BMS COLLEGE OF ENGINEERING, BANGALORE – 560 019

(Autonomous institute, Affiliated to VTU)

Department of Information Science and Engineering

Deep Learning - 20IS6PEDLG

Flower Classification Using CNN

2021 – 2022 – 6TH SEMESTER

Submitted by:

Mohan D - 1BM19IS092

Moksh Jayanth GR - 1BM19IS094

N Prabhu - 1BM19IS096

BMS COLLEGE OF ENGINEERING, BANGALORE -19

(An autonomous institute, affiliated to VTU)

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

2021 – 2022 – EVEN SEMESTER

CERTIFICATE

Certified that Mr. Mohan D bearing USN **1BM19IS092** , Mr.Moksh Jayanth GR bearing USN **1BM19IS094** and Mr.N Prabhu bearing USN **1BM19IS096** of Sixth semester belonging to the Department of Information Science and Engineering had successfully completed AAT as a part of the course Deep Learning [20IS6PEDLG]

Faculty Incharge

Mrs.Rashmi R

BMS COLLEGE OF ENGINEERING, BANGALORE -19

(An autonomous institute, affiliated to VTU)

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING
2021 – 2022 – EVEN SEMESTER

CONTENTS

Course: Deep Learning **Course code: 20IS6PEDLG**

ABSTRACT

The app's functionality is simple and minimal. It uses the concepts of Deep Learning to predict the name of the Flower in the image.

The App was actually inspired by many of the popular Advanced Plant Detection App available on the play store such as *PlantNet*, *LeafSnap*, etc.

These Advanced Plant Detection App also use Concepts of Deep Learning And Computer Vision to Correctly Identify the Plant Species. Training such models requires TeraBytes of labeled data.

Inspired by the following, we built a simple **CNN Classification Model** which detects and identifies the image of the given flower.

INTRODUCTION

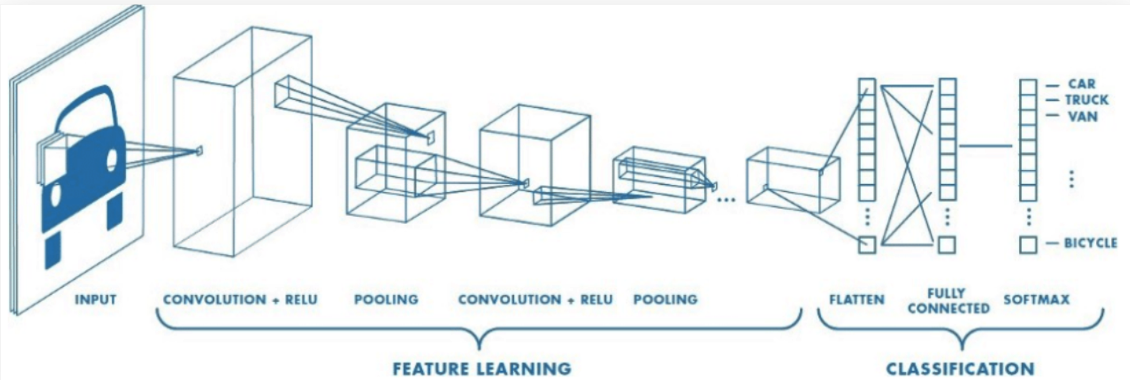


Fig 1.1 The CNN Architecture

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what color each pixel should be.

Apps Such as PlantNet, LeafSnap etc. use the same concept to predict the species of thousands of plants, the model being trained by Terabytes of data.

Our simple Flower Classification Web Application offers limited functionality. It predicts the classes of five flowers namely:

1. Roses
2. Tulips
3. Dandelion
4. Sunflower
5. Daisy

LeafSnap ***PlantNet***

PROBLEM STATEMENT

Problem Statement of our project stands as follows:

- To build a total of 3 Models
- To compare the 3 Models and to deploy the best performing model as Web Application
- To build a **Regular CNN Model** as our first model using over 4000 labeled flower images
- To build our second model and train it by applying the techniques of **Data Augmentation.**
- To build our third model by applying the techniques of **Transfer Learning.**
- To Deploy the best performing Model as a Web Application.

THE COMPLETE CODE IMPLEMENTATION

Flower Classification Using CNN And Transfer Learning

```
import matplotlib.pyplot as plt

import numpy as np

import cv2

import os

import PIL

import tensorflow as tf


from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential
```

We will download the flower dataset from google website and store it locally. In the below call it downloads the zip file (.tgz) in cache_dir which is . meaning the current folder

Load flowers dataset

```
dataset_url =
"https://storage.googleapis.com/download.tensorflow.org/example_images/
flower_photos.tgz"

data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url,
cache_dir='.', untar=True)

data_dir

import pathlib

data_dir = pathlib.Path(data_dir)

'./datasets/flower_photos'

Data_dir

PosixPath('datasets/flower_photos')
```

```python
list(data_dir.glob('*/*.jpg'))[:5]
```

```
[PosixPath('datasets/flower_photos/roses/16209331331_343c899d38.jpg'),
PosixPath('datasets/flower_photos/roses/5777669976_a205f61e5b.jpg'),
PosixPath('datasets/flower_photos/roses/4860145119_b1c3cbaa4e_n.jpg'),
PosixPath('datasets/flower_photos/roses/15011625580_7974c44bce.jpg'),
PosixPath('datasets/flower_photos/roses/17953368844_be3d18cf30_m.jpg')]
```

```python
image_count = len(list(data_dir.glob('*/*.jpg')))

print(image_count)
```

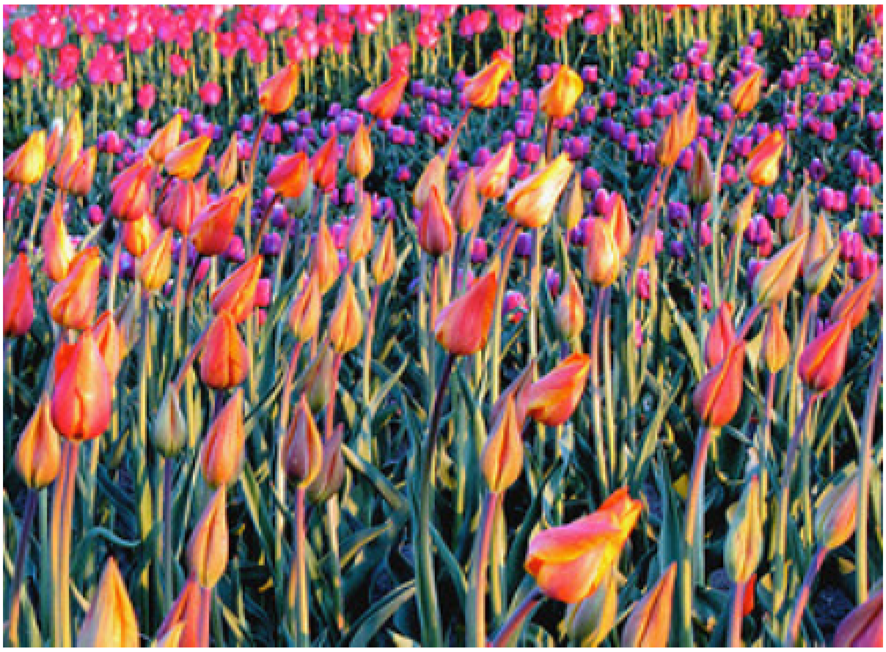
```
3670
```

```python
roses = list(data_dir.glob('roses/*'))

roses[:5]
```

```
[PosixPath('datasets/flower_photos/roses/16209331331_343c899d38.jpg'),
 PosixPath('datasets/flower_photos/roses/5777669976_a205f61e5b.jpg'),
 PosixPath('datasets/flower_photos/roses/4860145119_b1c3cbaa4e_n.jpg'),
 PosixPath('datasets/flower_photos/roses/15011625580_7974c44bce.jpg'),

PosixPath('datasets/flower_photos/roses/17953368844_be3d18cf30_m.jpg')]
```

```python
PIL.Image.open(str(roses[1]))
```

```
tulips = list(data_dir.glob('tulips/*'))

PIL.Image.open(str(tulips[0]))
```



Read flowers images from disk into numpy array using opencv

```
flowers_images_dict = {

    'roses': list(data_dir.glob('roses/*')),

    'daisy': list(data_dir.glob('daisy/*')),

    'dandelion': list(data_dir.glob('dandelion/*')),

    'sunflowers': list(data_dir.glob('sunflowers/*')),

    'tulips': list(data_dir.glob('tulips/*')),

}

flowers_labels_dict = {

     'roses' : 0,

     'daisy' : 1,

     'dandelion' : 2,

     'sunflowers' : 3,

     'tulips' : 4,
```

```
}
flowers_images_dict['roses'][:5]
```

```
[PosixPath('datasets/flower_photos/roses/16209331331_343c899d38.jpg'),
 PosixPath('datasets/flower_photos/roses/5777669976_a205f61e5b.jpg'),
 PosixPath('datasets/flower_photos/roses/4860145119_b1c3cbaa4e_n.jpg'),
 PosixPath('datasets/flower_photos/roses/15011625580_7974c44bce.jpg'),

PosixPath('datasets/flower_photos/roses/17953368844_be3d18cf30_m.jpg')]
```

```
str(flowers_images_dict['roses'][0])
```

```
'datasets/flower_photos/roses/16209331331_343c899d38.jpg'
```

```
img = cv2.imread(str(flowers_images_dict['roses'][0])) # Image to
TensorFlow Array
img.shape
```

```
(243, 500, 3)
```

```
cv2.resize(img,(180,180)).shape
```

```
(180, 180, 3)
```

```
X, y = [], []


for flower_name, images in flowers_images_dict.items():
    for image in images:
        img = cv2.imread(str(image))
        resized_img = cv2.resize(img,(180,180))
        X.append(resized_img)
        y.append(flowers_labels_dict[flower_name])
X = np.array(X)
y = np.array(y)
```

Train Test Split

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)
```

Preprocessing: scale images

```
X_train_scaled = X_train / 255

X_test_scaled = X_test / 255
```

Build convolutional neural network and train it

Model 1

```
num_classes = 5


model = Sequential([

  layers.Conv2D(16, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Conv2D(32, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Conv2D(64, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Flatten(),

  layers.Dense(128, activation='relu'),

  layers.Dense(num_classes)

])


model.compile(optimizer='adam',


loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),

              metrics=['accuracy'])


model.fit(X_train_scaled, y_train, epochs=10)
```

```
num_classes = 5

model = Sequential([
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
              
model.fit(X_train_scaled, y_train, epochs=10) 
```

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

Epoch 1/10

86/86 [==============================] - 72s 812ms/step - loss: 1.4518 - accuracy: 0.3699

Epoch 2/10

86/86 [==============================] - 60s 695ms/step - loss: 1.0961 - accuracy: 0.5658

Epoch 3/10

86/86 [==============================] - 59s 685ms/step - loss: 0.9009 - accuracy: 0.6595

Epoch 4/10

86/86 [==============================] - 60s 695ms/step - loss: 0.7650 - accuracy: 0.7049

Epoch 5/10

86/86 [==============================] - 57s 661ms/step - loss: 0.5782 - accuracy: 0.7925

Epoch 6/10

86/86 [==============================] - 57s 659ms/step - loss: 0.3821 - accuracy: 0.8750

Epoch 7/10

86/86 [==============================] - 57s 660ms/step - loss: 0.2305 - accuracy: 0.9226

Epoch 8/10

86/86 [==============================] - 57s 661ms/step - loss: 0.1566 - accuracy: 0.9488

Epoch 9/10

86/86 [==============================] - 56s 654ms/step - loss: 0.0930 - accuracy: 0.9727

Epoch 10/10

86/86 [==============================] - 61s 710ms/step - loss: 0.0734 - accuracy: 0.9786

<keras.callbacks.History at 0x7f7be4b8a7f0>

```
model.evaluate(X_test_scaled,y_test)
```

```
29/29 [==============================] - 7s 234ms/step - loss: 1.7062 -
accuracy: 0.6449

[1.7061505317687988, 0.6448801755905151]
```

Here we see that while train accuracy is very high (99%), the test accuracy is significantly low (64.6%) indicating overfitting. Let's make some predictions before we use data augmentation to address overfitting

```
predictions = model.predict(X_test_scaled)

predictions

29/29 [==============================] - 6s 215ms/step

array([[  1.5555904 ,   7.527551  ,   1.2190027 ,  -2.2760706 ,
         -1.9772646 ],
       [  2.2061357 ,  -0.8289972 ,  -1.1854742 ,  -3.4541612 ,
         -0.12450492],
       [ -3.4386787 ,  -9.29333   ,   9.819073  ,   2.8700411 ,
         -0.939783  ],
       ...,
       [ -3.6978543 ,   0.08210021,  -0.3332369 ,   6.0461645 ,
```

```
         -4.407517  ],
       [  3.4663086 ,  -3.3083336 ,  -1.974571  ,  -3.5091949 ,
          3.1733582 ],
       [ -6.8757224 , -13.090323  ,   6.503688  ,  10.634938  ,
         -0.60365117]], dtype=float32)
```

```
score = tf.nn.softmax(predictions[0])
```

```
predictions[0]
```

```
array([ 1.5555904,  7.527551 ,  1.2190027, -2.2760706, -1.9772646],
      dtype=float32)
```

```
np.argmax(score)
```

```
1
```

```
Y_test[0]
```

```
1
```

Improve Test Accuracy Using Data Augmentation

```
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
input_shape=(180, 180, 3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

Original Image

```
plt.axis('off')
plt.imshow(X[1])
```

<matplotlib.image.AxesImage at 0x7f7ce34f70a0>

Newly generated training sample using data augmentation

```
plt.axis('off')

plt.imshow(data_augmentation(X)[1].numpy().astype("uint8"))
```

<matplotlib.image.AxesImage at 0x7f7ce2efdd00>

Train the model using data augmentation and a drop out layer

Model 2

```
num_classes = 5
```

```
model = Sequential([

  data_augmentation,

  layers.Conv2D(16, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Conv2D(32, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Conv2D(64, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Dropout(0.2),

  layers.Flatten(),

  layers.Dense(128, activation='relu'),

  layers.Dense(num_classes)

])


model.compile(optimizer='adam',


loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),

              metrics=['accuracy'])


model.fit(X_train_scaled, y_train, epochs=10)

Epoch 1/10

86/86 [==============================] - 75s 845ms/step - loss: 1.3303
- accuracy: 0.4328

Epoch 2/10

86/86 [==============================] - 75s 878ms/step - loss: 1.0322
- accuracy: 0.5796

Epoch 3/10
```

```
86/86 [==============================] - 72s 838ms/step - loss: 0.9234
- accuracy: 0.6359

Epoch 4/10

86/86 [==============================] - 75s 871ms/step - loss: 0.8602
- accuracy: 0.6777

Epoch 5/10

86/86 [==============================] - 70s 816ms/step - loss: 0.7890
- accuracy: 0.7039

Epoch 6/10

86/86 [==============================] - 69s 808ms/step - loss: 0.7703
- accuracy: 0.7053

Epoch 7/10

86/86 [==============================] - 69s 799ms/step - loss: 0.7315
- accuracy: 0.7195

Epoch 8/10

86/86 [==============================] - 69s 804ms/step - loss: 0.7011
- accuracy: 0.7384

Epoch 9/10

86/86 [==============================] - 68s 791ms/step - loss: 0.6598
- accuracy: 0.7467

Epoch 10/10

86/86 [==============================] - 69s 797ms/step - loss: 0.6459
- accuracy: 0.7533

<keras.callbacks.History at 0x7f7b24fca9a0>
```

```
model.evaluate(X_test_scaled,y_test)
```

```
29/29 [==============================] - 7s 234ms/step - loss: 0.7363 -
accuracy: 0.7200

[0.7362971305847168, 0.7200435996055603]
```

You can see that by using data augmentation and drop out layer the accuracy of test set predictions is increased to 76.14%

Transfer Learning

```
import tensorflow_hub as hub
```

Model 3

Mobilenet V2 Model - Trained at Google

- 1.4 Million Images
- 1000 Classes

```
IMAGE_SHAPE = (224, 224)


classifier = tf.keras.Sequential([


hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/4", input_shape=IMAGE_SHAPE+(3,))

])

IMAGE_SHAPE+(3,)
```

```
(224, 224, 3)
```

```
x5_resized = cv2.resize(X[5], IMAGE_SHAPE)

plt.axis('off')

plt.imshow(X[5])
```

```
<matplotlib.image.AxesImage at 0x7f7a68696070>
```

```
predicted = classifier.predict(np.array([x5_resized]))

predicted = np.argmax(predicted, axis=1)

predicted
```

```
1/1 [==============================] - 1s 555ms/step

array([795])
```

Prepare Train and Test set and Resize them (224, 224, 3)

```
X, y = [], []


for flower_name, images in flowers_images_dict.items():

    for image in images:

        img = cv2.imread(str(image))

        resized_img = cv2.resize(img,(224,224))

        X.append(resized_img)

        y.append(flowers_labels_dict[flower_name])

X = np.array(X)

y = np.array(y)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)


X_train_scaled = X_train / 255

X_test_scaled = X_test / 255
```

Now take pre-trained model and retrain it using flowers images

```
feature_extractor_model =
"https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"


pretrained_model_without_top_layer = hub.KerasLayer(

    feature_extractor_model, input_shape=(224, 224, 3),
trainable=False)

num_of_flowers = 5


model = tf.keras.Sequential([

  pretrained_model_without_top_layer,
```

```
  tf.keras.layers.Dense(num_of_flowers)
])
```

```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer_1 (KerasLayer)  (None, 1280)              2257984

 dense_4 (Dense)             (None, 5)                 6405

=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
```

```
model.compile(
  optimizer="adam",
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['acc'])
```

```
model.fit(X_train_scaled, y_train, epochs=5)
```

```
Epoch 1/5
86/86 [==============================] - 104s 1s/step - loss: 0.8416 - acc: 0.6944
Epoch 2/5
86/86 [==============================] - 91s 1s/step - loss: 0.4212 - acc: 0.8492
```

```
Epoch 3/5

86/86 [==============================] - 80s 927ms/step - loss: 0.3270
- acc: 0.8881

Epoch 4/5

86/86 [==============================] - 79s 913ms/step - loss: 0.2691
- acc: 0.9135

Epoch 5/5

86/86 [==============================] - 78s 906ms/step - loss: 0.2309
- acc: 0.9313

<keras.callbacks.History at 0x7f7a5e750cd0>
```

```
model.evaluate(X_test_scaled,y_test)

29/29 [==============================] - 30s 1000ms/step - loss: 0.3873
- acc: 0.8725

[0.3872581720352173, 0.8725489974021912]
```

Conclusion: **As we can notice there is a significant increase in accuracy. The accuracy increased to 87.8 %. Hence we will be deploying Model 3.**

DEPLOYING THE MODEL

To Deploy Our Model we have used a python library known as **Gradio**.

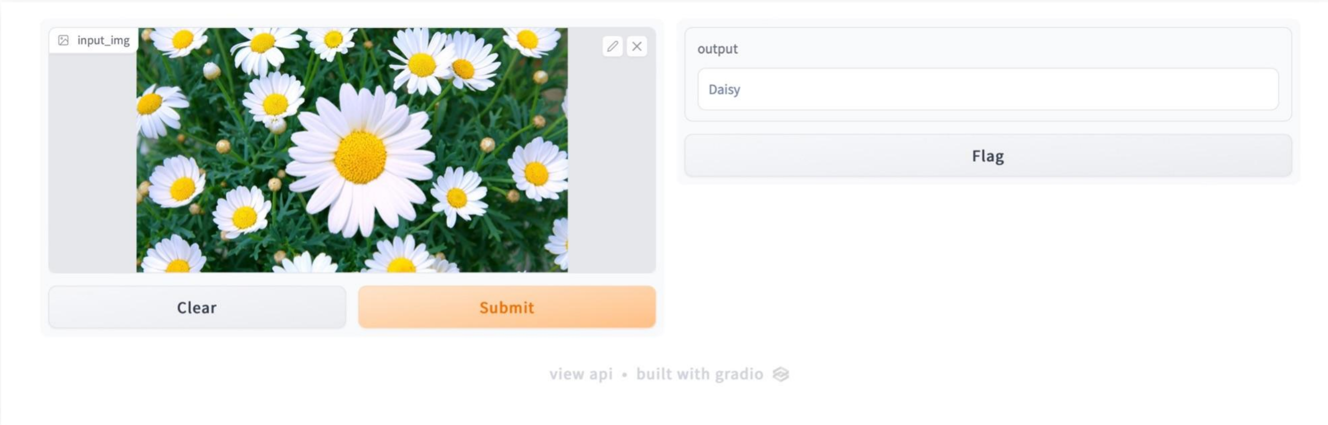
Gradio is the fastest way to demo your machine learning model with a friendly web interface.

- **Gradio can be** installed with pip**.**
- **Gradio can be embedded in** Python notebooks **or presented as a** web page**.**
- **A Gradio interface can automatically generate a public link you can share with colleagues that lets them interact with the model.**

Deploying Model 3

```
flowers = ["Roses", "Daisy", "Dandelion", "Sunflowers", "Tulips"]

import gradio as gr


def flower_classify(input_img):

    img_arr = input_img

    img_arr_scaled = cv2.resize(img_arr,(224,224))

    result = model.predict(img_arr_scaled[np.newaxis, ...])

    score = tf.nn.softmax(result)

    return flowers[np.argmax(score)]


demo = gr.Interface(flower_classify, gr.Image(shape=(224, 224)),
"text")


demo.launch()
```

WEB APPLICATION SCREENSHOT



Link for the Flower Classification Web App:

https://50546.gradio.app/502.html

RESEARCH PAPER AND OTHER REFERENCES

Reference :

Research Papers -

- Flower Classification with Deep CNN and Machine Learning Algorithms - **https://ieeexplore.ieee.org/document/8932908**
- Learning Salient Features for Flower Classification Using Convolutional Neural Network - https://ieeexplore.ieee.org/document/9194931
- Flower classification using CNN and transfer learning in CNN- Agriculture Perspective - https://ieeexplore.ieee.org/document/9316030

Other References -

- Kaggle For Flower Dataset - Kaggle: Your Machine Learning and Data Science Community
- Tensorflow - https://www.tensorflow.org
- Gradio As Deployment Library - Gradio