VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

FLOWER IMAGE CLASSIFICATION USING KERAS AND TENSORFLOW

PROJECT RESEARCH ARTICLE

CSE4019 IMAGE PROCESSING

By

19BCE1535 – ARASADA EKAVEERA ANEEL KUMAR [1]

19BCE1721 - B. KIRAN TEJA [1]

19BCE1216 - G. VINAY KUMAR [1]

Under the guidance of

Dr. GEETHA. S [2]

School of Computer Science and Engineering

Fall Semester 2021-2022

- 1 - Undergraduate, Vellore Institute of Technology, Chennai
- 2 - Professor Grade 2, Vellore Institute of Technology, Chennai

Abstract – **The classification of objects into their respective classes has always been one of the most important objectives in machine learning. Although the study of flowers and categorizing specific classes of flowers is an important subject in the field of botany, there are still some challenges in the recognition of flower images due to the similarity of diverse species of flowers, texture and color of flowers, and the dissimilarities amongst the same species of flowers. In comparison, the existing Google inception-v3 model needs more time and space to classify with high accuracy. We have demonstrated experimental performance on the TensorFlow platform to retrain flower category datasets, which can considerably reduce the time and space required for flower classification while marginally reducing accuracy.**

Keywords - **Flora Classification, Sequential Model, Tensorflow, Tkinter, Keras, Data Augmentation, Dropout**

I. INTRODUCTION

Flower classification is an important research project not only in Botany but also in Image Processing through Machine Learning. Flowers, one of the most abundant species in the world, have been discovered to have hundreds of species. When people use vision equipment to photograph flowers, they may become confused if they are unfamiliar with the species. As a result, having a quick and accurate flower classifier will capture people's interest. The similarities between different kinds of flowers, as well as the complex background of the flower image, present some obstacles in flower classification. We can't only use one property to distinguish them, such as color, texture, or shape etc. The same species of flowers will be different because of the shades of colors, shape, scale, viewpoint etc.

Flower classification has become a demanded requirement in computer science discipline since many applications ranging from medical to GIS. There are 250000 flower plants that have been classified and so many more still have not. Furthermore, many flower species may have similar names and shapes, but differ in color. Earlier, many approaches had been taken towards flower image classification. Earlier, in 1999 an approach that focuses on color only was proposed by Das at el. After that many approaches were developed considering different flower features such as texture, shape etc, and introducing new classification methods and algorithms.

II. MOTIVATION

The flower is the most distinctive and perceptible part of a plant, a subject of intense study among botanists, and is commonly used to identify species. Flowers come in a variety of colors, shapes, and textures, making it challenging to recognise the correct flora kind. Furthermore, as a result of hybrid planting practices, a few plants are developing as new forms that are even more difficult to recognise. So, in order to address the aforementioned issues, we decided to create a machine learning-based flower classifier model leveraging Tensorflow and Keras.

III. PROBLEM STATEMENT

It is critical to be able to recognise and recognise naturally occurring items. It is important for identifying flower types in a variety of sectors, including gardening, botany study, Ayurveda, medicine, farming, and floriculture. There are many various types of flowers in nature, and some of their characteristics are similar. Many flowers, for example, share the color red. These crimson blossoms, on the other hand, stand out

from the rest of the characteristics. Red flowers do not always have the same form. These similarities and contrasts emphasize the difficulties of automatically classifying each flower species. A botanist performs the traditional flower identification activity. Botanists have several obstacles when it comes to flower identifying. To solve the problem, we used the tensorflow flower dataset to train a model that classifies and identifies flower species.In addition, for accurate results, we employed data augmentation and dropout methods.

IV. LITERATURE SURVEY

[1] investigated Neural Network Architecture (NNA) as an image categorization approach. The framework is made out of two pairs of human eye mimics and variation sequence auto-encoding. It included many complicated photos, but as the investigation progressed, the system gradually improved the MNIST models. The open source database to be utilized as the training set is the MNIST. It was also tested using the Street View House Numbers dataset, and the results were enhanced because even human eyes couldn't tell the difference.

The magazine described an image categorization system based on the construction of a Convolutional Neural Network, according to [2]. (CNN). The training was carried out in such a way that a balanced amount of face pictures and non-face images were employed for training by extracting additional face images from the data of face images. On the Face Detection Data Set and Benchmark (FDDB), the image classification system employs the biscale CNN with 120 trained data and the auto-stage training achieves 81.6 percent detection rate with only six false positives, whereas the current state of the art achieves about 80 percent detection rate with 50 false positives.

According to [3], the picture classification techniques employed in the study were Decision Tree (DT). The DT contains many datasets, one for each Hierarchical classifier. It is required in order to compute membership in each of the classes. During the intermediate phases, the classifier allowed for partial rejection of the class. This method likewise has three (3) components: the first is to detect terminal nodes, and the second is to insert the class within it. The final one is node partitioning. This approach is believed to be highly simple and efficient.
This study is published in the journal [4], and it examines Support Vector Machine (SVM) active learning, which was gaining popularity at the time. It also provided some novel concepts by merging geographical information from the trial procedure with spectral data. It necessitates three techniques, the first of which is Euclidean distance. It derived some of the training samples from the major section of the spatial. The second option employs the Parzen window technique and, lastly, spatial entropy. According to the results, two of the photos have a high resolution in terms of efficacy.

According to the publication [5], rapid picture classification by enhancing the Fuzzy Classifiers was presented. It was a straightforward method of distinguishing between known and unknown categories. This strategy merely improves Meta knowledge in areas where local traits are prevalent. It was evaluated on large amounts of picture data and compared to the bag-of-features image model. The outcome was substantially greater classification accuracy since it was a testing method that delivered results in a short period of time that was 30% shorter than the prior one.

V. PROPOSED WORK

This section intends to employ a transfer learning strategy. The procedure is broken into major components, which are as follows:

- Efficiently loading a dataset off disk.
- Creating a Sequential model with Keras.
- Training and testing the model.
- Improving the model using data augmentation and dropout to overcome the overfitting issue.
- Identification & Classification of flower species.

VI. EXPERIMENTAL SETUP

Dataset-

https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz

Software Requirements - Python3, pip and Tensorflow2 packages, Anaconda Command Prompt, Jupyter notebook, Tkinter package (Note: Tensorflow2 is used for optimized performance.)

Hardware Requirements - 64 bit processor with GPU enabled (NVIDIA GPUs preferred).

VII. IMPLEMENTATION

Load data using a Keras utility:

Let's use the tf.keras.utils.image dataset from the directory tool to load these photos from the disc. In only a few lines of code, you can go from a directory containing photographs on disc to a tf.data.Dataset.

When creating the model, use a validation split. Let us utilize 75% of the photos for training and 25% for validation.

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_directory,
  validation_split=0.25,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 2753 files for training.
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_directory,
  validation_split=0.25,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 917 files for validation.
```

Fig.1 Use of Validity Split while loading the dataset.

Let's be sure to employ buffered prefetching so that we may get data from the disc without I/O becoming blocked. When loading data, we should utilize the following two methods:

After the pictures are loaded from the disc during the first epoch, Dataset.cache maintains them in memory. This will prevent the dataset from becoming a bottleneck when training your model. If your dataset is too huge to store in memory, you may use this approach to build a fast on-disk cache. During training, Dataset.prefetch overlaps data preparation and model execution.

Visualize the data:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 12))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

Fig.2 Visualizing few images

The output was shown in the Results section as Fig.11, Fig.12 and Fig.13.

Standardize the data:

The RGB channel values are between [0, 255]. This is not ideal for a neural network; in general, we should try to keep our input values as tiny as possible.

Using tf.keras.layers.Rescaling, we will normalize values to be in the [0, 1] range.

This layer may be used in two ways. We may use it on a dataset by executing Dataset.map.

Alternatively, we may incorporate the layer within our model definition, which will make deployment easier. Let's take the second approach here.

Create the Model:

Sequential Model:

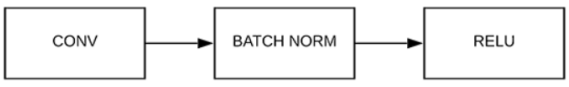


Fig.3 Sequential API - Flow Chart

The Sequential model is made up of three convolution blocks (tf.keras.layers.Conv2D), each with a max pooling layer (tf.keras.layers.MaxPooling2D). A ReLU activation function ('relu') activates a fully-connected layer (tf.keras.layers.Dense) with 128 units on top of it.

Choose the tf.keras.optimizers.Adam optimizer & tf.keras.losses.SparseCategoricaCrossentropy loss function. To view training and validation accuracy for each training epoch, pass the metrics argument to Model.

Henceforth, after building the model, train the model and improve it using Data Augmentation and Dropout.

Data Augmentation:

Overfitting is more common when there are a limited number of training instances. The concept of data augmentation is to generate new training data from our current examples by supplementing them with random changes that produce believable-looking visuals. This allows the model to be exposed to more parts of the data and generalize more effectively.

We will use the following Keras preprocessing layers to achieve data augmentation: tf.keras.layers.RandomRotation, tf.keras.layers.RandomFlip, as well as tf.keras.layers.RandomZoom. These, like other layers, may be incorporated in our model and executed on the GPU.

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

Fig.4 Data Augmentation to overcome overfitting.

Few enhanced instances by applying data augmentation to the same image many times.

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

Fig.5 Applying Data Augmentation to the same image.

The output was shown in the Results section as Fig.14.

Dropout:

Dropout regularization is another strategy for reducing overfitting in networks.
When we apply dropout to a layer, it randomly removes a number of output units from the layer throughout the training phase (by setting the activation to zero). Dropout accepts a fractional number as an input value, such as 0.1, 0.2, 0.4, and so on. This entails randomly removing 10%, 20%, or 40% of the output units from the layer in question.

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  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)
])
```

Fig.6 Dropout to overcome overfitting.

Compiling and Training the model:

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

```
model.summary()
```

Fig.7 Summary of the model

```
epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Fig.8 Training the model

The output was shown in the Results section as Fig.15 and Fig.16 respectively.

Classify the picture of flower of your own choice:
Now let us apply our model to the test by classifying a picture that wasn't in the training or validation sets.
Paste the image url of your choice in **'sample_image_url'**

```
sample_image_url = "https://static01.nyt.com/images/2021/02/21/realestate/17garden1/17garden1-mediumSquareAt3X-v2.jpg"
sample_image_path = tf.keras.utils.get_file('Sample_Image', origin=sample_image_url)

img = tf.keras.utils.load_img(
    sample_image_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

Fig.9 Classifying the pictures of flower

The output was shown in the Results section as Fig.17.

Now, install the Tkinter package in the system to create the Graphical User Interface (GUI) for the classification module. Before doing this, save the model in HD5 (Hierarchical Data Format) format.
Further, Use this model to classify the image using Tkinter GUI.

```python
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy

#load the trained model to classify the images

from keras.models import load_model
model = load_model('model1_flowers.h5')

#dictionary to label all the flower dataset classes.

classes = {
    0:'daisy',
    1:'dandelion',
    2:'roses',
    3:'sunflowers',
    4:'tulips',
}
#initialise GUI

top=tk.Tk()
top.geometry('800x600')
top.title('Image Classification of flowers')
top.configure(background='#FFFFFF')
label=Label(top,background='#CDCDCD', font=('Ubuntu',18,'bold'))
sign_image = Label(top)

def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((180,180))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    pred=numpy.argmax(model.predict([image])[0])
    sign = classes[pred]
    print(sign)
    label.configure(background='white', foreground='#DC143C',font=('montserrat',18,'bold'), text=sign)


def show_classify_button(file_path):
    classify_b=Button(top,text="Check Image",command=lambda: classify(file_path),padx=10,pady=5)
    classify_b.configure(background='white', foreground='#DC143C',font=('montserrat',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)

def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)

        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass

upload=Button(top,text="Upload new image",command=upload_image,padx=10,pady=5)
upload.configure(background='#DC143C', foreground='white',font=('montserrat',10,'bold'))

upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Image Classification on flowers",pady=20, font=('montserrat',20,'bold'))
heading.configure(background='#DC143C',foreground='white')
heading.pack()
top.mainloop()
```

Fig.10 Code Snippet for GUI using TKINTER package

The output was shown in the Results section as Fig.18.

VIII. RESULTS

Few Images from the dataset:

Fig.11

```
tulips = list(data_directory.glob('tulips/*'))
PIL.Image.open(str(tulips[0]))
```

Fig.12

Output for Visualizing few images:

Fig.13

Output for Data Augmentation to same image:

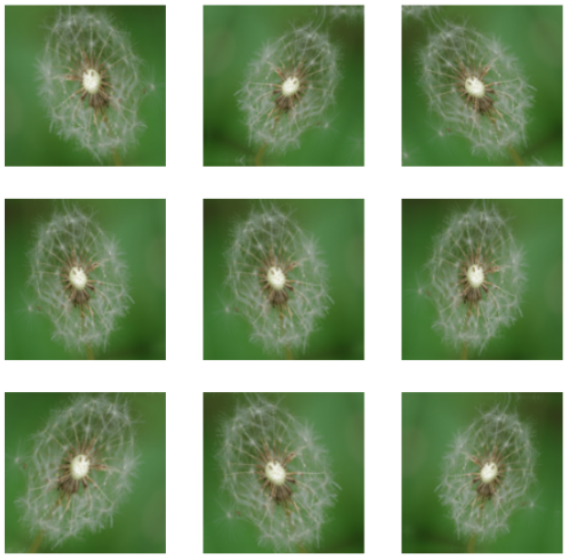


Fig.14

Output for Model Summary:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_1 (Sequential)    (None, 180, 180, 3)       0

rescaling_2 (Rescaling)      (None, 180, 180, 3)       0

conv2d_3 (Conv2D)            (None, 180, 180, 16)      448

max_pooling2d_3 (MaxPooling  (None, 90, 90, 16)        0
2D)

conv2d_4 (Conv2D)            (None, 90, 90, 32)        4640

max_pooling2d_4 (MaxPooling  (None, 45, 45, 32)        0
2D)

conv2d_5 (Conv2D)            (None, 45, 45, 64)        18496

max_pooling2d_5 (MaxPooling  (None, 22, 22, 64)        0
2D)

dropout (Dropout)            (None, 22, 22, 64)        0

flatten_1 (Flatten)          (None, 30976)             0

dense_2 (Dense)              (None, 128)               3965056

dense_3 (Dense)              (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

Fig.15

Output for training the model:

```
Epoch 1/15
87/87 [==============================] - 21s 236ms/step - loss: 1.3576 - accuracy: 0.3974 - val_loss: 1.1745 - val_accuracy: 0.
4973
Epoch 2/15
87/87 [==============================] - 20s 229ms/step - loss: 1.1085 - accuracy: 0.5387 - val_loss: 1.0720 - val_accuracy: 0.
5780
Epoch 3/15
87/87 [==============================] - 20s 231ms/step - loss: 1.0020 - accuracy: 0.5993 - val_loss: 0.9796 - val_accuracy: 0.
5987
Epoch 4/15
87/87 [==============================] - 20s 235ms/step - loss: 0.9192 - accuracy: 0.6487 - val_loss: 0.9004 - val_accuracy: 0.
6401
Epoch 5/15
87/87 [==============================] - 20s 234ms/step - loss: 0.8651 - accuracy: 0.6684 - val_loss: 0.9391 - val_accuracy: 0.
6434
Epoch 6/15
87/87 [==============================] - 20s 231ms/step - loss: 0.7999 - accuracy: 0.6974 - val_loss: 0.8244 - val_accuracy: 0.
6816
Epoch 7/15
87/87 [==============================] - 20s 231ms/step - loss: 0.7572 - accuracy: 0.7123 - val_loss: 0.7553 - val_accuracy: 0.
6979
Epoch 8/15
87/87 [==============================] - 20s 231ms/step - loss: 0.7147 - accuracy: 0.7319 - val_loss: 0.7808 - val_accuracy: 0.
6925
Epoch 9/15
87/87 [==============================] - 21s 241ms/step - loss: 0.7498 - accuracy: 0.7163 - val_loss: 0.7855 - val_accuracy: 0.
6957
Epoch 10/15
87/87 [==============================] - 20s 235ms/step - loss: 0.6984 - accuracy: 0.7323 - val_loss: 0.7272 - val_accuracy: 0.
7154
Epoch 11/15
87/87 [==============================] - 20s 232ms/step - loss: 0.6444 - accuracy: 0.7563 - val_loss: 0.7366 - val_accuracy: 0.
7143
Epoch 12/15
87/87 [==============================] - 20s 229ms/step - loss: 0.6207 - accuracy: 0.7581 - val_loss: 0.7028 - val_accuracy: 0.
7219
Epoch 13/15
87/87 [==============================] - 20s 231ms/step - loss: 0.6219 - accuracy: 0.7606 - val_loss: 0.7819 - val_accuracy: 0.
7088
```

```
Epoch 11/15
87/87 [==============================] - 20s 232ms/step - loss: 0.6444 - accuracy: 0.7563 - val_loss: 0.7366 - val_accuracy: 0.
7143
Epoch 12/15
87/87 [==============================] - 20s 229ms/step - loss: 0.6207 - accuracy: 0.7581 - val_loss: 0.7028 - val_accuracy: 0.
7219
Epoch 13/15
87/87 [==============================] - 20s 231ms/step - loss: 0.6219 - accuracy: 0.7606 - val_loss: 0.7819 - val_accuracy: 0.
7088
Epoch 14/15
87/87 [==============================] - 20s 230ms/step - loss: 0.5923 - accuracy: 0.7755 - val_loss: 0.7209 - val_accuracy: 0.
7317
Epoch 15/15
87/87 [==============================] - 20s 230ms/step - loss: 0.5525 - accuracy: 0.7904 - val_loss: 0.7167 - val_accuracy: 0.
7339
```

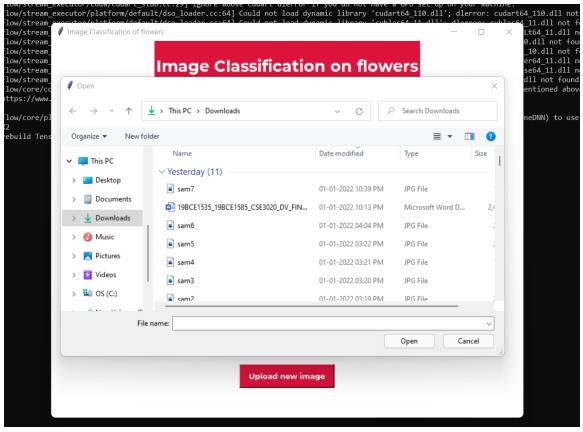
Fig.16

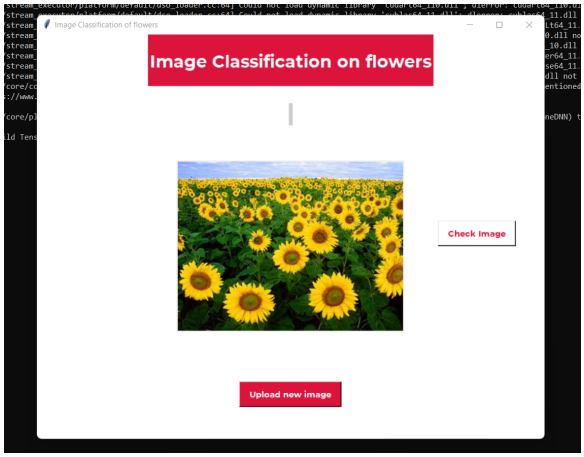
Output for Classifying the picture of flower:

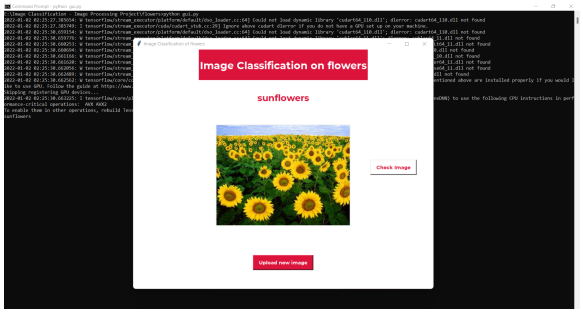
```
Downloading data from https://static01.nyt.com/images/2021/02/21/realestate/17garden1/17garden1-mediumSquareAt3X-v2.jpg
573440/571710 [==============================] - 0s 0us/step
581632/571710 [==============================] - 0s 0us/step
This image most likely belongs to roses with a 97.05 percent confidence.
```

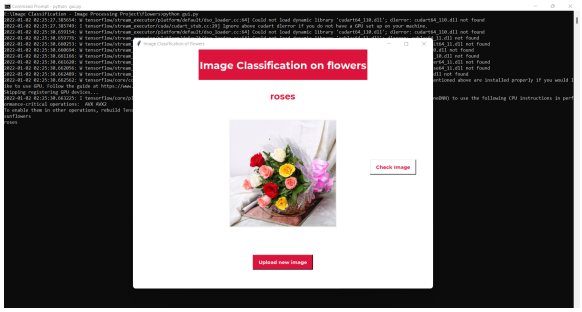
Fig.17

Output for GUI:









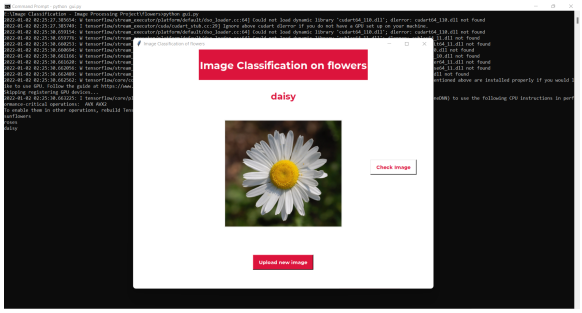


Fig.18

IX. CONCLUSION

The flower categorization relies heavily on image processing techniques. Classifiers are essential for testing data and determining the accuracy of categorization algorithms. Identifying various flower photos based on surface parameters is a difficult and costly operation.

we trained a flower classifier on the flower dataset.The results of classification showed that the proposed method could achieve higher accuracy than other methods.As future work, we are planning to develop a more accurate and effective deep network for flower image classification.Besides, we can extend the proposed method to be useful in other classification applications.

X. REFERENCES

- https://ijeecse.webs.com/N1001.pdf
- https://ieeexplore.ieee.org/document/8336590
- https://ieeexplore.ieee.org/document/9443129
- https://www.ripublication.com/irph/ijert19/ijertv12n4_16.pdf
- https://dl.acm.org/doi/10.1145/3436369.3437427
- [1] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation. https://doi.org/10.1038/nature14236
- [2] Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). XNOR-net: Imagenet classification using binary convolutional neural networks. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9908 LNCS, 525–542. https://doi.org/10.1007/978-3-319-46493-0_32
- [3] Kamavisdar, P., Saluja, S., & Agrawal, S. (2013). A survey on image classification approaches and techniques. Nternational Journal of Advanced Research in Computer and Communication Engineering, 2(1), 1005–1009. https://doi.org/10.23883/IJRTER.2017.3033.XTS7Z
- [4] Pasolli, E., Melgani, F., Tuia, D., Pacifici, F., & Emery, W. J. (2014). SVM active learning approach for image classification using spatial information. IEEE Transactions on Geoscience and Remote Sensing, 52(4), 2217–2223. https://doi.org/10.1109/TGRS.2013.2258676
- [5] Korytkowski, M., Rutkowski, L., & Scherer, R. (2016). Fast image classification by boosting fuzzy classifiers. Information Sciences, 327, 175–182. https://doi.org/10.1016/j.ins.2015.08.030