
Lab Assignment: 1

Full name: Gardyan Priangga Akbar

Student number: 7695883

Email address: gardyanakbar99@gmail.com

Flower Image Recognition by KNN, MLP and CNN

This lab assignment aims to compare and analyze the various image classification approaches, namely KNN, MLP, and CNN. The dataset used for this assignment is a dataset consisting of over 4000 images of flowers spread across 5 different classification labels (daisy, dandelion, rose, sunflower, and tulip). This assignment was conducted using a machine with the following specifications:

Processor: Intel i7-9700K

Graphics Card: Nvidia RTX 2070

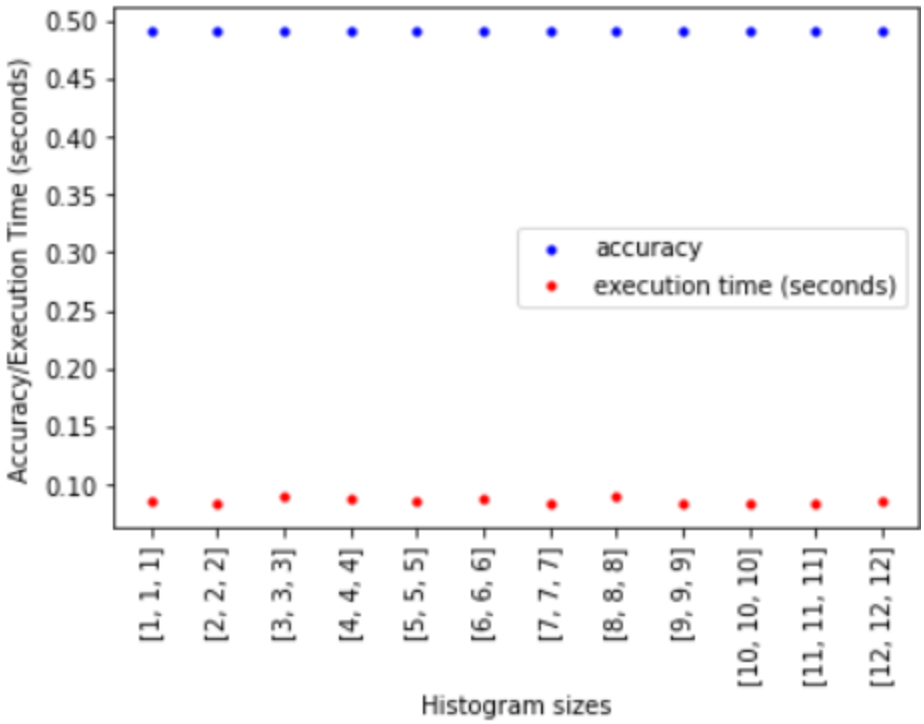
RAM: 32 GB

KNN

1. The python modules used for this task are as follows:

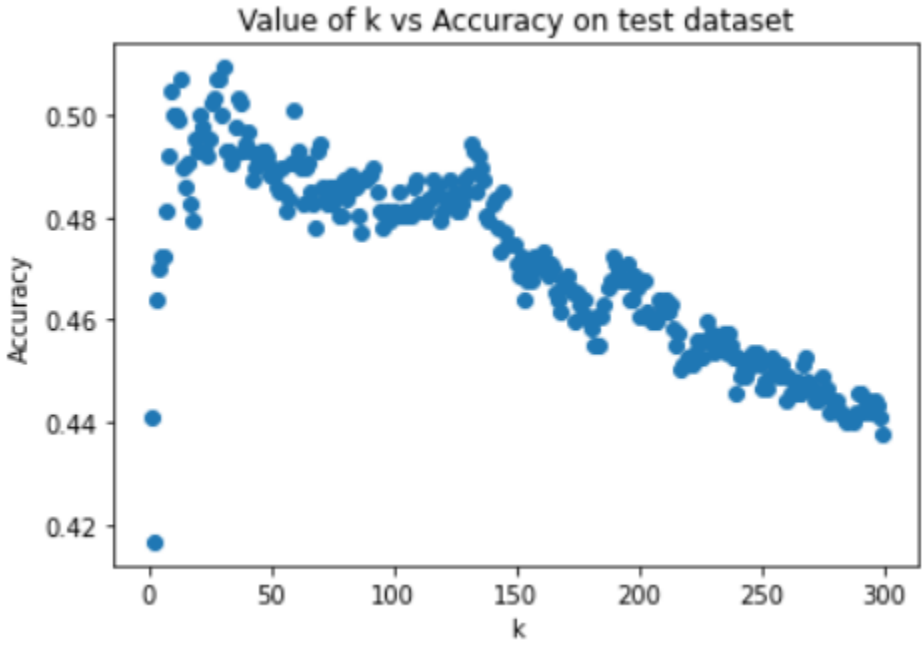
Module	Usage
Pandas	To convert the resulting confusion matrix into a DataFrame alongside its proper labels
OpenCV	To load the flower images and generate the color histogram
NumPy	To convert the images into input for KNN
Matplotlib	To visualize the color histogram, images, and confusion matrix
Sklearn	To split the dataset into training, validation, and test sets, generate confusion matrix, and perform KNN
Seaborn	To generate a heatmap of the confusion matrix
os	To get list of files from a directory
time	To track the time taken to evaluate the test set

2. Changing the size of the color histogram lower or higher than the default [6, 6, 6] does not seem to display any significant changes to performance of the model on the test set.



As shown from the graph above, the changes to the execution time is miniscule. The changes are even less visible to the accuracy of the model. This means despite the input size or the number of features, KNN is able to deliver similar performance.

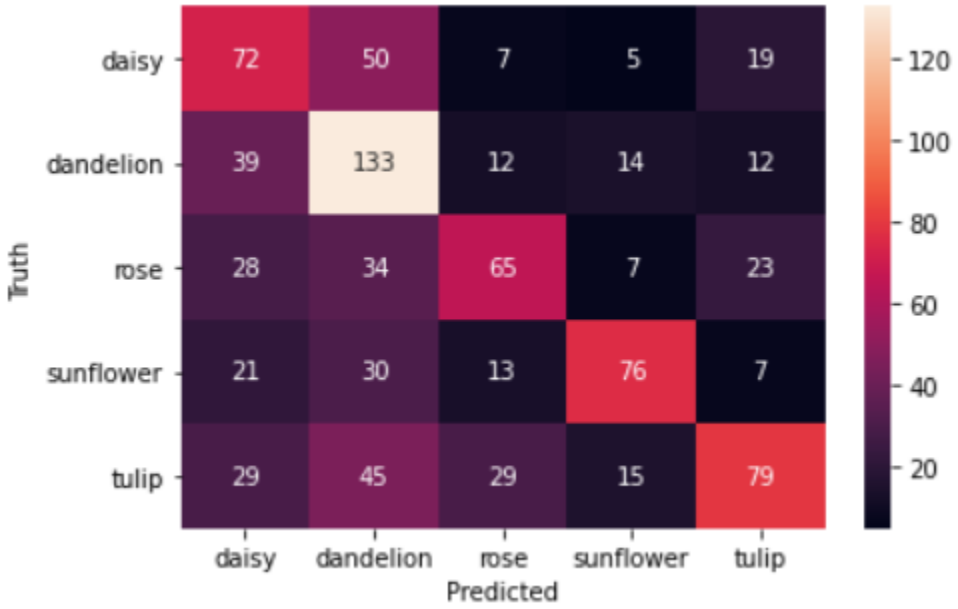
3. I created a list of k to test and then compare them to see which one has the highest accuracy on the validation dataset. Plotting the results visualizes where the value of k is best.



I did not go beyond 300 for the value of k as the accuracy seem to continue going downhill as shown by the graph. By further utilizing Python's max() and index() functions I was able to

determine the best value for k is 31, with it having the highest accuracy score of 0.5092592592592593 on the validation dataset.

4. The classification accuracy of the KNN model on the test set was 0.49189814814814814.

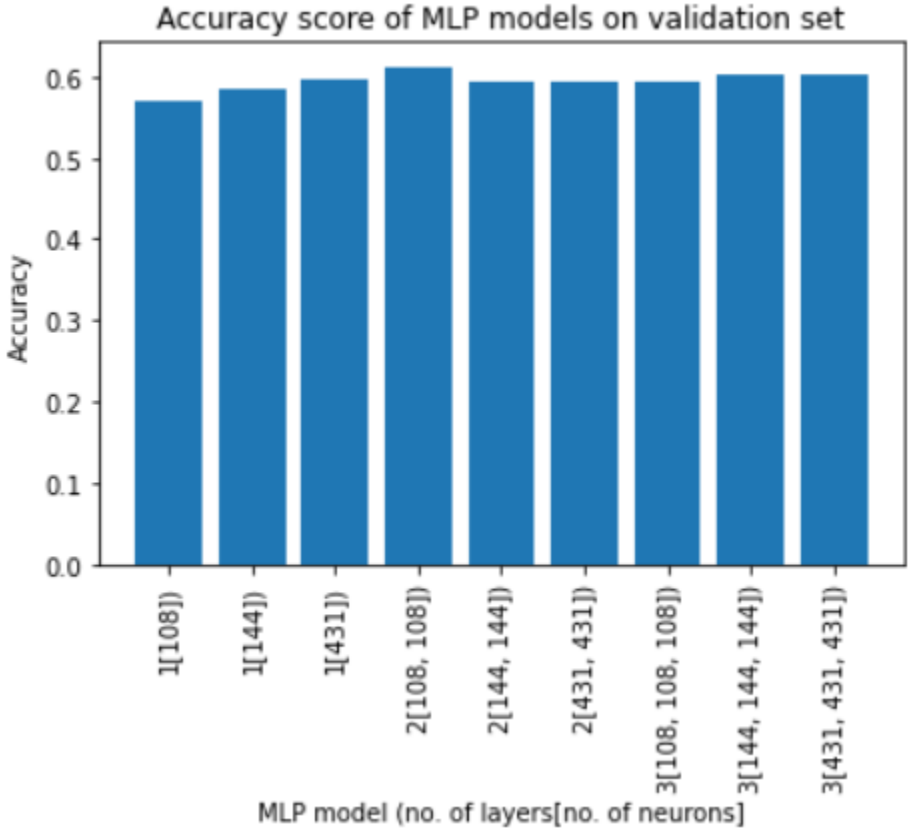


According to the confusion matrix, the model struggles the most in classifying daisy as dandelion.

5. It took 0.08403491973876953 seconds for the KNN classifier to classify all the samples in the test set.

MLP

1. I use the MLPClassifier class provided by sklearn library to create the MLP models. I can specify the number of hidden layers and the number of neurons in each hidden layers through the class' constructor by specifying the hidden_layer_sizes property. For example, if I wanted to have 3 hidden layers with 512 neurons each I would pass in the value [512, 512, 512] into that attribute. I then utilized the fit() method from the MLPClassifier class to train the model and its predict() method to make predictions on the test set.
2. The 9 MLP structures I designed have different amounts of layers and neurons in each layers following the requirements specified in the instructions document. Implementing them is simply creating a list of the number of layers (1 to 3) and another list containing the number of neurons. I then iterate through both lists using a nested for loop, all the while creating a new MLPClassifier object at every iteration. Every model uses early stopping to stop training if the the training does not improve after 10 epochs.
3. To find which of the nine models was the best, I used the score() method that comes with the MLPClassifier class on the validation set that would return the accuracy score. The results are graphed below:

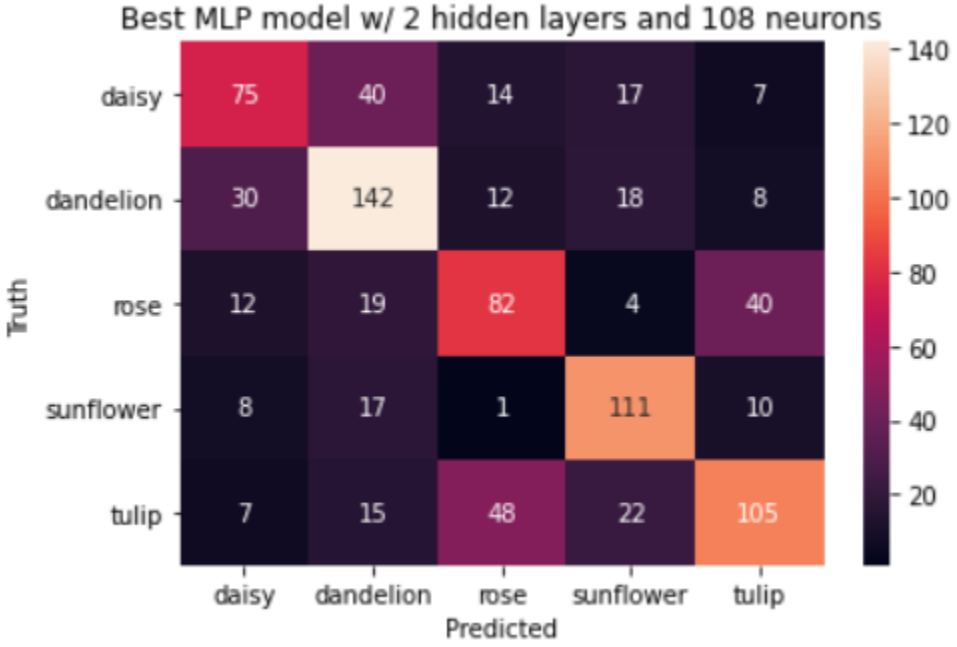


By utilizing Python's built-in max() and index() functions I can get the best model to be the MLP with 2 layers at 108 neurons in each layer. This model has the highest accuracy score of 0.6122685185185185.

4.

1) 0.6006944444444444

2) The model confuses tulips with roses the most



5.

1) 3.439999580383301 seconds
2) 0.008000373840332031 seconds

CNN

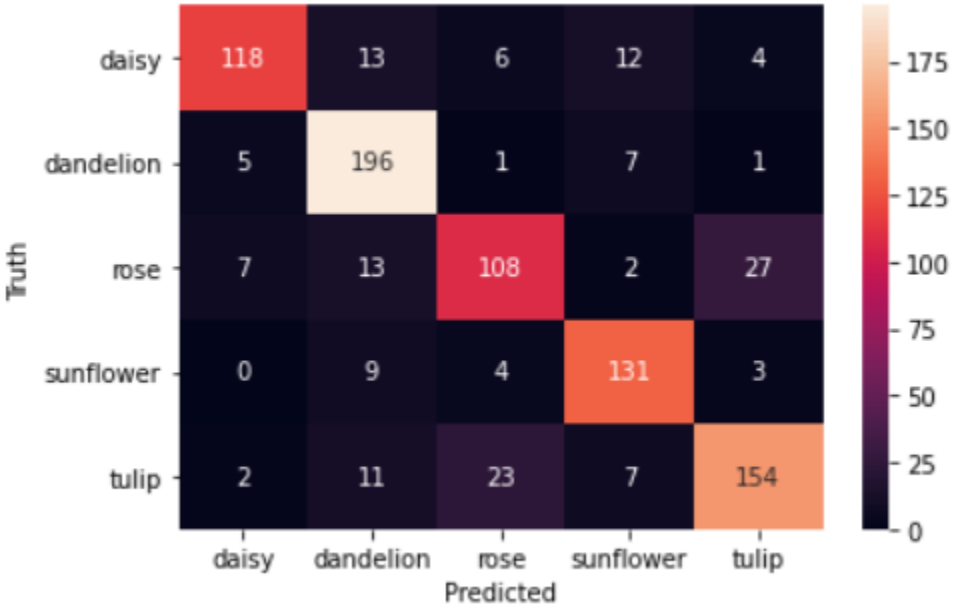
1. To build and train the CNN classifier several key functions from Keras was used. The add() and compile() functions were used to append different types of layers into the sequential model and to compile the model with the selected optimizer and loss function and metrics to track for, respectively. To train the CNN model, I used the fit_generator() function in which I specified the training and validation dataset, the amount of epochs, and the number of steps per epoch.
2. The CNN model architecture are as follows:

```
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding="Same", activation="relu", input_shape = (150, 150, 3)))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(filters = 64, kernel_size = (3, 3), padding="Same", activation="relu"))
model.add(MaxPooling2D(pool_size = (2, 2), strides=(2, 2)))

model.add(Conv2D(filters = 96, kernel_size = (3, 3), padding="Same", activation="relu"))
model.add(MaxPooling2D(pool_size = (2, 2), strides=(2, 2)))

model.add(Conv2D(filters = 96, kernel_size = (3, 3), padding="Same", activation="relu"))
model.add(MaxPooling2D(pool_size = (2, 2), strides=(2, 2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation("relu"))
model.add(Dense(5, activation = "softmax"))
```

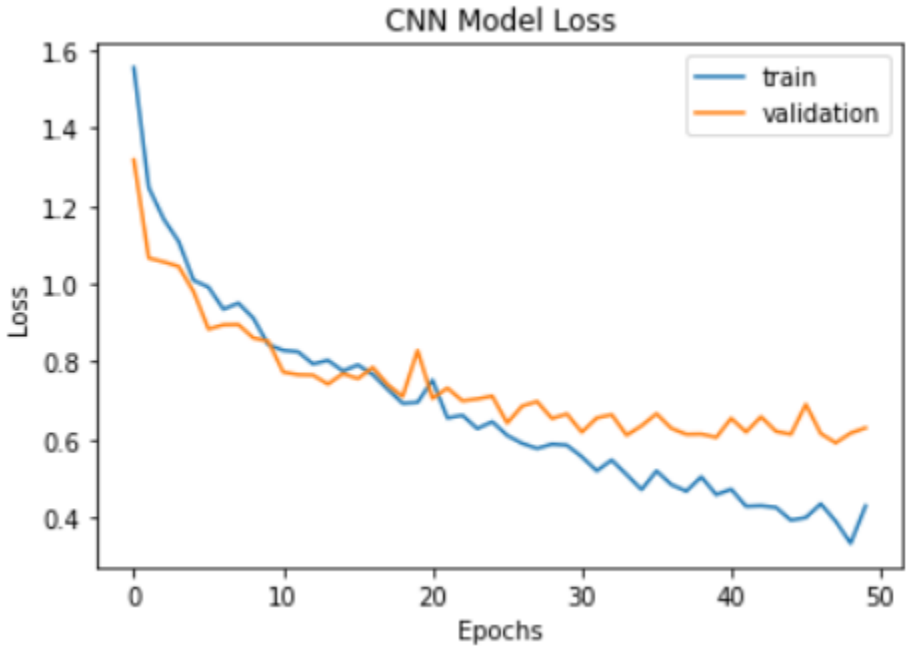
   The Conv2D layers are used to perform convolutions on the input image, that is to detect and extract features. The MaxPooling2D layers are used to summarize the features identified by the convolution layer. It takes the most prominent features from each region of the feature map where the filters from the convolution layer was applied. The Flatten reshapes the output of the previous layer into a 1-dimensional tensor so that it can be passed into the Dense layer. The Dense layer is a fully-connected layer that connects each of its neurons with its preceding layer and performs matrix multiplication. The final Dense layer is the output layer using the softmax activation function to return the probability of each class (every other layer uses the ReLU activation function).
3.

   1) 0.8182870149612427

2) The CNN model confuses between tulip and rose the most

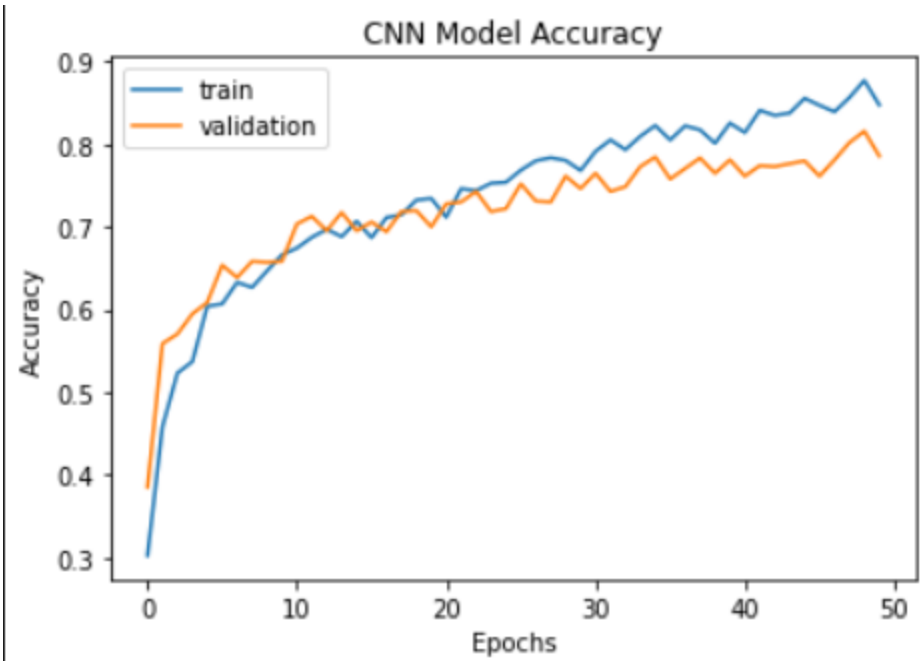


4. Changes in loss:



Loss decreases for both training and validation sets as the number of epochs increases. It can also be observed that the difference in the value of loss between the two sets gets larger as the number of epochs increases.

Changes in accuracy:



Accuracy increases for both training and validation sets as the number of epochs increases. As the draws closer to 50 epochs, the training set came close to reaching 90% accuracy, while the validation set was able to achieve around 80%. Similar to that observed in the changes in loss, the difference of accuracy between the two sets increases as the number of epoch increases.

5.

1) 573.3135676383972 seconds (about 10 minutes)
2) 0.6942403316497803 seconds