1.7726 - val_accuracy: 0.3281
Epoch 43/100
19/19 [==============================] - 10s 553ms/step - loss: 1.6089 - accuracy: 0.3586 - val_loss: 1.8179 - val_accuracy: 0.3203
Epoch 44/100
19/19 [==============================] - 10s 521ms/step - loss: 1.5488 - accuracy: 0.4206 - val_loss: 1.8842 - val_accuracy: 0.3164
```

```
Epoch 45/100
19/19 [==============================] - 10s 535ms/step - loss: 1.5634 - accuracy: 0.4105 - val_loss: 1.7726 - val_accuracy: 0.3398
Epoch 46/100
19/19 [==============================] - 10s 531ms/step - loss: 1.5644 - accuracy: 0.4071 - val_loss: 1.7721 - val_accuracy: 0.3594
Epoch 47/100
19/19 [==============================] - 10s 538ms/step - loss: 1.5994 - accuracy: 0.3919 - val_loss: 1.7299 - val_accuracy: 0.3516
Epoch 48/100
19/19 [==============================] - 11s 571ms/step - loss: 1.5758 - accuracy: 0.3885 - val_loss: 1.7787 - val_accuracy: 0.3398
Epoch 49/100
19/19 [==============================] - 9s 493ms/step - loss: 1.5260 - accuracy: 0.4155 - val_loss: 1.7683 - val_accuracy: 0.3633
Epoch 50/100
19/19 [==============================] - 10s 532ms/step - loss: 1.5624 - accuracy: 0.3953 - val_loss: 1.7842 - val_accuracy: 0.3281
Epoch 51/100
19/19 [==============================] - 11s 588ms/step - loss: 1.5086 - accuracy: 0.4341 - val_loss: 1.7938 - val_accuracy: 0.3477
Epoch 52/100
19/19 [==============================] - 10s 521ms/step - loss: 1.5180 - accuracy: 0.4189 - val_loss: 1.9159 - val_accuracy: 0.3555
Epoch 53/100
19/19 [==============================] - 11s 560ms/step - loss: 1.5347 - accuracy: 0.4139 - val_loss: 1.7445 - val_accuracy: 0.3711
Epoch 54/100
19/19 [==============================] - 11s 576ms/step - loss: 1.5310 - accuracy: 0.4139 - val_loss: 1.6891 - val_accuracy: 0.3711
Epoch 55/100
19/19 [==============================] - 11s 558ms/step - loss: 1.5003 - accuracy: 0.4071 - val_loss: 1.7975 - val_accuracy: 0.3203
Epoch 56/100
19/19 [==============================] - 10s 505ms/step - loss: 1.5209 - accuracy: 0.4020 - val_loss: 1.7096 - val_accuracy: 0.3828
Epoch 57/100
19/19 [==============================] - 10s 535ms/step - loss: 1.4565 - accuracy: 0.4206 - val_loss: 1.8401 - val_accuracy: 0.3516
Epoch 58/100
19/19 [==============================] - 11s 565ms/step - loss: 1.5149 - accuracy: 0.3818 - val_loss: 1.7710 - val_accuracy: 0.3633
Epoch 59/100
19/19 [==============================] - 10s 507ms/step - loss: 1.4703 - accuracy: 0.4358 - val_loss: 1.7608 - val_accuracy: 0.3984
Epoch 60/100
19/19 [==============================] - 10s 523ms/step - loss: 1.4397 - accuracy: 0.4527 - val_loss: 1.6525 - val_accuracy: 0.3906
Epoch 61/100
19/19 [==============================] - 9s 483ms/step - loss: 1.4249 - accuracy: 0.4561 - val_loss: 1.6545 - val_accuracy: 0.3672
Epoch 62/100
19/19 [==============================] - 11s 557ms/step - loss: 1.4502 - accuracy: 0.4223 - val_loss: 1.6365 - val_accuracy: 0.3828
Epoch 63/100
19/19 [==============================] - 10s 522ms/step - loss: 1.4244 - accuracy: 0.4510 - val_loss: 1.8017 - val_accuracy: 0.3672
Epoch 64/100
19/19 [==============================] - 10s 541ms/step - loss: 1.4707 - accuracy: 0.4493 - val_loss: 1.6675 - val_accuracy: 0.4062
Epoch 65/100
19/19 [==============================] - 11s 552ms/step - loss: 1.4647 - accuracy: 0.4274 - val_loss: 1.5945 - val_accuracy: 0.3984
Epoch 66/100
19/19 [==============================] - 10s 538ms/step - loss: 1.4334 - accuracy: 0.4459 - val_loss: 1.7059 - val_accuracy: 0.3828
Epoch 67/100
19/19 [==============================] - 10s 541ms/step - loss: 1.4803 - accuracy: 0.4307 - val_loss: 1.6762 - val_accuracy: 0.3672
Epoch 68/100
19/19 [==============================] - 10s 558ms/step - loss: 1.4583 - accuracy: 0.4527 - val_loss: 1.7272 - val_accuracy: 0.3828
Epoch 69/100
19/19 [==============================] - 10s 517ms/step - loss: 1.4565 - accuracy: 0.4510 - val_loss: 1.6551 - val_accuracy: 0.3359
Epoch 70/100
19/19 [==============================] - 10s 510ms/step - loss: 1.4170 - accuracy: 0.4392 - val_loss: 1.9587 - val_accuracy: 0.3320
Epoch 71/100
19/19 [==============================] - 10s 531ms/step - loss: 1.5229 - accuracy: 0.4324 - val_loss: 1.6856 - val_accuracy: 0.4141
Epoch 72/100
19/19 [==============================] - 10s 540ms/step - loss: 1.4197 - accuracy: 0.4696 - val_loss: 1.6802 - val_accuracy: 0.3984
Epoch 73/100
19/19 [==============================] - 11s 582ms/step - loss: 1.4381 - accuracy: 0.4426 - val_loss: 1.7262 - val_accuracy: 0.4141
Epoch 74/100
19/19 [==============================] - 10s 542ms/step - loss: 1.3929 - accuracy: 0.4527 - val_loss: 1.6343 - val_accuracy: 0.4062
Epoch 75/100
19/19 [==============================] - 11s 563ms/step - loss: 1.3927 - accuracy: 0.4510 - val_loss: 1.7079 - val_accuracy: 0.3906
Epoch 76/100
19/19 [==============================] - 11s 572ms/step - loss: 1.3948 - accuracy: 0.4679 - val_loss: 1.6945 - val_accuracy: 0.3906
Epoch 77/100
19/19 [==============================] - 14s 728ms/step - loss: 1.4013 - accuracy: 0.4662 - val_loss: 1.6718 - val_accuracy: 0.3945
Epoch 78/100
19/19 [==============================] - 11s 572ms/step - loss: 1.3892 - accuracy: 0.4611 - val_loss: 1.6565 - val_accuracy: 0.4375
Epoch 79/100
19/19 [==============================] - 10s 510ms/step - loss: 1.3879 - accuracy: 0.4865 - val_loss: 1.7705 - val_accuracy: 0.3984
Epoch 80/100
19/19 [==============================] - 10s 508ms/step - loss: 1.3679 - accuracy: 0.4696 - val_loss: 1.6171 - val_accuracy: 0.4336
Epoch 81/100
19/19 [==============================] - 8s 449ms/step - loss: 1.3532 - accuracy: 0.4764 - val_loss: 1.6218 - val_accuracy: 0.4062
Epoch 82/100
19/19 [==============================] - 10s 519ms/step - loss: 1.3455 - accuracy: 0.4882 - val_loss: 1.6418 - val_accuracy: 0.4141
Epoch 83/100
19/19 [==============================] - 9s 450ms/step - loss: 1.3969 - accuracy: 0.4628 - val_loss: 1.6023 - val_accuracy: 0.4258
Epoch 84/100
19/19 [==============================] - 9s 495ms/step - loss: 1.3555 - accuracy: 0.4831 - val_loss: 1.7113 - val_accuracy: 0.3984
Epoch 85/100
19/19 [==============================] - 10s 524ms/step - loss: 1.3918 - accuracy: 0.4730 - val_loss: 1.5925 - val_accuracy: 0.4219
Epoch 86/100
19/19 [==============================] - 11s 567ms/step - loss: 1.3731 - accuracy: 0.4730 - val_loss: 1.6356 - val_accuracy: 0.4102
Epoch 87/100
19/19 [==============================] - 10s 510ms/step - loss: 1.3516 - accuracy: 0.4932 - val_loss: 1.6384 - val_accuracy: 0.4414
Epoch 88/100
19/19 [==============================] - 10s 539ms/step - loss: 1.3126 - accuracy: 0.5051 - val_loss: 1.5978 - val_accuracy: 0.4258
Epoch 89/100
19/19 [==============================] - 11s 596ms/step - loss: 1.2752 - accuracy: 0.5049 - val_loss: 1.6498 - val_accuracy: 0.4023
Epoch 90/100
19/19 [==============================] - 10s 552ms/step - loss: 1.3224 - accuracy: 0.5084 - val_loss: 1.6138 - val_accuracy: 0.4180
Epoch 91/100
19/19 [==============================] - 10s 539ms/step - loss: 1.3360 - accuracy: 0.4831 - val_loss: 1.5720 - val_accuracy: 0.4414
Epoch 92/100
19/19 [==============================] - 11s 558ms/step - loss: 1.3001 - accuracy: 0.5084 - val_loss: 1.6118 - val_accuracy: 0.4219
Epoch 93/100
19/19 [==============================] - 11s 559ms/step - loss: 1.3257 - accuracy: 0.4899 - val_loss: 1.5603 - val_accuracy: 0.4453
Epoch 94/100
19/19 [==============================] - 11s 577ms/step - loss: 1.3211 - accuracy: 0.4865 - val_loss: 1.6664 - val_accuracy: 0.4336
Epoch 95/100
19/19 [==============================] - 10s 521ms/step - loss: 1.3334 - accuracy: 0.5084 - val_loss: 1.6031 - val_accuracy: 0.4102
Epoch 96/100
19/19 [==============================] - 10s 539ms/step - loss: 1.3300 - accuracy: 0.4865 - val_loss: 1.6680 - val_accuracy: 0.4375
Epoch 97/100
19/19 [==============================] - 10s 525ms/step - loss: 1.2822 - accuracy: 0.5016 - val_loss: 1.6512 - val_accuracy: 0.4219
Epoch 98/100
19/19 [==============================] - 10s 544ms/step - loss: 1.2562 - accuracy: 0.5068 - val_loss: 1.6787 - val_accuracy: 0.3984
Epoch 99/100
19/19 [==============================] - 11s 574ms/step - loss: 1.2878 - accuracy: 0.5084 - val_loss: 1.6058 - val_accuracy: 0.4570
Epoch 100/100
19/19 [==============================] - 10s 550ms/step - loss: 1.2771 - accuracy: 0.5017 - val_loss: 1.7177 - val_accuracy: 0.4219
```

Fig: Training of the Model

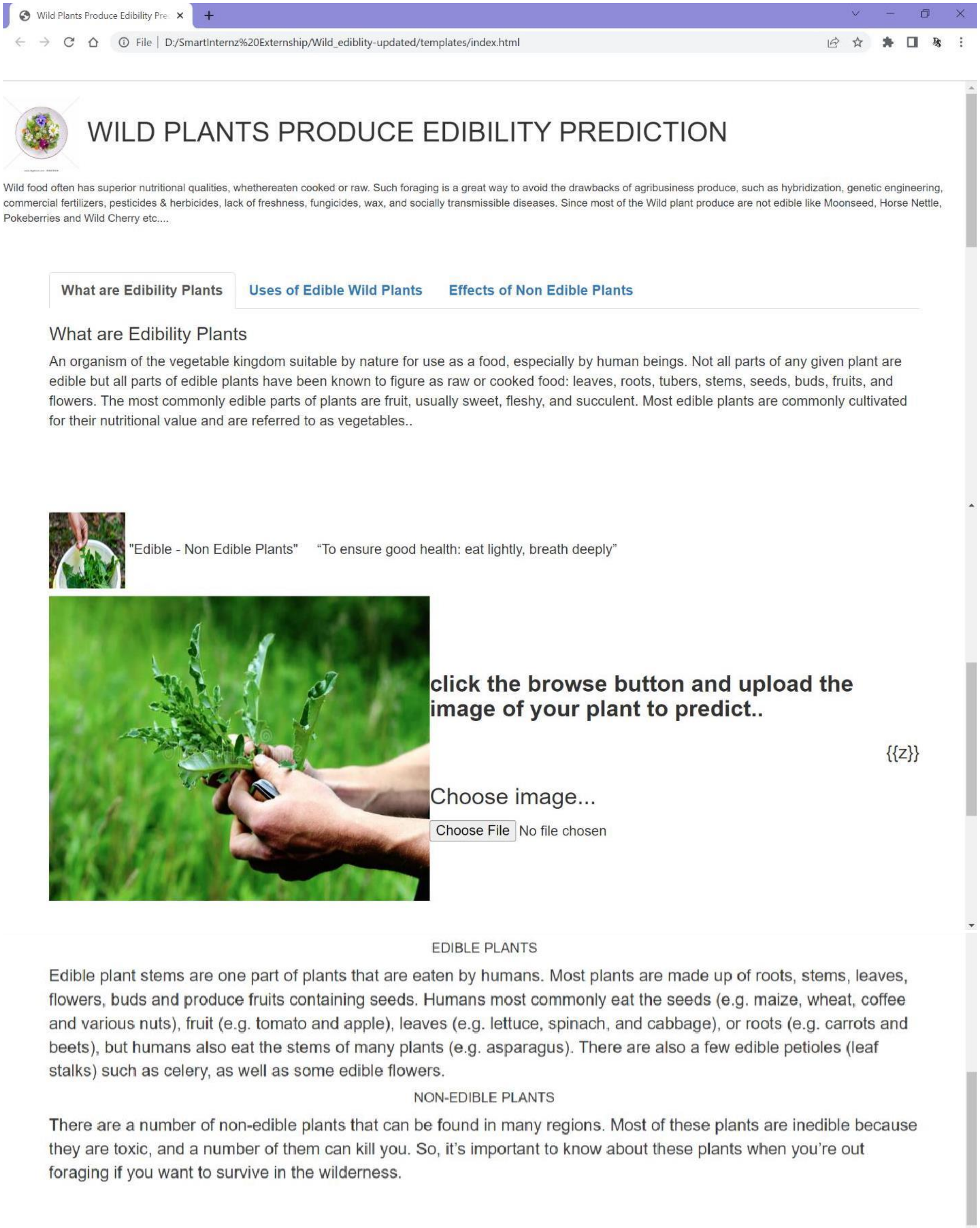

Fig: Screenshots of the webpage

7. APPLICATION

Quantitative ethnoecological analysis of seasonal availability and implication to food security of wild edible plants (WEPs) was conducted in Boosat and Fantalle districts of semiarid east Shewa, Ethiopia from October, 2009 to September, 2010. Semistructured interview, focus group discussions, key informants discussions, seasonal record of fruits abundance were used to collected data on gathering and consumption of WEPs to cope with food shortage and adapt to climate change. Collected data was summarised into frequency tables, graph and qualitatively described under each subtopic. Thirty seven WEPs were identified for use as human food, and livestock feed and other multipurpose uses. About 24.3 % of WEPs were locally marketed, 75.7% were not marketed. All wild fruits were not included in official production system in the study area. It has indicated the underutilized existing potential of WEPs. Wild edible plants were preferred by local people of the study area not only for their food value, but also for their availability during dry seasons and shortage of food, potential for dryland agrobiodiversity and multipurpose to human wellbeing, livestock and environmental services they provide. Pairwise ranking by key informants was in agreement with direct matrices ranking for multiple uses of WEPs. The pairwise ranking, market survey and participant observations, community preference has confirmed the real potential of top seven priority WEPs species for dryland agrobiodiversity and agroforestry. Hence, these WEPs can be potential for dryland agrobiodiversity and agroforestry, to enhance people's livelihoods in semiarid areas. This result can shed light on further research and promotion work on WEPs utilization and management.

8. CONCLUSION

The research within this paper aimed to explore DL techniques to address the food limitations in 2050. Some of the challenges identified include resource scarcity and climate change that are mitigated using disease identification and plant recognition techniques. However, research shows that these techniques alone cannot fully resolve these issues, requiring consideration of alternative methods. One of these methods, explored in this paper, involves identifying wild edible plants in natural environments, providing an opportunity for scientists and researchers the ability to retrieve and examine them as a potential food source.

Using a combination of the Waterfall and Kanban methodologies, and the Jupyter Lab IDE, an artefact was created that involved transfer learning to train and tune three state-of-the-art CNN architectures, GoogLeNet, MobileNet v2, and ResNet-34, on a dataset containing 16,535 images, divided into 35 classes of wild edible plants. When tuning the architectures, seven hyperparameters, consisting of three batch sizes and four sets of hidden node sizes, were used to create 36 models evaluated on the dataset.

Further evaluation revealed that only a small number of plant classes were incorrectly classified, varying per architecture. Nevertheless, the models struggled to correctly classify two main sets of plant types, meadowsweet against cow parsley and common mallow against geranium.
Overall, determining the best model for classifying wild edible plants depends on the users' requirements.

9.FUTURE SCOPE

Wild food plants are increasingly considered a potential source of natural healthy products, it is fundamental to foster biochemical research aimed at documenting their nutritional properties and main bioactive products. The phytochemical and nutritional profiles of the species in question can therefore constitute basic knowledge for food pairing with other ingredients to improve nutritional and/or sensory quality and to find innovative cooking methods, allowing key molecules responsible for functional properties to be enhanced.

In conclusion, wild plants represent a crucial section of the human diet. It is hoped that an increasing amount of scientific research will focus on plant diversity, traditional knowledge, and agricultural studies and will foster bio-conservation strategies and sustainable food production. Biochemical knowledge is of crucial importance to evaluate the health benefits and physiological effects of WEPs in order to develop clinical investigations concerning their mechanisms of action, safety, and efficacy.

10. BIBLIOGRAPHY

Adams, R., Hurd, B., Lenhart, S. and Leary, N. (1998) Effects of global climate change on agriculture: an interpretative review. *Climate Research*, 11(1) 19–30. Available from https://www.intres.com/abstracts/cr/v11/n1/p19-30/ [accessed 19 January 2021].

Affouard, A., Goeau, H., Bonnet, P., Lombardo, J-C. and Joly, A. (2017) *Pl@ntNet app in the era of deep learning.* OpenReview. Available from https://openreview.net/forum?id=HJVJpENFg [accessed 22 January 2021].

Ahmad, M.O., Markkula, J. and Oivo, M. (2013) Kanban in Software Development: A Systematic Literature Review. In: *Software Engineering and Advanced Applications (SEAA)*, Santander, Spain, 4-6 September. IEEE Computer Society, 9–16. Available from https://www.researchgate.net/publication/260739586_Kanban_in_Software_Development_A_Systematic_Literature_Review [accessed 28 January 2021].

Al-Masri, A. (2019) *How Does Back-Propagation in Artificial Neural Networks Work?* Available from https://towardsdatascience.com/how-does-back-propagation-in-artificialneural- networks-work c7cad873ea7 [accessed 2 March 2021].

Ampatzidis, Y., De, L. and Luvisi, A. (2017) iPathology: Robotic Applications and Management of Plants and Plant Diseases. *Sustainability*, 9(6) 1010. Available from https://www.mdpi.com/2071 1050/9/6/1010 [accessed 23 January 2021].

Anaconda (Undated) *Individual Edition - Your data science toolkit.* Anaconda. Available from https://www.anaconda.com/products/individual [accessed 3 February 2021].

Aydinalp, C. and Cresser, M. (2008) The Effects of Global Climate Change on Agriculture. *American-Eurasian J. Agric. Environ. Sci.*, 3. Available from https://www. researchgate.net/publication/238091112_The_Effects_of_Global_Climate_Change_on_Agriculture [accessed 19 January 2021].

Barré, P., Stöver, B., Müller, K. and Steinhage, V. (2017) LeafNet: A computer vision system for automatic plant species identification. *Ecological Informatics*, 40 50–56. Available from http://www.sciencedirect.com/science/article/pii/S1574954116302515 [accessed 23 January 2021].

Bulajic, A., Sambasivam, S. and Stojic, R. (2013) An Effective Development Environment Setup for System and Application Software. *Issues in Informing Science and Information Technology*, 10 037–066. Available from https://www.informingscience.org/ Publications/1795 [accessed 2 February 2021].