Flower Recognition

Project Increment 1

Course: CSCE 5200 – Feature Engineering, Fall 2021

Instructor: *Dr. Sayed Khushal Shah* (sayed.shah@unt.edu)

Team Members

1. Ngoc Phan (ngocphan@my.unt.edu)
2. Naga Sumanth Vankadari (nagasumanthvankadari@my.unt.edu)
3. Lakshmi Vandana Nunna (lakshmivandananunna@my.unt.edu)
4. Manoj Kolluri (manojkolluri@my.unt.edu)

GitHub Repository

https://github.com/nphan20181/Feature_Engineering_Project

Introduction

Idea Description

The idea of our project is to build a model that could recognize/identify the type of the flower given in the input image. A user-friendly front end would be integrated with this fully trained model. Users would be able to upload an image, which would then be fed to the backend model to retrieve the prediction (type of flower) and display the result in the frontend. It is reasonable to state that this model would not be equivalent to a much more sophisticated model like AWS Rekognition API since it is trained on massive datasets and is built on top of state-of-the art neural networks. So, we could consider this model as standard - to compare against our model based on the Receiving Operating Curve (ROC) /accuracy metrics. The result page of the application would have the predictions from both our model and AWS Rekognition API along with the comparison metrics. This GUI would help us measure the improvement in performance, every time we update/increment the model.

Motivation & Significance

The real-applications of plant/flower detections are emerging year on year. There are numerous unidentified flowers around the globe. These flowers could just be new types of an existing flower species or they could also be a completely new species. Identifying their family helps in the breakthrough research efforts in the fields of pollination, cloning, medicine, evolution study, plant science. Adding on, identifying the flower helps us determine its host plant. This helps to understand whether the flower belongs to a poisonous plant (refer to Figure 1) or the flower attracts poisonous bugs or the flower releases certain enzymes that trigger allergic reactions. Pollen allergies are extremely common - especially in the United States impacting almost 50 million Americans. These allergies range from mild to severe, which often lead to the death of a person. So, lack of an automated process that could detect and provide information on the vegetation is a challenge to people who do not have extensive knowledge on plants.

Our application could be improvised to connect to a reliable source on the internet and pull up data on the identified image label and display it to the user. This helps the user to gain knowledge and refrain from the vicinity of harmful/allergy triggering plants.

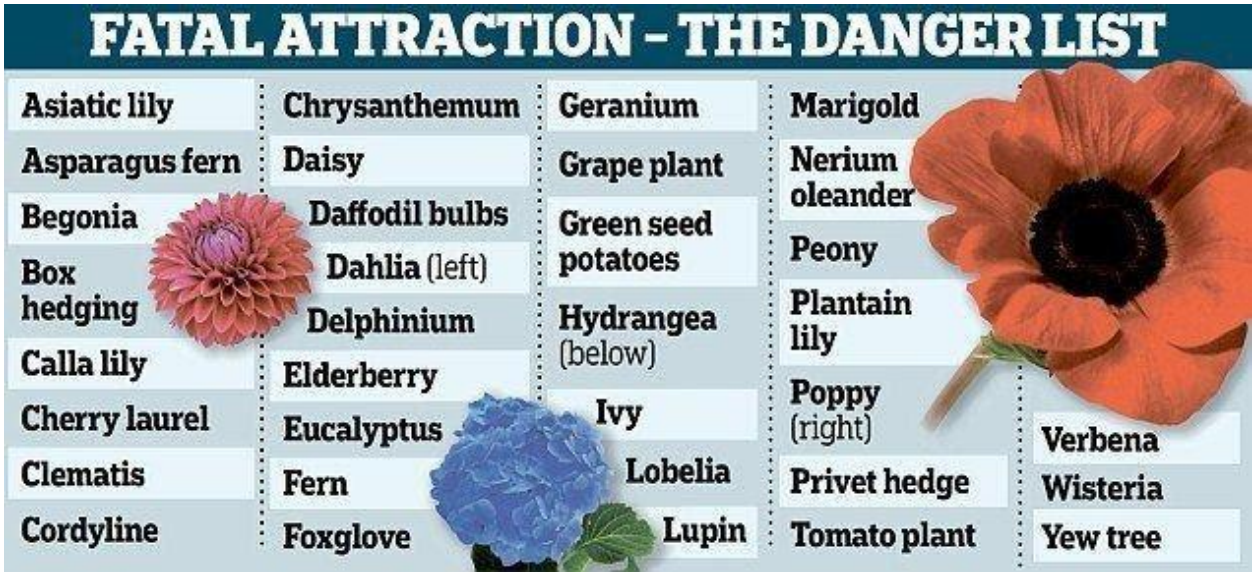


***Fig. 1**: Flowering plants that are fatal to humans [3].*

Goals & Objective

The objective of our application is to create a fully trained model that identifies the type of the flower with optimal accuracy. Then, leverage this model to predict the image fed by the user from the front end and display the output. Since we also have the prediction of the AWS Rekognition API displayed as well, we get to see the performance of our model with ease.

Technical Goals:

1. GUI to enable the user to upload an image.
2. Fully trained model that takes in vectorized images and predicts the label.
3. GUI to display the predictions and metrics of both AWS Rekognition API and the trained model

Related Work

1. **Hyperspectral Images Classification with Gabor Filtering and Convolutional Neural Network** [1]

   This article, written by Chen et al in 2017, proposed a Gabor-Convolutional Neural Network (Gabor-CNN) to classify the hyperspectral images. The Gabor-CNN model is trained on three different hyperspectral datasets captured in different regions. These datasets include Salinas, Kennedy Space Center, and Indian Pines. The proposed method first uses Principal Component Analysis (PCA) to reduce the dimensionality of the images, and then apply a Gabor filter to extract the spatial features from the dimensionally reduced images. Next, the Gabor features are fed into a Convolutional Neural

Network (CNN) model that contains three convolutional layers, three ReLU layers, and three pooling layers. The result of the Gabor-CNN model is then compared with that of the CNN model, PCA-CNN model, extended morphological profiles (EMP) CNN model, EMP Support Vector Machine (SVM) model, and radial basis function kernel (RBF) SVM model. As a result, the Gabor-CNN model seems to work well with a small dataset because it outperforms all other models in terms of overall accuracy, average accuracy, and Kappa coefficient for a sample size of 200 or less in all three datasets. However, for sample sizes between 300 and 800, the EMP-CNN model performs better than the Gabor-CNN model in most cases. Furthermore, the CNN and PCA-CNN models seem to perform worse on small sample sizes as the models suffer from overfitting because there are a large number of features in the images but not enough examples to learn.

2. **Gaussian Transfer Convolutional Neural Networks** [4]

   This article, written by Wang et al in 2019, proposed a Gaussian transfer convolutional neural networks (GT-CNN) models for image classification tasks. The datasets used in the study are MNIST, CIFAR-10, CIFAR-100, and NWPU-RESISC45. For CFAR-10 and CFAR-100 datasets, three GT-CNN models are trained using different kernel sizes of 3 by 3, 5 by 5, and 7 by 7, respectively. For other datasets, only one GT-CNN model is trained. Finally, the results of the GT-CNN models are compared with those of the CNN-Base, VGG, ResNet, and Wide ResNet models. Unlike the CNN-Base model that separates the convolutional layer and the max pooling and ReLu layer, the GT-CNN models combine the convolutional layer and max pooling and ReLu layer into a Gaussian Convolutional (GaussC) layer to produce a condensed feature map. The GaussC layer replaces the pooling layer in the CNN model by incorporating Gaussian filters with convolutional kernels to mitigate the loss of data features caused by the pooling layer. As a result, the GT-CNN model outperforms all other models in the CFAR-10, CFAR-100 datasets in terms of error percentage. For the MNIST dataset, a GT-CNN with 7 by 7 kernels has the lowest error percentage compared to that of other models. Furthermore, the GT-CCN model also has higher accuracy than that of other models for the NWPU-RESISC45 dataset.

3. **Extension of Convolutional Neural Network with General Image Processing Kernels** [5]

   This article, written by Jay Hoon Jung in 2018, proposes applying various image processing kernels such as sharpening, embossing, blur etc., to design a general filter convolutional neural network (GFNN) for image classification tasks. The data set used for this study is MNIST dataset that contains 60000 training images and 10000 testing images and since the classification features were simple the 28*28 one channel grey scale images were highly regularized and then trained on both GFNN and a normal CNN model. The CNN model contains 3 convolution and max pooling layers

followed by 41 3×3 filters 64 3×3 filters, and 128 3×3 filters and then 2×2 max pooling was used after the convolution layer. The two dense layers have 2048×625 and 625×10 dimensions respectively. The GFNN has the exact same structure as the CNN model except for the first layer. The first layer contains a layer, named as filter layer, that has 41 predefined 3×3 kernels that dominate the whole network. The filter consists of 3 sharpening filters, 1 embossing filter, 2 blurring filters, 32 edge detecting compass gradient operators, 2 second order operators, and 1 discrete cosine transformation operator. After training the models and comparing their performances it was found that training time for 55000 samples of MNIST dataset GFNN is 30% less than that of CNN on average and the accuracy is 0.2% higher.

Dataset

Data Collection

The flower dataset which has been collected on Kaggle [2] contains 4,317 RGB images of different sizes. There are five flower types in the dataset: chamomile, tulip, rose, sunflower, and dandelion. The distribution of images in each flower type is disproportionate. There are a greater percentage of images for dandelion (24.4%) and tulip (22.8%) than those of rose, daisy, and sunflower. The image distribution for rose, daisy, and sunflower are approximately the same. Figure 2 displays the data distribution for the five flower types.

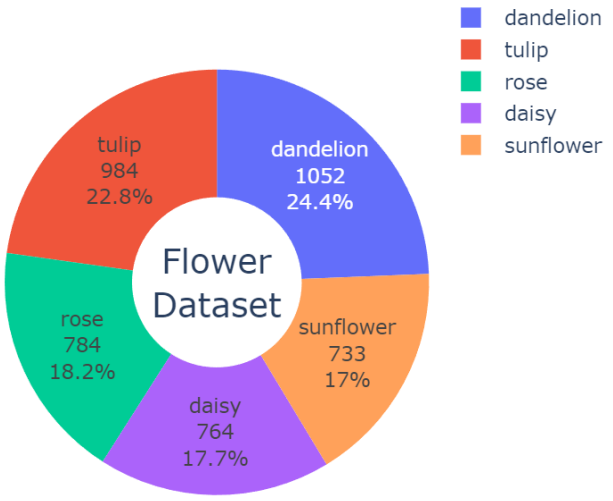


***Fig. 2**: Data distribution of Flower Dataset for each flower type along with the corresponding number of images and percentage of images.*

Detail Design of Features

Based on the descriptive statistics of flower images in Table 1, the image's width ranges from 134 to 1,024 pixels while the image's height ranges from 80 to 442 pixels. Furthermore, the number of image pixels ranges from 19,200 to 221,000. In addition, the range of red color percentage seems to be widest while the range of blue color percentage seems to be shortest.

	Range	Min	Max	Mean	Median	Standard Deviation
Image Width	890.000000	134.000000	1024.000000	338.379893	320.000000	119.067232
Image Height	362.000000	80.000000	442.000000	253.073662	240.000000	61.558205
Image Pixels	201800.000000	19200.000000	221000.000000	91445.254112	76480.000000	52130.492116
Red %	0.888605	0.084733	0.973338	0.395124	0.377033	0.079271
Green %	0.807317	0.010713	0.818030	0.358717	0.355041	0.060148
Blue %	0.623759	0.003307	0.627066	0.246159	0.260725	0.083258

Table 1: *Descriptive statistics of flower images.*

As shown in Figure 3, most flower images have less than 100,000 pixels. Additionally, there are also a good number of image pixels that are between 160,000 and 170,000.

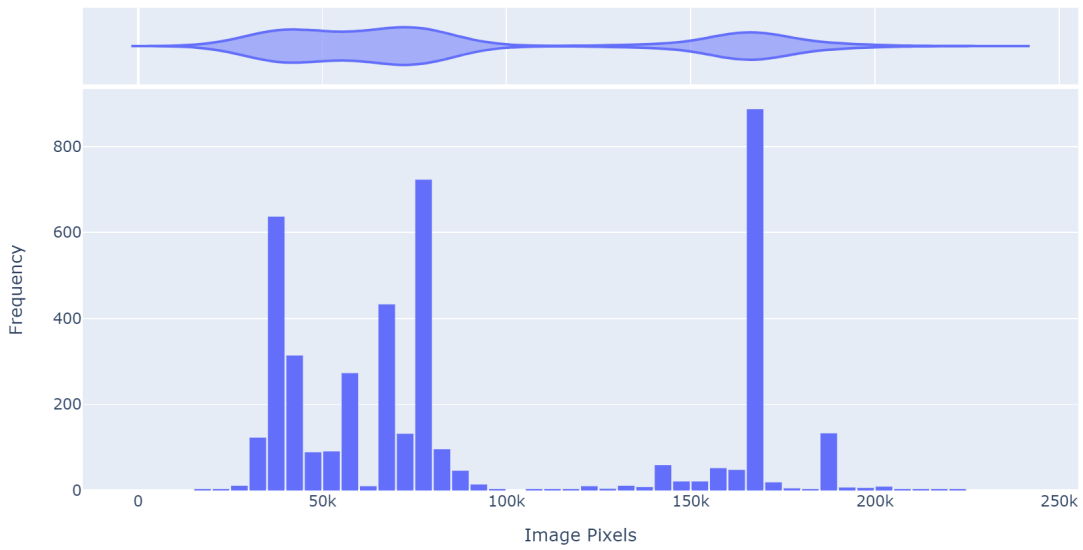


Fig. 3*: Frequency distribution of image pixels.*

Figure 4 displays scatter plots of Red/Green/Blue percentage per flower type. Based on figure 4a, most rose flowers seem to have a higher red percentage and a lower green percentage than those of other flowers. Additionally, sunflowers seem to have a lower range for green percentage compared to that of other flowers. Furthermore, figure 4b also indicates that most flowers have a blue percentage less than 50%.

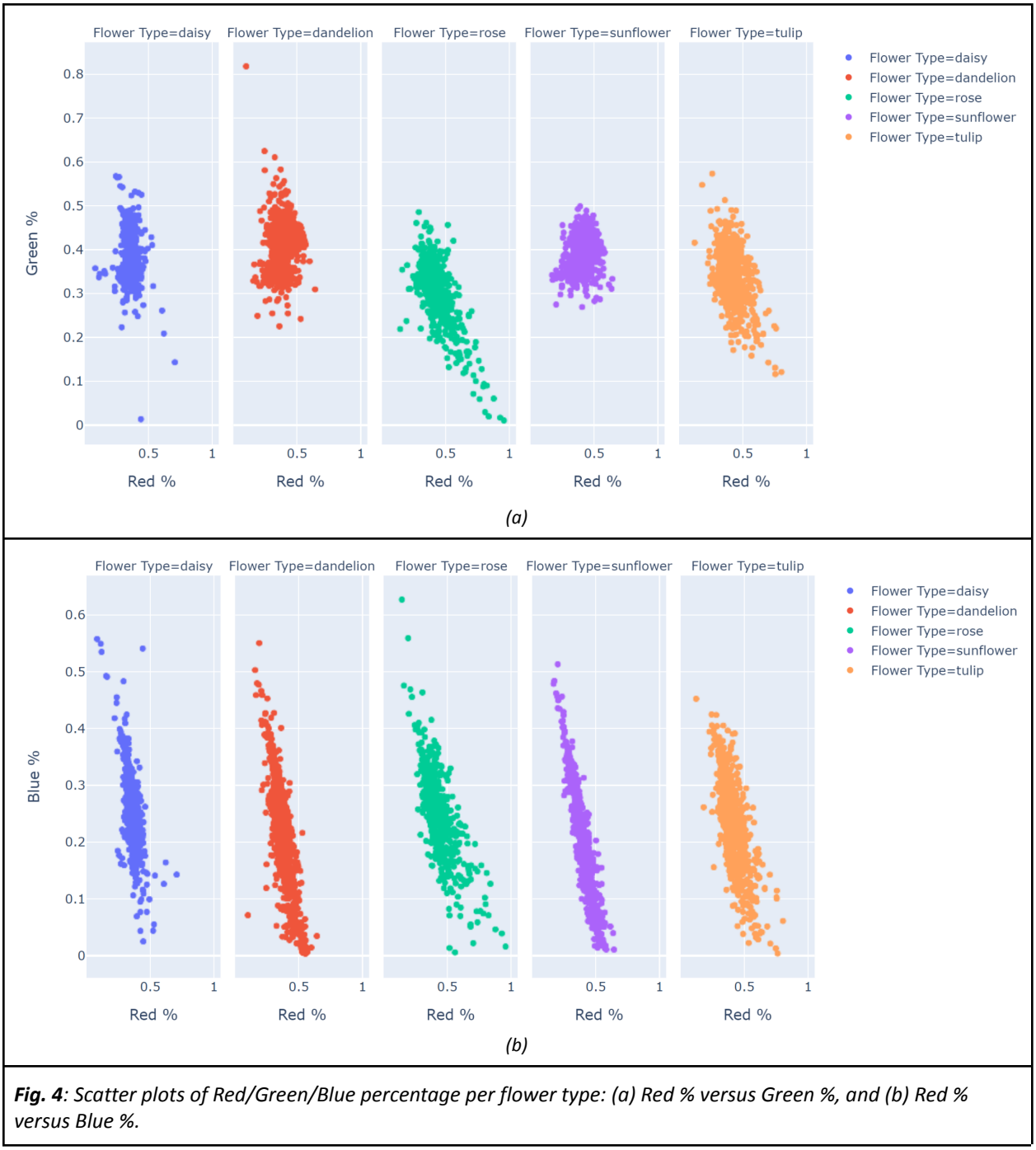
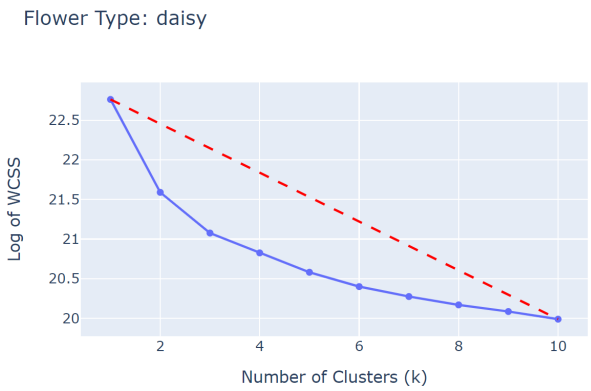


Fig. 4*: Scatter plots of Red/Green/Blue percentage per flower type: (a) Red % versus Green %, and (b) Red % versus Blue %.*

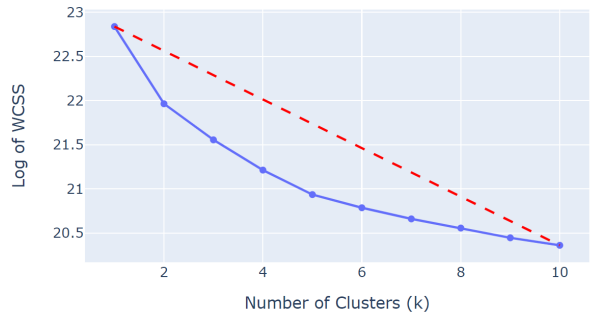
Analysis

For further analysis, we study the dominant colors for each flower type separately. Before plotting the dominant colors for each flower type dataset, we fit five K-Means models, one for each flower type dataset, and select the best k value using the Elbow method. Figure 5 displays plots of the natural logarithm of Within-Cluster Sum of Square Error (WCSS) for each k value. Based on the plots in Figure 5, the best k values for daisy, dandelion, rose, sunflower, and tulip dataset are 3, 5, 3, 5, and 3, respectively.

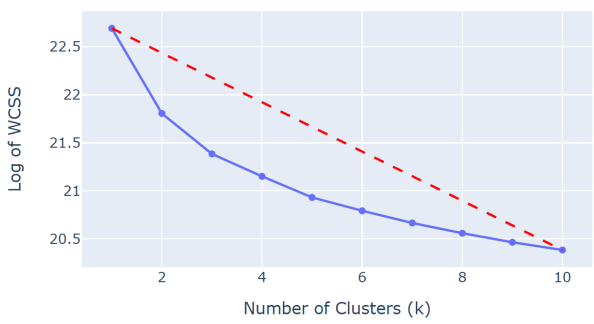
Fig. 5: *Plot of the natural logarithm of WCSS versus number of clusters (blue line) for (a) daisy dataset, (b) dandelion dataset, (c) rose dataset, (d) sunflower dataset, and (e) tulip dataset. The dash red line is used as a benchmark for selecting the best value of k, which is a point on the blue line that is farthest away from the dash red line.*



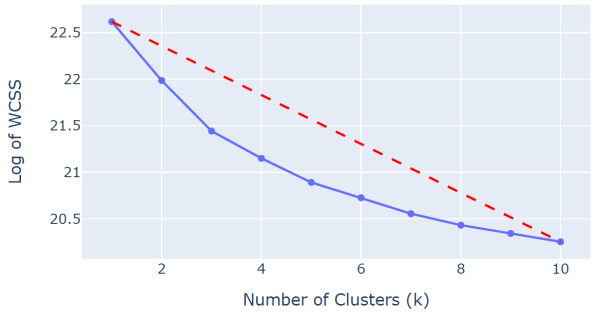
(a)



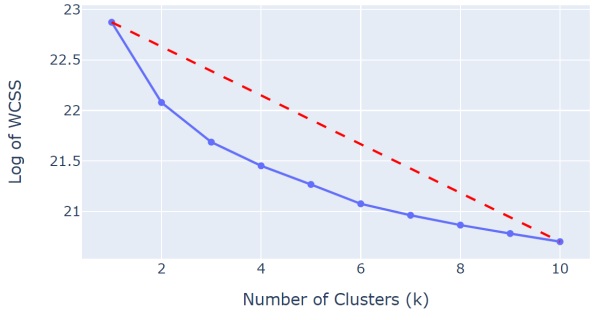
(b)



(c)



(d)



(e)

After identifying the best k values, we train five separated K-Means models to identify dominant colors for daisy flowers, dandelion flowers, rose flowers, sunflowers, and tulip flowers. According to the dominant colors plots in Figure 6, all flowers have black as their dominant color. Both dandelion and sunflower have yellow as their dominant color; however, sunflowers have a higher percentage of yellow than that of dandelion. As expected, two out of the three dominant colors for roses are pink and brownish red. Furthermore, the dominant colors for tulips are somewhat similar to the dominant colors for roses.

***Fig. 6**: Dominant colors for (a) daisy flowers, (b) dandelion flowers, (c) rose flowers, (d) sunflowers, and (e) tulip flowers.*

(a)

(b)

(c)

(d)

(e)

Implementation

Work Process Flow

Feature Extraction & Model Training

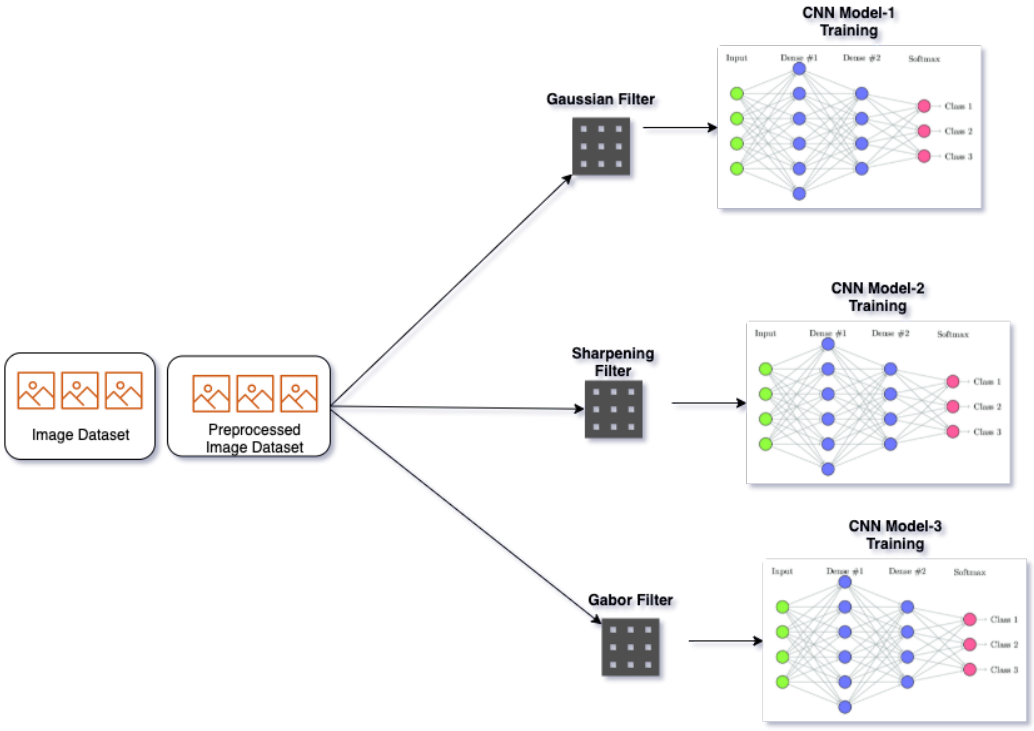


***Fig. 7**: Work Process Flow Diagram*

- In the preprocessing phase, images will be converted to uniform resolution and to increase the dataset size - images will be rotated (to more than 45 degrees).
- Each of this set of images will be used for training 3 independent models. Different image filters - Gaussian, Sharpening, Gabor to extract features for the image dataset and will be passed as an input to the CNN models respectively.
- Standard metrics such as confusion matrix, ROC, Accuracy & etc. will be recorded.
- Optional: Need to research on model concatenation, if there is a way to merge all 3 trained models

Website Integration

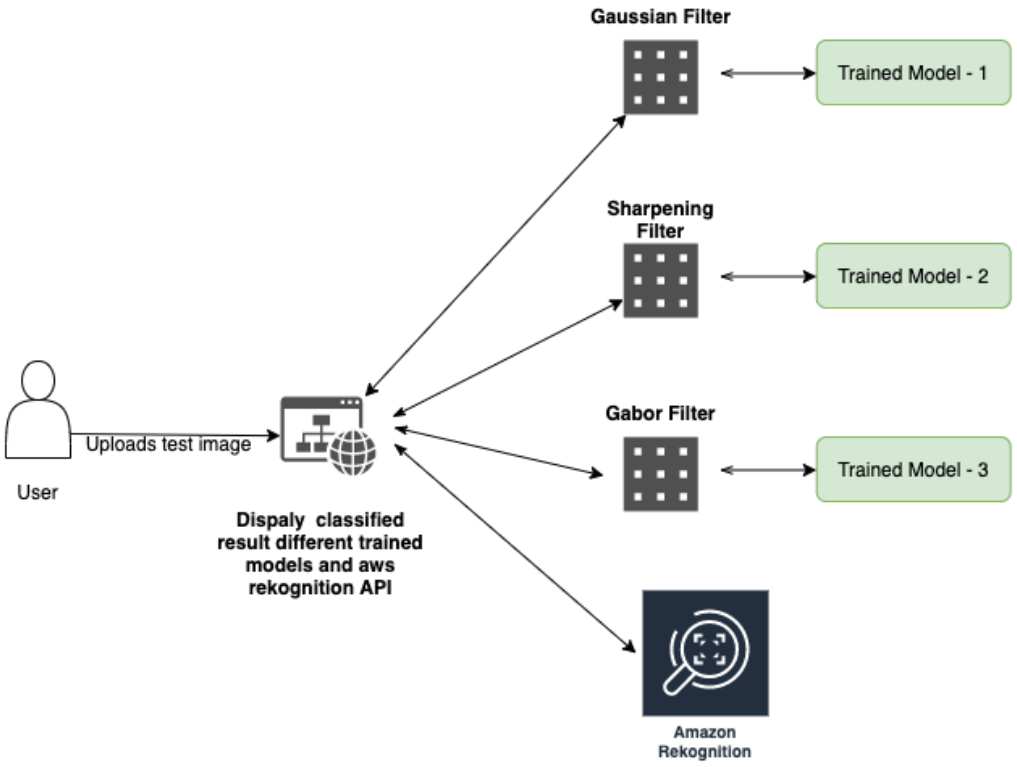


***Fig. 8**: Website Integration*

- When a user uploads an image, necessary preprocessing is performed on the image; as in model training such as resize, apply any filter. Resultant image is passed as input to trained model is invoked to extract the classified result.
- Also, the uploaded image is passed as an input to the AWS rekognition API to get the classified result.
- Website will be developed using the flask framework and deployed on a local host.

Models

We are building a neural network with convolutional layers along with the respective max pooling layers whose output would then be flattened to create a vector of the input image. The neural network is then further composed of fully connected(Dense) layers (**refer to Figure 9**) that would learn the weights of each dimension of the resultant vector.

Currently, we added 4 drop out layers along with the aforementioned layers to avoid overfitting of the model.

Visualization of the Dense and Dropout layers in the neural network:

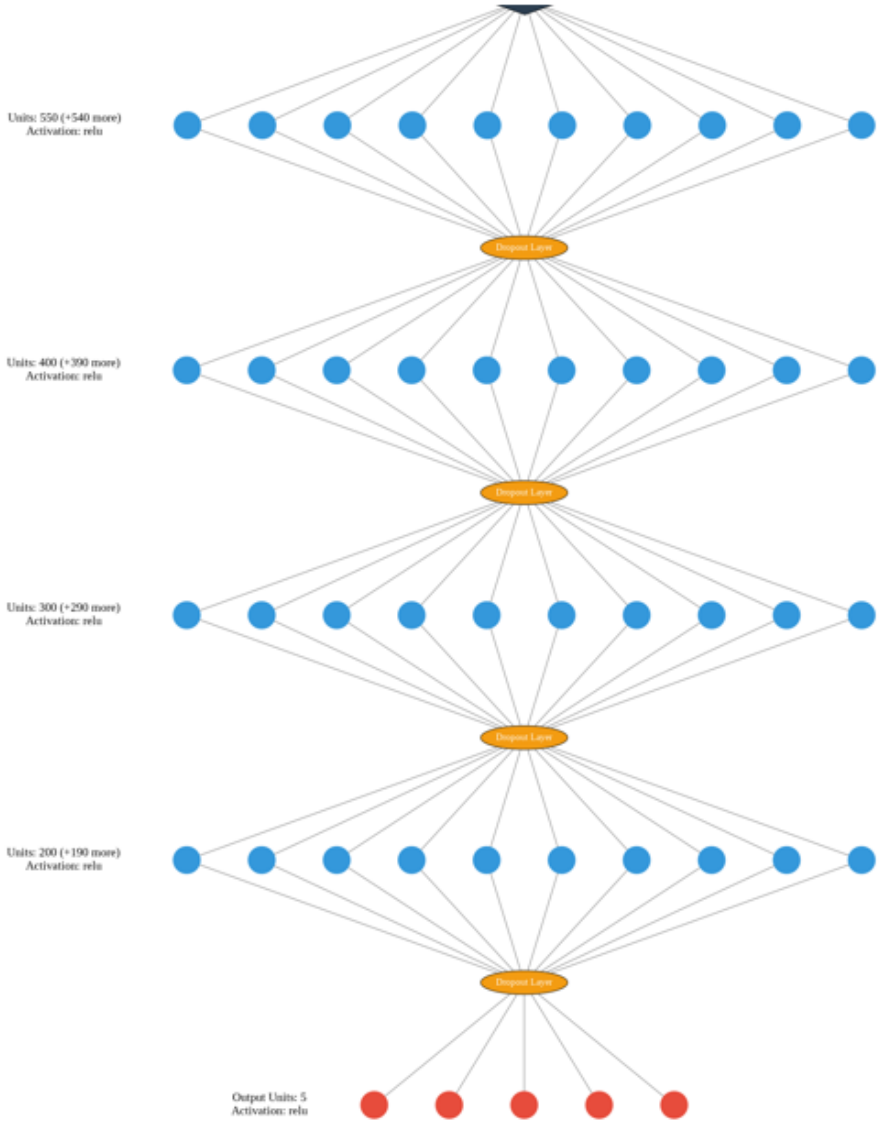


***Fig. 9**: Fully Connected layers and Dropout layers in the model*

Gaussian-Convolutional Neural Network

We applied cv2 Gaussian Blur function on the image data and created a new column that holds these newly generated gaussian image data. Below is a code snippet from the notebook for your reference.

```
gauss_features=[]

for image in image_df['image_data']:
   gauss_feature = cv2.GaussianBlur(image, (3, 33), 0)
   gauss_features.append(gauss_feature)


image_df=image_df.assign(gauss = gauss_features)
```

This is used as the X parameter while the categorically encoded (class) column is treated as the Y parameter. A stratified split is applied on this dataset to balance the class proportions in train, validation and test sets.

```
from sklearn.model_selection import train_test_split


x, x_test, y, y_test =
train_test_split(image_df['gauss'].to_list(),image_df['category'].to_list(
), stratify=image_df['category'].to_list(), test_size=0.2,train_size=0.8)

x_train, x_val, y_train, y_val =
train_test_split(image_df['gauss'].to_list(),image_df['category'].to_list(
), stratify=image_df['category'].to_list(),test_size = 0.25,train_size
=0.75)


#Class frequency in trainset

import numpy as np

(uniq, freq) = (np.unique(y_train, return_counts=True))

print("Class frequency in Train Set:")

print(np.column_stack((uniq,freq)))


#Class frequency in testset

import numpy as np

(uniq, freq) = (np.unique(y_test, return_counts=True))
```

```
print("Class frequency in Test Set:")
print(np.column_stack((uniq,freq)))

#Class frequency in validationset
import numpy as np
(uniq, freq) = (np.unique(y_val, return_counts=True))
print("Class frequency in Validation Set:")
print(np.column_stack((uniq,freq)))
```

Output:

```
Class frequency in Train Set:
[[  0 193]
 [  1 307]
 [  2 215]
 [  3 208]
 [  4 392]]
Class frequency in Test Set:
[[  0  51]
 [  1  82]
 [  2  57]
 [  3  56]
 [  4 105]]
Class frequency in Validation Set:
[[  0  64]
 [  1 102]
 [  2  72]
```

```
[  3  70]

[  4 131]]
```

Aforementioned model is then trained and tested based on this data.

Gabor-Convolutional Neural Network

We applied cv2 Gabor Kernel on the image data and created a new column in the pandas dataframe to hold these newly generated gbror image data, the function that was used to create these features is given in the code snippet below.

```
gabor_features=[]
g_kernel = cv2.getGaborKernel((21, 21), 8.0, np.pi/4, 10.0, 0.5, 0, ktype=cv2.CV_32F)
for image in image_df['image_data']:
    gabor_feature = cv2.filter2D(image, cv2.CV_8UC3, g_kernel)
    gabor_features.append(gabor_feature)
image_df=image_df.assign(gabor = gabor_features)
```

+ Code + Markdown

```
image_df.head()
```

	image_data	class	category	gabor
0	[[[39, 44, 45], [39, 44, 45], [41, 46, 48], [4...	dandelion	1	[[[255, 255, 255], [255, 255, 255], [255, 255,...
1	[[[11, 30, 38], [9, 28, 35], [7, 27, 33], [6, ...	dandelion	1	[[[43, 103, 0], [45, 114, 4], [55, 141, 55], [...
2	[[[163, 162, 158], [169, 166, 162], [165, 160,...	dandelion	1	[[[255, 255, 255], [255, 255, 255], [255, 255,...
3	[[[0, 199, 212], [0, 195, 210], [0, 191, 210],...	dandelion	1	[[[5, 255, 255], [6, 255, 255], [8, 255, 255],...
4	[[[1, 12, 9], [1, 12, 9], [3, 14, 11], [3, 14,...	dandelion	1	[[[37, 77, 47], [37, 76, 49], [31, 73, 54], [1...

These features unter the gabor column were then used as the x parameter while the labels in the category column were used as the y parameter, after which a stratified split was performed to split the data into training, testing and validation sets which were then used to train and test the model.

Preliminary Results

Summary of current version of model:

```
Model: "sequential_4"
_________________________________________________________________
```

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 178, 178, 16)	448
max_pooling2d_16 (MaxPooling	(None, 89, 89, 16)	0
conv2d_17 (Conv2D)	(None, 87, 87, 32)	4640
max_pooling2d_17 (MaxPooling	(None, 43, 43, 32)	0
conv2d_18 (Conv2D)	(None, 41, 41, 64)	18496
max_pooling2d_18 (MaxPooling	(None, 20, 20, 64)	0
conv2d_19 (Conv2D)	(None, 18, 18, 128)	73856
max_pooling2d_19 (MaxPooling	(None, 9, 9, 128)	0
flatten_4 (Flatten)	(None, 10368)	0
dense_20 (Dense)	(None, 550)	5702950
dropout_16 (Dropout)	(None, 550)	0
dense_21 (Dense)	(None, 400)	220400
dropout_17 (Dropout)	(None, 400)	0
dense_22 (Dense)	(None, 300)	120300
dropout_18 (Dropout)	(None, 300)	0
dense_23 (Dense)	(None, 200)	60200

```
dropout_19 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_24 (Dense)             (None, 5)                 1005
=================================================================
Total params: 6,202,295
Trainable params: 6,202,295
Non-trainable params: 0
_________________________________________________________________
```

Gaussian-Convolutional Neural Network

We observed that the training accuracy and loss are comparatively better than the validation dataset (**refer to Fig 10**). Our goal is to reduce the difference in results between these 2 datasets to make sure that the model is not over fitting on the training dataset. We are planning to apply grid search to arrive at the optimal hyperparameters that could achieve this.

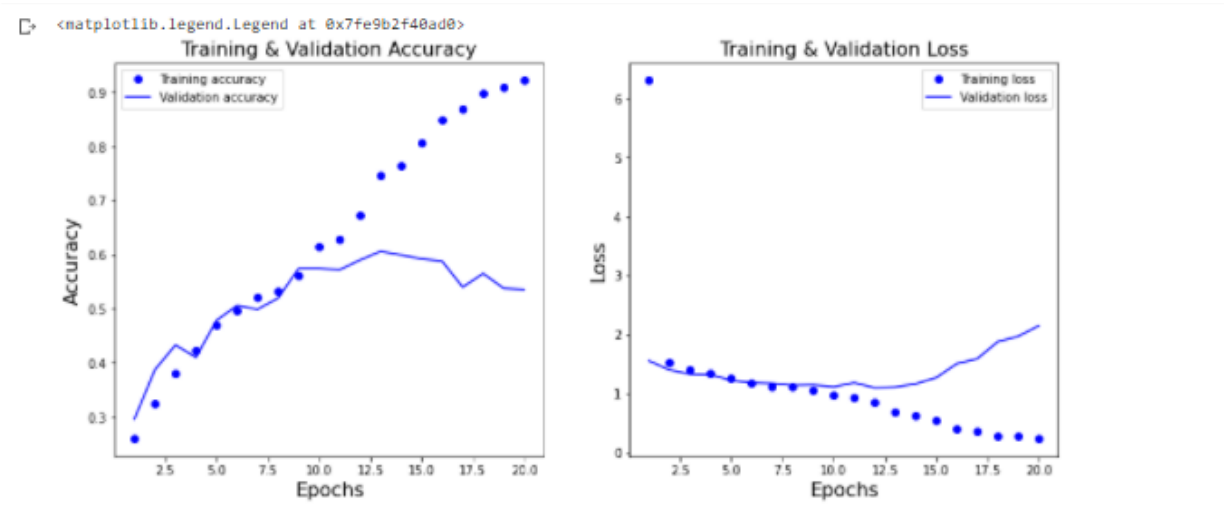


***Fig. 10**: Accuracy and Loss result over multiple epochs*

Below is a snippet from the notebook that displays the data frame that is created after generating gaussian image data along with encoding the class column to generate the category column. You can see that the image data is of shape (180,180,3). This is because we resize every image to 180 by 180 which creates the image data. The 3rd dimension represents the proportions of 3 basic colors ie; Red,Green and Blue.

```
image_df.head()
```

	image_data	class	category	gauss
0	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	tulip	4	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...
1	[[[56, 64, 48], [15, 36, 23], [31, 41, 32], [3...	tulip	4	[[[50, 50, 36], [65, 61, 44], [86, 78, 57], [7...
2	[[[15, 46, 18], [17, 47, 14], [18, 46, 17], [1...	tulip	4	[[[28, 63, 26], [28, 63, 27], [28, 64, 28], [2...
3	[[[99, 91, 66], [4, 30, 26], [4, 10, 6], [33, ...	tulip	4	[[[49, 66, 49], [41, 58, 44], [33, 48, 38], [4...
4	[[[27, 38, 41], [58, 83, 96], [203, 135, 159],...	tulip	4	[[[46, 71, 93], [57, 79, 100], [91, 101, 116],...

```
[ ] print(image_df.loc[3]['gauss'].shape)
```

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

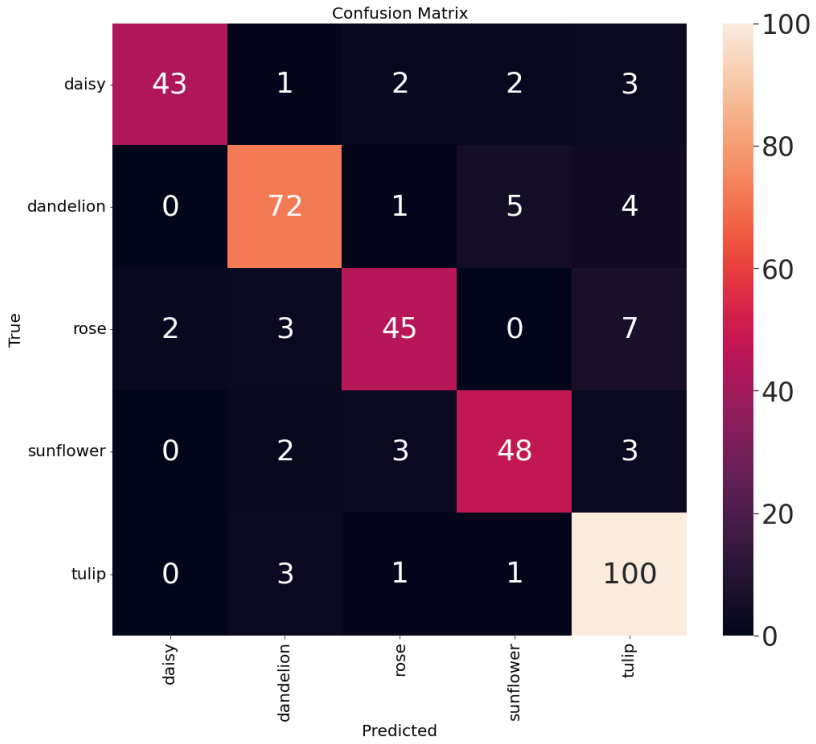


***Fig. 11**: Confusion Matrix*

Once we pulled a confusion matrix with the prediction versus actual labels (**refer to Fig 11**), we observed that the model is majorly successful in identifying the classes appropriately except for few images. This is the reason why the heatmap is more concentrated along the diagonal.

ROC

We plotted the receiver operating characteristic curve (ROC) for the generated results and could observe the Area Under the Curve (AUC) values for all the classes are approximately the same indicating the fact that the model is successful in classifying all the classes (Figure 12). Below is the AUC value for each class:

- AUC for daisy class : 0.92
- AUC for dandelion class : 0.92
- AUC for rose class : 0.88
- AUC for sunflower class : 0.92
- AUC for tulip class : 0.94

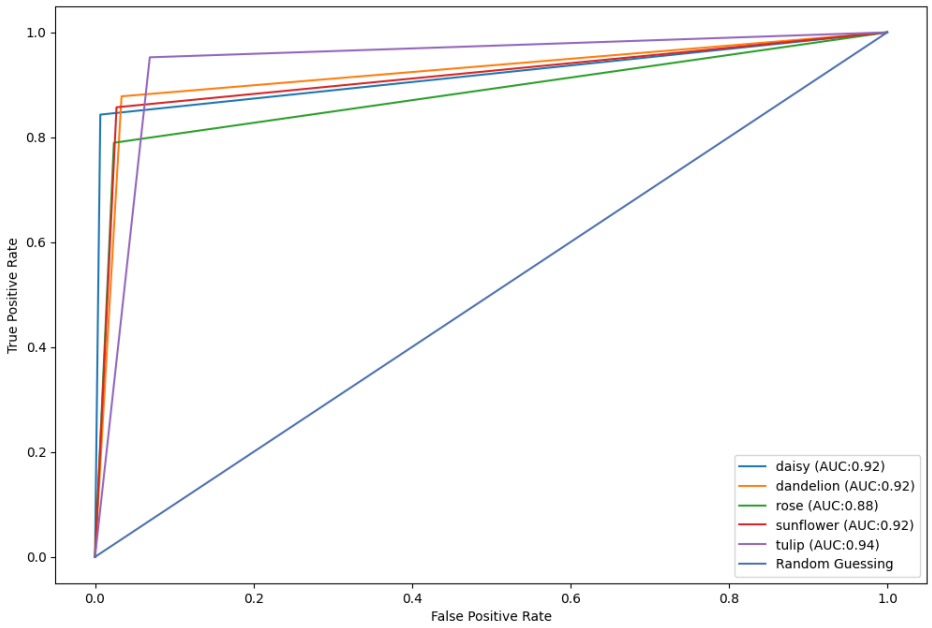


***Fig. 12**: ROC curve*

Gabor-Convolutional Neural Network

From Figure 13, we can observe that the graphs generated for Loss and Accuracy metrics by our model trained on images after applying gabor is similar to the graph in Figure 10, the training accuracy and loss are comparatively better than the validation, this could be due to the same aforementioned problem.

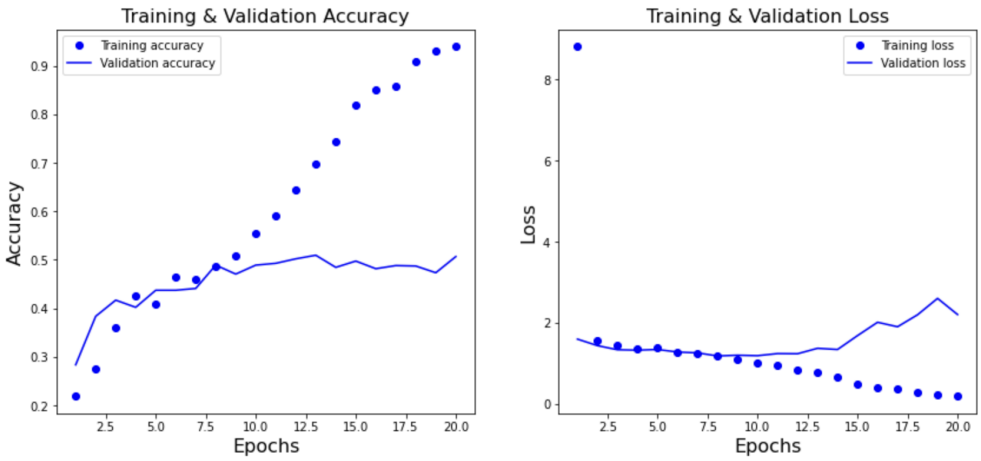
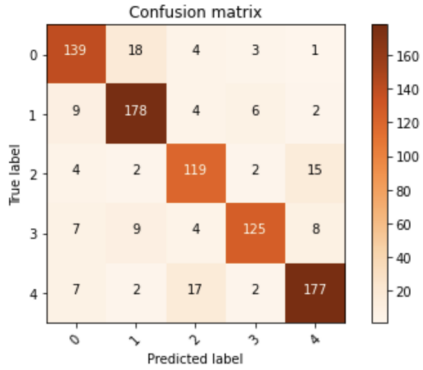


***Fig. 13**: Accuracy and Loss result over 20 epochs using gabor filter.*

```
from sklearn.metrics import confusion_matrix
confusion_mtx = confusion_matrix(y_pred, y_test)
plot_confusion_matrix(confusion_mtx, classes = range(5))
```



+ Code + Markdown

***Fig. 14**: Confusion Matrix after using Gabor Filter*

From the confusion matrix shown in Figure 14 above, we can say that the model's performance in correctly predicting the images from the testing dataset was mostly successful except in a few cases.

Web Application:

AWS setup, invoking AWS Rekognition API [6]

This requires creating an aws account, creating an IAM user, setting MFN authentication, setting credentials in AWS CLI. Screenshot provided in Figure 15.

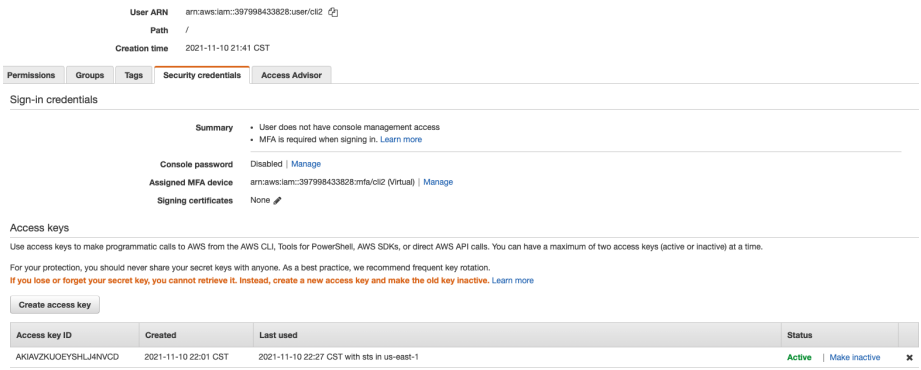


***Fig 15**: IAM User in AWS console.*

Once the credentials are set up, we will be able to access the AWS Rekognition API using the credentials to get the classification result of a flower image. Figure 16 and Figure 17 show the flower image used to invoke the API and response from API respectively.

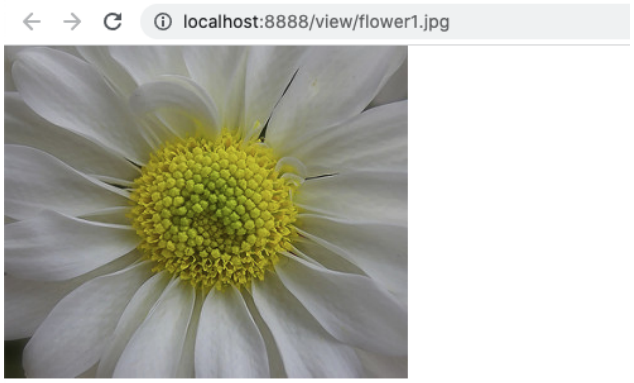


***Fig 16:** Flower used for invoking the AWS Rekognition API*

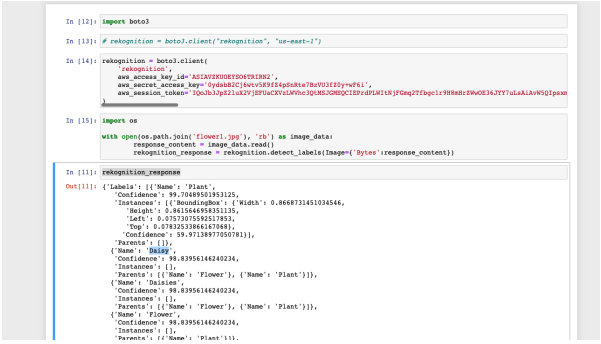


***Fig 17**: Response from AWS Rekognition API*

Flask App Setup [7]

- We will be using the flask app to demo the results. This website would allow users to upload an image, display results from trained models and AWS Rekognition API.
- So far sample hello world flask has been set up and will be adding user upload functionality, integrating the trained models, integration aws rekognition response.
- Figure 18 and Figure 19 show the flask sample helloworld setup and website running in localhost respectively.

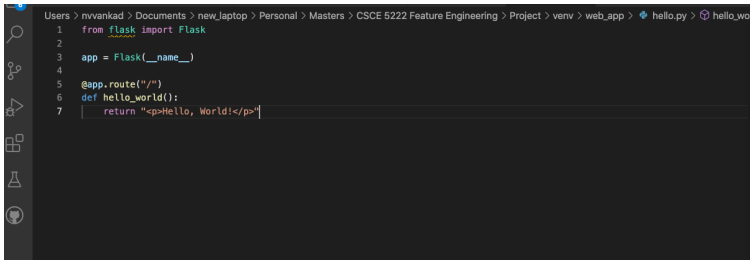


***Fig 18**: Sample hello world flask program*

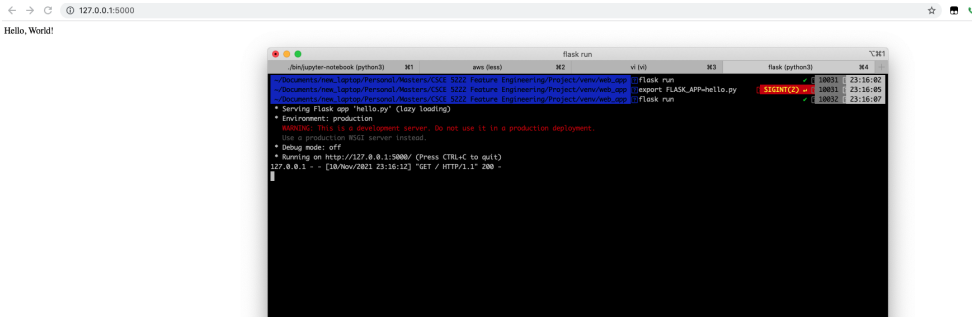


***Fig 19**: flask website running in local host*

Project Management

Work Completed so far:

Team Member	Responsibility	Contribution Percentage
Ngoc Phan	• Contributed to Related Work and Dataset sections • Wrote 2 literature reviews (Gaussian and Gabor) • Created visualizations for Dataset section • Trained 5 K-Means models to extract dominant colors for 5 flower types • Formatted and proofreaded the report for submission • Compiled the recording videos for submission	25%
Naga Sumanth Vankadari	• Research - project idea, available kaggle datasets, ability to fetch classes for flowers from aws rekognition. • Documentation - Created Architecture diagrams, Website Flask implementation plan • AWS CLI (Command Line Interface) setup and configuration of credentials. • Implement code for fetching rekognition response for a given image. • Flask app setup for sample hello world.	25%
Lakshmi Vandana Nunna	• Implemented logic to Iterate through the entire image dataset , resize each of them and create a dataframe with the image data and the classname(same as the folder name in which the image is present) and created a new column to represent label in numerical categorical form • Applied cv2 gaussian filter on the image data column and created a new “gauss” column that holds the transformed image data. • Created a neural network with convolution, max pooling, Dense & dropout layers with the final layer having a softmax activation function. • I configured Adam and sparse categorical cross entropy as the optimizer and loss functions. Accuracy is set as the metric for the model. • I implemented stratified split to generate train,validation and test sets on which i trained and tested the model	25%

	• Contributed to Idea description, Motivation & Significance, Goals & Objectives sections.	
Manoj Kolluri	• Wrote a literature review (sharpening CNN) • Applied cv2 gabor filter on the image data column and created a new “gabor” column that holds the transformed image data. • Created a Convolutional Neural Network and trained the gabor images using the training, validation and testing datasets • Created and plotted a confusion matrix to evaluate our models performance	25%

Work to be Completed:

Team Member	Responsibility	Issues or Concerns
Ngoc Phan	• Apply sharpening filter on the images • Train and evaluate a Sharpening-CNN model	
Naga Sumanth Vankadari	• Add functionality for image upload. • Integrate image upload, invoke aws rekognition API and display the class of flower. • Once an image is uploaded, pre-process the image, apply filters on the image. • Take the output of filters and load trained models to fetch the output of classification from 3 different models. • Display the results in flask app	
Lakshmi Vandana Nunna	• Fine tune the hyper parameters and model to avoid overfitting. • Perform data augmentation.	
Manoj Kolluri	• Apply Gabor filter on images • Train and Evaluate a Gabor CNN model	

References

[1] Chen, Yushi, et al. “Hyperspectral Images Classification with Gabor Filtering and Convolutional Neural Network.” *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 12, 2017, pp. 2355 - 2359. https://doi.org/10.1109/LGRS.2017.2764915.

[2] Mamaev, Alexander. “Flowers Recognition Dataset.” *Kaggle*, 2021, https://www.kaggle.com/alxmamaev/flowers-recognition. Accessed 31 October 2021.

[3] McGrath, Michael. “How to Pick Plants to Avoid Allergies in Your Pets.” *How Does Your Garden Mow*, 2021, https://www.howdoesyourgardenmow.com/pick-plants-avoid-allergies-pets/. Accessed 31 October 2021.

[4] Wang, Hongren, et al. “Gaussian Transfer Convolutional Neural Networks.” *IIET Comput. Vis., 2018, Vol. 12 Iss. 6, pp. 855-862 © The Institution of Engineering and Technology 2018* https://doi.org/10.1109/TETCI.2018.2881225.

[5] Jay Hoon Jung, Yousun Shin, YoungMin Kwon. “Extension of Convolutional Neural Network with General Image Processing Kernels” *Proceedings of TENCON 2018 - 2018 IEEE Region 10 Conference (Jeju, Korea, 28-31 October 2018)*. https://doi.org/10.1109/TENCON.2018.8650542

[6] Amazon. *Setup AWS Session token*. AWS CLI, 2021. *AWS CLI Setup*, https://awscli.amazonaws.com/v2/documentation/api/latest/reference/sts/get-session-token.html.

[7] Pallets. “Flask Setup.” *Flask Pallets*, 2010, https://flask.palletsprojects.com/en/2.0.x/