Flower Recognition

Project Report

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

Background

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.

Work Process Flow

Feature Extraction & Model Training

***Fig. 2**: Work Process Flow Diagram*

- In the preprocessing phase, images are converted to uniform size of 150*150.
- Each of this set of images are for training 3 independent models. Different image filters - Gaussian, Sharpening, Gabor to extract features for the image dataset and are passed as an input to the CNN models respectively.
- Standard metrics such as confusion matrix and accuracy are recorded.
- All 3 CNN model layers are concatenated to form a merged CNN model, input to this is all 3 filters passed in as list.
- All the trained models are saved as '.h5' files using keras save API, these are further loaded into the flask website to perform prediction.

Website Integration

***Fig. 3**: Website Integration*

In this phase, all the pieces are stitched together and results are displayed on the website in interactive fashion. Steps are outlined as below.

- When a user uploads an image, necessary preprocessing is performed on the image; as in model training such as resize to 150*150. This image is converted to np array representation. Filters are applied on the result - np array image. This filter is passed as input to the trained model to perform the prediction.

- Also, the uploaded image is passed as an input to the AWS rekognition API to get the classified result.
- Website is developed using the flask framework and deployed on a local host.
- Final results from - 5 different predictions (as outlined below) are displayed in the grid format in the website.

    1. Gabor filter CNN model result.
    2. Gaussian filter CNN model result.
    3. Sharpening filter CNN model result.
    4. Concatenated CNN model result.
    5. AWS Rekognition API result.

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 4 displays the data distribution for the five flower types.

***Fig. 4**: 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 5, 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. 5**: Frequency distribution of image pixels.*

Figure 6 displays scatter plots of Red/Green/Blue percentage per flower type. Based on figure 6a, 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 6b also indicates that most flowers have a blue percentage less than 50%.

(a)

(b)

Fig. 6*: 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 7 displays plots of the natural logarithm of Within-Cluster Sum of Square Error (WCSS) for each k value. Based on the plots in Figure 7, the best k values for daisy, dandelion, rose, sunflower, and tulip dataset are 3, 5, 3, 5, and 3, respectively.

Fig. 7: *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)

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 8, 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. 8**: Dominant colors for (a) daisy flowers, (b) dandelion flowers, (c) rose flowers, (d) sunflowers, and (e) tulip flowers.*

Model Development

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)
```

***Fig. 10**: Code snippet for Gaussian blur filter*

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(x,y, stratify=y,
                                                  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)))
```

```
Class frequency in Train Set:
[[  0 155]
 [  1 245]
 [  2 172]
 [  3 167]
 [  4 313]]
Class frequency in Test Set:
[[  0  51]
 [  1  82]
 [  2  57]
 [  3  56]
 [  4 105]]
Class frequency in Validation Set:
[[  0  51]
 [  1  82]
 [  2  58]
 [  3  55]
 [  4 105]]
```

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

Gabor-Convolutional Neural Network

We applied a stratified split to split the image data into training, testing and validation sets and then applied a cv2 Gabor Kernel on the image data and created new lists to hold these newly generated Sharp image data, the function that was used to create these features is given in the code snippet below.

Applying Gaussians Filter

```
[9]:
gaussx_train=[]
gaussx_val=[]
gaussx_test=[]


for image in x_train:
    gauss_feature = cv2.GaussianBlur(image, (3, 33), 0)
    gaussx_train.append(gauss_feature)

for image in x_val:
    gauss_feature = cv2.GaussianBlur(image, (3, 33), 0)
    gaussx_val.append(gauss_feature)

for image in x_test:
    gauss_feature = cv2.GaussianBlur(image, (3, 33), 0)
    gaussx_test.append(gauss_feature)
```

These features in the lists were then used as the x parameter while the labels generated by the stratified split were used as the y parameter which were then used to train, validate and test the model.

Sharpening-Convolutional Neural Network

We applied a stratified split to split the image data into training, testing and validation sets and then applied a custom made sharpening Kernel on the image data and created new lists to hold these newly generated Sharp image data, the function that was used to create these features is given in the code snippet below.

Applying Sharpning Filter

```
[7]:
sharpx_train=[]
sharpx_val=[]
sharpx_test=[]

skernel = np.array([[-1,-1,-1], [-1,10,-1], [-1,-1,-1]])

for image in x_train:
    sharp = cv2.filter2D(image, -1, skernel)
    sharpx_train.append(sharp)

for image in x_val:
    sharp = cv2.filter2D(image, -1, skernel)
    sharpx_val.append(sharp)

for image in x_test:
    sharp = cv2.filter2D(image, -1, skernel)
    sharpx_test.append(sharp)
```

These features in the lists were then used as the x parameter while the labels generated by the stratified split were used as the y parameter which were then used to train, validate and test the model.

Concatenated Model

For creating a concatenated model we first applied all three filters to the image data generated from stratified split and used all three feature maps generated from the images as x parameters while the labels generated by the stratified split were used as the y parameter, we then created a concatenated CNN model that has 3 input layers followed by convolutional layers and max pooling layers, merging at the flattening layer followed by Dense layers and a single output layer. The architecture of the model is given in the figure below.

***Fig. 11**: Architecture of concatenated model*

Model Results

Training & Validation Accuracy

Below are the figures that illustrate the training and validation accuracies of the 4 models over each epoch.

***Fig. 12**: Train & validation accuracy of CNN models*

By looking at the individual graphs we can observe that the Merged CNN Model has the least amount of difference between its validation and training accuracies.

Training & Validation Loss

Below are the figures that illustrate the training and validation loss of the 4 models over each epoch.

(a) Gabor-CNN

(b) Gaussian-CNN

(c) Sharpening-CNN

(d) Merged-CNN

***Fig. 13**: Training & validation loss of CNN models.*

By looking at the individual graphs we can observe that the Merged CNN Model has the least amount of difference between its validation and training loss.

Confusion Matrix

Below are the figures illustrating the confusion matrices of all the 4 models that we have trained and tested. By analyzing the performance of these models based on the confusion matrix layout, we could arrange the models as below based on their performance:

Merged CNN > Sharp CNN > Gaussian CNN > Gabor CNN

***Fig. 14**: Confusion matrix of CNN models.*

Overall Accuracy

Figure 15 displays the screenshots illustrating the final accuracies of all the four models. Based on the output results, Gabor-CNN model seems to give the worst performance, and Merged-CNN model seems to give the best performance. However, the test accuracy for Gaussian-CNN, Sharp-CNN, and

Merged-CNN are very closed. All three aforementioned models have accuracy close to 87% if we round up the results.

Test Accuracy

- **Gabor CNN** : 0.8384
- **Gaussian CNN** : 0.8648
- **Sharp CNN** : 0.8664
- **Merged CNN** : 0.8687

```
gabormodel.evaluate(gaborx_test,y_test)
```

```
864/864 [==============================] - 1s 632us/step

[0.37113827687722667, 0.8384259298995689]
```

(a) Gabor-CNN

```
gaussmodel.evaluate(gaussx_test,y_test)
```

```
864/864 [==============================] - 1s 634us/step

[0.32439251133689173, 0.8648148267357437]
```

(b) Gaussian-CNN

```
sharpmodel.evaluate(sharpx_test,y_test)
```

```
864/864 [==============================] - 1s 605us/step

[0.3116294416012587, 0.8664351856267011]
```

(c) Sharpening-CNN

```
model2.evaluate([gaussx_test,gaborx_test,sharpx_test],y_test)
```

```
864/864 [==============================] - 2s 2ms/step

[0.2997655145548008, 0.8687500026490953]
```

(d) Merged-CNN

***Fig. 15**: Overall accuracy of CNN models.*

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 is provided in figure 16.

***Fig 16**: 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. Figures 17 and 18 show the flower image used to invoke the API and response from API respectively.

***Fig 17:** Flower used for invoking the AWS Rekognition API*

***Fig 18**: Response from AWS Rekognition API*

Flask App Setup [7]

- We have used 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.
- AWS Rekognition API code using boto3 python client as shown above is integrated into flask website backend. Result from the aws rekognition API - class of the flower and confidence values are displayed on the image in the grid format. Place holders for other models are also added into the grid.
- Validation mechanisms for file extension are done when the file is uploaded. Only image formats - jpg, jpeg, png format files are allowed to be uploaded.
- Figures 19 and 20 show the flask code implementations, website folder setup respectively.

***Fig 19**: flask code - file handling*

***Fig 20**: flask structure setup*

Website Results

Launch Website in Localhost

***Fig 21**: Initiate flask website*

Home page looks as below (figure 22), clicking on the upload page redirects users to the upload page, where the user is allowed to select and upload a file (figure 23).

***Fig. 22**: Website Home page*

***Fig. 23**: Upload file page*

Prediction Results

Screenshots below (Fig 24a - 24d) shows predicted results from trained models

Model	Predicted Flower Class	Probability
Gaussian Filter CNN	Dandelion	0.9573889970779419
Gabor Filter CNN	Dandelion	0.9605498909950256
Sharpening Filter CNN	Dandelion	0.9979783892631531
Merged Model CNN	Dandelion	0.9670974612236023
Amazon Rekognition API	Dandelion	0.89904

***Fig. 24a**: Classified result of Dandelion Image*

Model	Predicted Flower Class	Probability
Gaussian Filter CNN	Sunflower	0.988854169845581
Gabor Filter CNN	Sunflower	0.8464449644088745
Sharpening Filter CNN	Sunflower	0.9742432832717896
Merged Model CNN	Sunflower	0.8552935719490051
Amazon Rekognition API	Sunflower	0.960577

***Fig. 24b**: Classified result of Sunflower Image*

Model	Predicted Flower Class	Probability
Gaussian Filter CNN	Sunflower	0.4323742985725403
Gabor Filter CNN	Daisy	0.4821966886520386
Sharpening Filter CNN	Dandelion	0.46889781951904297
Merged Model CNN	Sunflower	0.430497944355011
Amazon Rekognition API	Daisy	0.988396

***Fig. 24c**: Classified result of Daisy Image*

Model	Predicted Flower Class	Probability
Gaussian Filter CNN	Tulip	0.9948581457138062
Gabor Filter CNN	Tulip	0.989429771900177
Sharpening Filter CNN	Tulip	0.99793940782547
Merged Model CNN	Tulip	0.7826806902885437
Amazon Rekognition API	Tulip	0.94513

***Fig. 24d**: Classified result of Tulip image*

Project Management

Work Completed

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 website. Integrate amazon rekognition results into a website. • Research on how to save the keras model and load the model in the flask website. • While predicting a single image using a saved model, input has to be converted to an np array list as model input is a list of images. • Apply a gabor filter, invoke trained models and get the class and probability. • Similarly apply gaussian filter, sharpening filter invoke respective models. • Provide all 3 models as a list, invoke a concatenated CNN model and get classification results.	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	25%

	• 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 • Helped with debugging to fix the issues while training the models • Created the project presentation and contributed to report documentation	
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 • Applied sharpening filter on the image data column and created a new “sharp” column that holds the transformed image data. • Created a Convolutional Neural Network and trained the gaussian images using the training, validation and testing datasets • Created a Convolutional Neural Network and trained the sharp images using the training, validation and testing datasets • Created a Concatenated Convolutional Neural Network and trained the all three features of images using the training, validation and testing datasets • Created and plotted a confusion matrix to evaluate our models performance • Helped with debugging to fix the issues in the user interface • Saved the trained models after retrieving best weights during training given by callback functions so that it can be used to create UI	25%

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/