Project 2

Neural network

Report

Flower Classification

Introduction

Project description:-

we have the petals to metals dataset, which is a dataset for flowers We're classifying 104 types of flowers based on their images drawn from five different public datasets. Some classes are very narrow, containing only a particular sub-type of flower (e.g., pink primroses) While other classes contain many sub-types

Description of the Petals to Metals dataset:-

The Petals to Metals dataset is a collection of images of various flowers used for flower classification. The dataset consists of images from different flower species, with each image labeled according to its corresponding flower class. The dataset aims to provide a diverse range of flower images to enable accurate classification.

Objectives:-

1. Become familiar with CNNs models and transfer learning.
2. Applying fine-tuning and analyzing the performance of different classifiers on a large image dataset.

(1)DataSET

a) **First** we will Download the dataset
b) **functions** that loads an image and visualize some images from different classes

Visualizing random images:

The **visualize_images()** function is defined to randomly select images from different flower classes and display them.

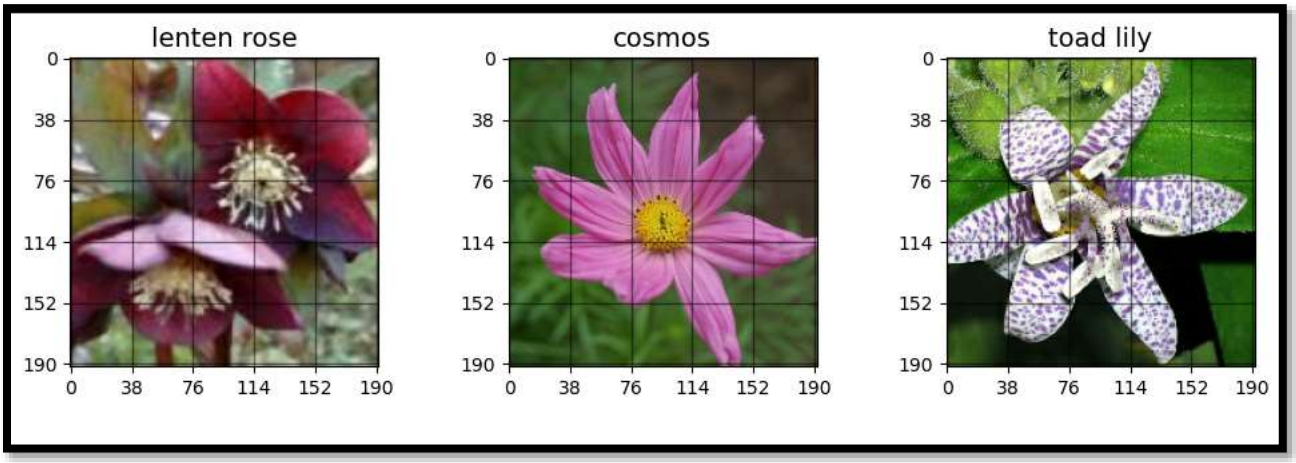
The function takes the training path, the list of classes, and an optional parameter for the number of images to display (default is 9).

It uses matplotlib to create a grid of subplots, where each subplot shows an image and its corresponding class name.

A random class and image are selected, and the image is loaded and displayed using **plt.imshow().**

The class name is shown as the subplot title, and the grid is customized with axes and gridlines.

The function is called to visualize the first image from 9 randomly chosen flower classes.



Displaying the number of samples per class:

The display_samples_per_class() function is defined to count and display the number of samples in each flower class.

It iterates through the CLASSES list and obtains the class path by joining the train path with the index of the class.

The number of samples is determined by counting the files in the class path using len(os.listdir(cls_path)).

The function prints the class name and the corresponding number of samples.

The function is called to display the number of samples in each flower class.

```
Number of samples for class  pink primrose====> 272
Number of samples for class  hard-leaved pocket orchid====> 26
Number of samples for class  canterbury bells====> 20
Number of samples for class  sweet pea====> 21
Number of samples for class  wild geranium====> 703
Number of samples for class  tiger lily====> 87
Number of samples for class  moon orchid====> 18
Number of samples for class  bird of paradise====> 105
Number of samples for class  monkshood====> 87
Number of samples for class  globe thistle====> 84
Number of samples for class  snapdragon====> 136
Number of samples for class  colt's foot====> 43
Number of samples for class  king protea====> 92
Number of samples for class  spear thistle====> 263
Number of samples for class  yellow iris====> 227
Number of samples for class  globe-flower====> 21
Number of samples for class  purple coneflower====> 55
```

(2)Split Data and Data Preprocessing

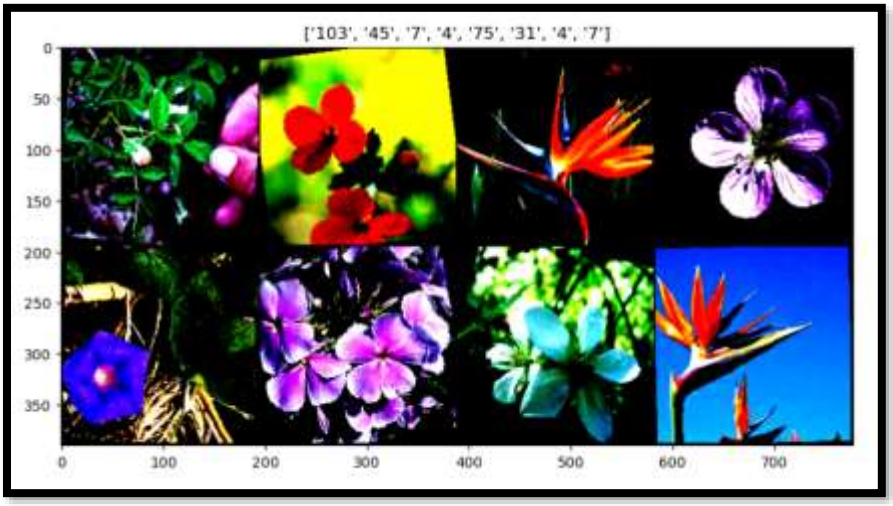
1) Split the Training Dataset:-

The training dataset is split into a training subset (90% of the data) and a validation subset (10% of the data) using the **random_split** function from the **torch.utils.data** module. The sizes of the subsets are determined based on the length of the **train_dataset.**

2) data preprocessing:-

[Image Transformations] image transformations to be applied to the dataset. These transformations include resizing the images to (192, 192) pixels, random horizontal flipping, random rotation by 15 degrees, color jittering, converting the images to tensors, and normalizing the pixel values using mean and standard deviation.

After making Data splitting and data processing



(3) The Build CNN Models

1st The model Flower Classification with CNN Model Training and Validation

CNN Model Architecture:

The CNN model is designed, comprising convolutional, activation, max pooling, and fully connected layers.
The model architecture enables effective feature extraction and classification.
Model Initialization and Device Setup:

The CNN model is initialized with the appropriate number of output classes.
If available, the code sets the device to GPU for accelerated computation; otherwise, it uses the CPU.
Loss Function and Optimizer:

The cross-entropy loss function is employed to measure the model's performance.
The Adam optimizer is utilized to optimize the model's parameters.
Training and Validation Loop:

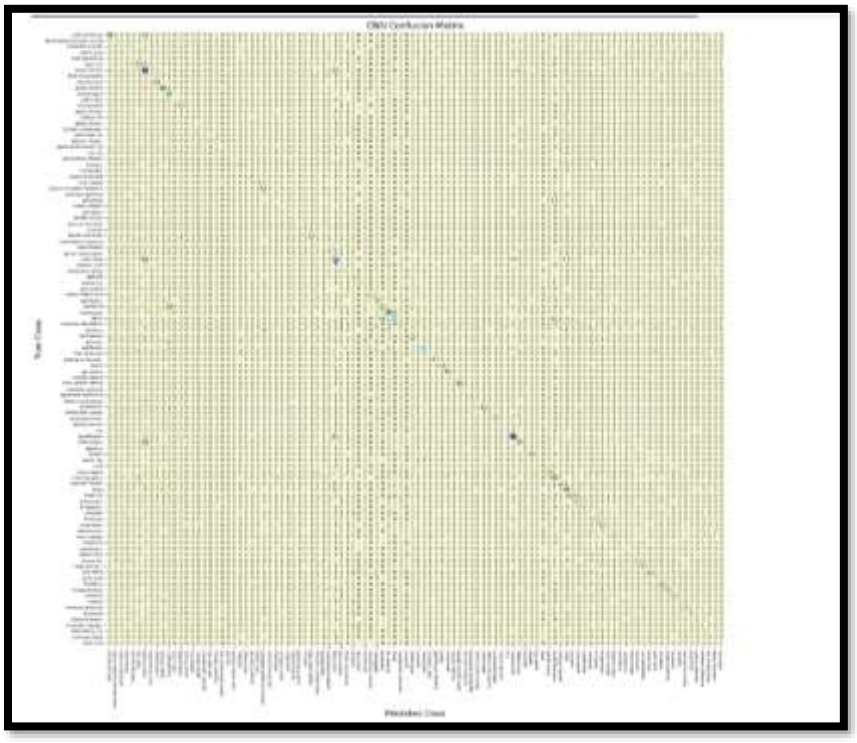
The code iterates over a specified number of epochs, comprising training and validation phases.
During the training phase, the model learns from the training dataset, calculates loss, and updates its parameters.
The validation phase evaluates the model's performance on the validation dataset, measuring accuracy and loss.
Metrics, including training and validation loss/accuracy, are recorded for analysis.
Model Saving:

After the training process, the trained model's parameters are saved for future usage and inference.
In summary, the CNN model for flower classification achieved the following results after training:

Epoch: 15/15
- Train Loss: 1.1463
- Train Accuracy: 67.39%
- Validation Loss: 2.5358

The model demonstrated a satisfactory level of accuracy during training, with a training accuracy of 67.39%. However, when evaluated on the validation set, the model's accuracy dropped to 41.69%. This suggests that the model may be overfitting to the training data and struggling to generalize well to unseen data.

Here’s the confusion matrix for the model that we created:



And here’s the confusion matrix as a table to be more readable

Note: we just sniped it you can see it clearer on the Notebook

	pink primrose	hard-leaved pocket orchid	canterbury bells	sweet pea	wild geranium	tiger lily	moon orchid	bird of paradise	monkshood	globe thistle	snapdragon
pink primrose	35	0	0	0	0	1	13	0	0	5	0
hard-leaved pocket orchid	0	4	0	0	0	0	0	0	0	0	0
canterbury bells	0	0	5	0	0	1	2	0	0	0	0
sweet pea	0	0	1	0	0	2	0	0	0	0	0
wild geranium	0	0	0	0	0	2	1	0	0	0	0
tiger lily	0	0	0	0	1	34	5	0	2	1	0
moon orchid	3	0	1	1	0	1	99	0	0	7	4
bird of paradise	0	0	0	0	0	0	0	2	0	0	0
monkshood	0	0	0	0	0	0	1	0	18	0	0
globe thistle	1	0	0	0	0	0	2	0	2	50	0
snapdragon	0	0	0	0	0	3	0	0	0	5	33

Here's evaluating the model and the scores and the most confusing classes:

```
Accuracy: 0.4033
Macro F-Score: 0.3353
Most confused class: sweet pea
```

We opted to utilize two renowned CNN architectures, namely ResNet and GoogLeNet, for our project.

Regarding the **ResNet and GoogLeNet models,** the following steps were undertaken**:**

1. **We loaded the pre-trained ResNet and GoogLeNet models.**
2. **The last fully connected layer was modified to accommodate the specific number of flower classes for both models.**
3. **Loss function and optimizer were defined for both models.**
4. **Empty lists were initialized to store the training loss, validation loss, training accuracy, and validation accuracy histories for both models.**
5. **A training loop was implemented, encompassing both the training phase and the validation phase for both models.**

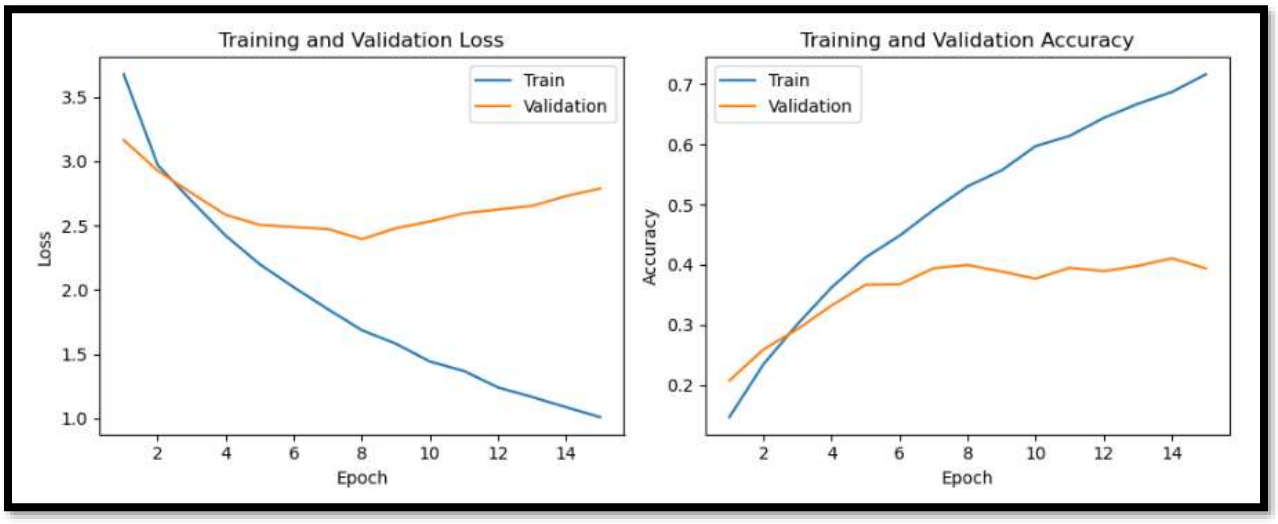
Following that, the models were ensembled, combining the **ResNet and GoogLeNet models, and the following actions were taken:**

Paths were defined for the pre-trained model weights.
The pre-trained GoogLeNet and ResNet models were loaded.
Variables were initialized to facilitate ensemble predictions.

An iteration was performed over the test dataset, comprising the following steps: a) Images were passed through the GoogLeNet and ResNet models. b) Ensemble predictions were calculated.
The ensemble accuracy on the test dataset was computed.

```
Ensemble Accuracy on Test Data: 0.9054
```

Plotting the training/validation loss and accuracy for the My model:-

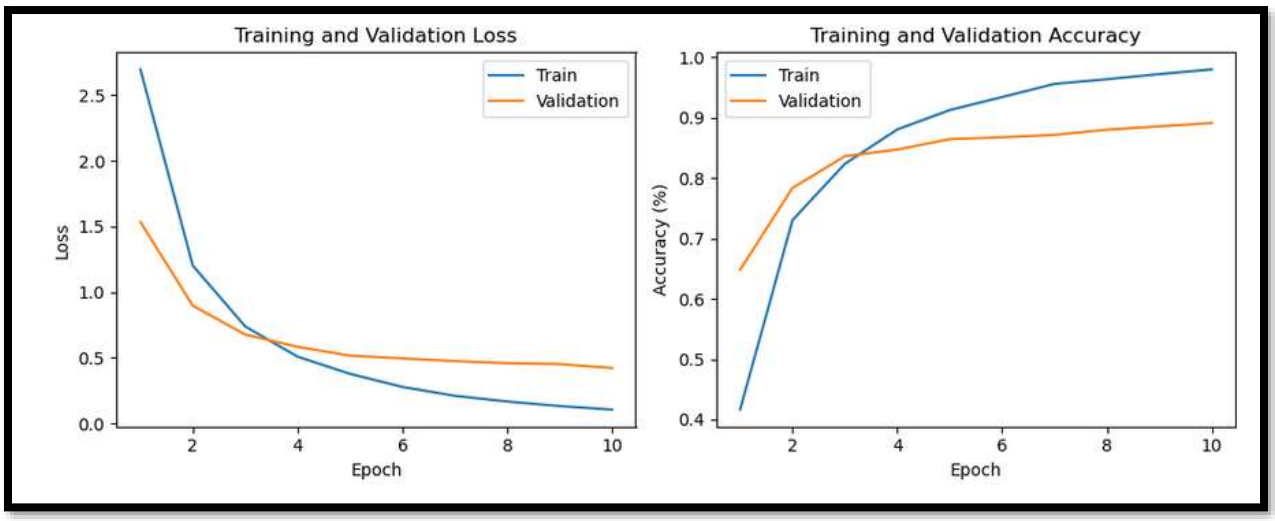


➔As we move forward, the train loss decreases gradually, indicating that the model is learning and becoming more efficient at minimizing the error while The accuracy also increases steadily, showing that the model's predictions align better with the ground truth labels.

➔ The validation loss decreases from Epoch 1 to Epoch 14, suggesting that the model generalizes better over time. The validation accuracy also demonstrates improvement, rising over in each epochs.

➔ the graph illustrates a positive learning curve for the model as the number of epochs increases. Both the training and validation metrics exhibit a consistent improvement, with the train loss decreasing and the accuracy increasing. This indicates that the model is gradually learning and becoming more effective at making accurate predictions.

Plotting the training/validation loss and accuracy for RESNET Model:-

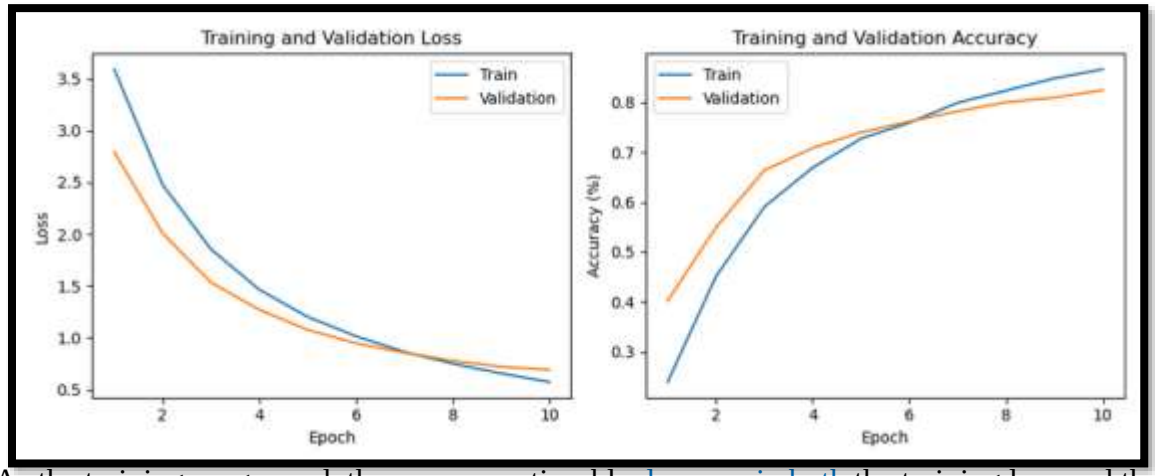


➔The training loss consistently decreases while the training accuracy increases Similarly, the validation loss decreases while the validation accuracy increases

➔These results indicate that the model progressively improves its predictive capabilities with each epoch. The decreasing loss values suggest that the model becomes more accurate in its predictions, while the increasing accuracy values indicate that it is better at correctly classifying the data.

➔Overall, the graph demonstrates successful training, where the model learns from the training data and generalizes well to the unseen validation data. These results are encouraging and suggest that the model has the potential to perform well on similar datasets.

Plotting the training/validation loss and accuracy google net model:-



➔As the training progressed, there was a noticeable decrease in both the training loss and the validation loss. This indicates that the model's ability to make accurate predictions improved over time.

➔ the accuracy of the model also increased steadily during the training process, reaching a high on the training set and on the validation set by the final epoch. This suggests that the model was able to generalize well and perform consistently on unseen data.

➔ The graph demonstrates the effectiveness of the training process in improving the model's performance. With further iterations and fine-tuning, it is likely that even better results could be achieved.

Compute the accuracy and Macro F-Score for each model:-

We evaluate the performance of pre-trained ResNet and GoogLeNet models on a test dataset. And measure their accuracy and calculate the macro F-score for both models.

The code begins by setting the device to CUDA if available; otherwise, it uses the CPU for computation. It initializes variables to keep track of ResNet and GoogLeNet model performance, including the number of correct predictions, total predictions, and prediction lists.

Next, it iterates over the test dataset using a **torch.no_grad()** context to disable gradient calculation. For each batch of images and labels, the code performs the following steps:

1. Passes the images through the ResNet model and obtains the predicted labels.
2. Updates the ResNet predictions, correct predictions count, and total predictions count.
3. Passes the images through the GoogLeNet model and obtains the predicted labels.
4. Updates the GoogLeNet predictions, correct predictions count, and total predictions count.

After iterating over the test dataset, the code calculates the accuracy and macro F-score for both models. The accuracy is computed by dividing the number of correct predictions by the total number of predictions. The macro F-score is calculated using the **f1_score** function from the **sklearn.metrics** module, considering the test dataset targets and the corresponding predictions from each model.

Finally, the code prints the accuracy and macro F-score values for the ResNet and GoogLeNet models on the test dataset. These metrics provide insights into the performance of the models in terms of overall accuracy and their ability to capture the diversity of classes in the dataset.

Then Calculate the accuracy and the f-score for each model and the most confusion classes

```
ResNet Accuracy on Test Data: 0.8971
ResNet Macro F-Score on Test Data: 0.8815
ResNet Most Confused Classes: (68, 68)
GoogLeNet Accuracy on Test Data: 0.8521
GoogLeNet Macro F-Score on Test Data: 0.7593
GoogLeNet Most Confused Classes: (68, 68)
```

Plot the confusion matrices and find the most confusing classes:

We plot it as a table using IPython.display to be more clear and readable

The GoogleNet Confusion Matrix

GoogLeNet Confusion Matrix

	pink primrose	hard-leaved pocket orchid	canterbury bells	sweet pea	wild geranium	tiger lily	moon orchid	bird of paradise	monkshood	globe thistle	snapdragon	colt's foot	king protea	spear thistle	yellow iris	globe-flower	purple coneflower	pe
pink primrose	68	0	0	0	0	0	2	0	0	3	0	0	0	0	0	0	0	
hard-leaved pocket orchid	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
canterbury bells	0	0	22	0	0	0	3	0	0	1	0	0	0	0	0	0	0	
sweet pea	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
wild geranium	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	
tiger lily	1	0	3	0	0	42	10	0	0	2	2	0	0	0	0	0	0	
moon orchid	12	0	0	0	0	0	149	0	0	15	0	0	0	0	0	0	0	

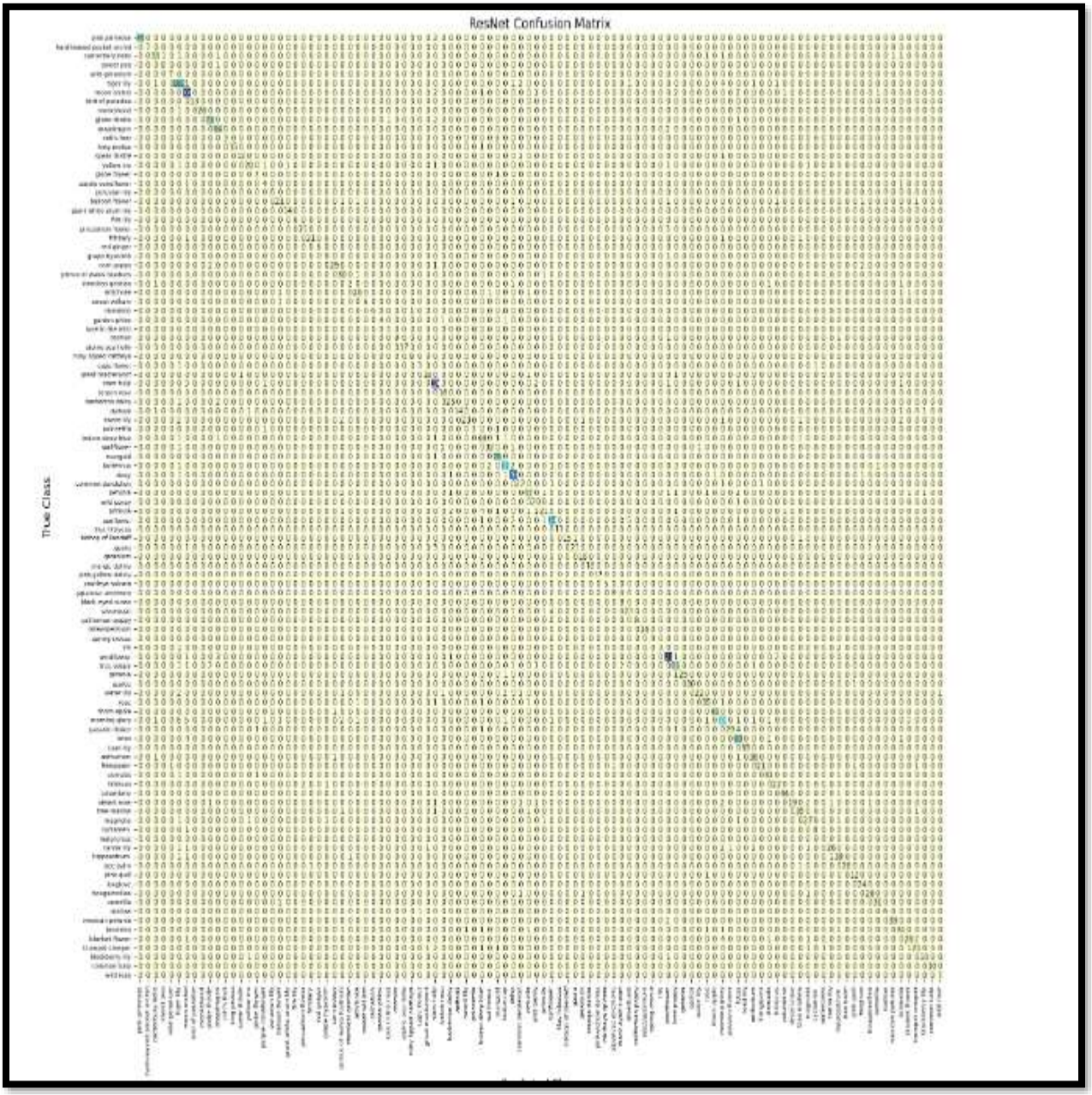
The ResNet Confusion Matrix

ResNet Confusion Matrix

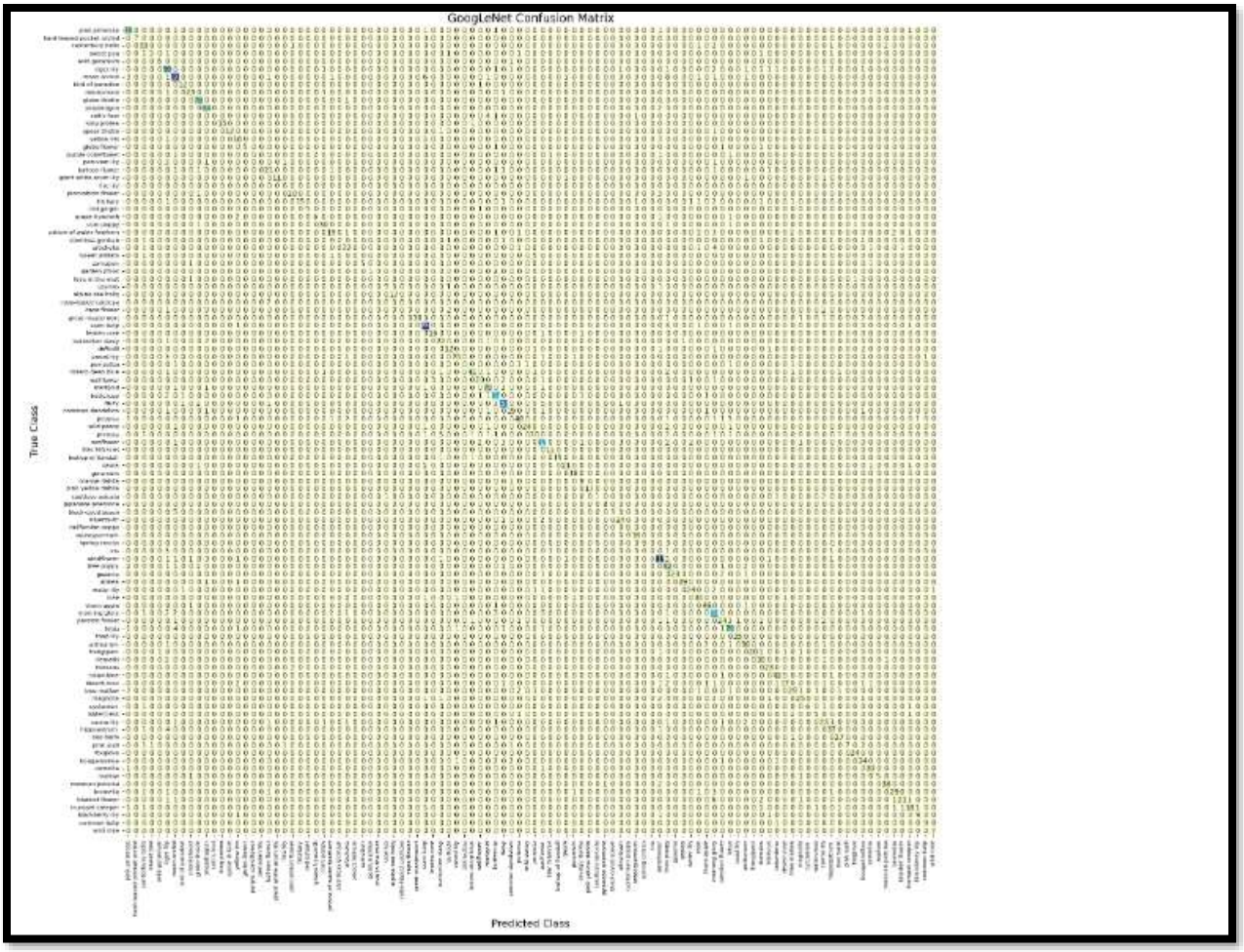
	pink primrose	hard-leaved pocket orchid	canterbury bells	sweet pea	wild geranium	tiger lily	moon orchid	bird of paradise	monkshood	globe thistle	snapdragon	colt's foot	king protea	spear thistle	yellow iris	globe-flower	purple coneflower	pe
pink primrose	55	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	
hard-leaved pocket orchid	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
canterbury bells	0	0	26	0	0	0	3	0	0	0	0	0	0	0	0	0	2	
sweet pea	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
wild geranium	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	
tiger lily	0	0	2	0	0	79	0	0	0	0	0	0	0	0	0	0	0	
moon orchid	4	0	1	0	0	0	133	0	0	3	0	0	0	0	0	0	1	

Plotting the confusion matrix

For Resnet model



Fot google net model



In summary, Project 2 focused on Flower Classification using Neural Networks. The project aimed to classify 104 types of flowers based on their images using CNNs and transfer learning techniques. The Petals to Metals dataset, consisting of diverse flower images, was used for training and validation.

The project involved downloading the dataset, visualizing random images, and displaying the number of samples per class. The training dataset was split, and data preprocessing techniques were applied, including image transformations and normalization.

A CNN model architecture was designed and trained on the training dataset. However, during evaluation on the validation set, the model showed signs of overfitting.

To address this issue, pre-trained models (ResNet and GoogLeNet) were utilized, and their performance was evaluated individually. An ensemble approach was also employed by combining the predictions of both models, resulting in improved accuracy.

Training and validation loss/accuracy plots were generated for the main model, ResNet model, and GoogLeNet model. These plots demonstrated an improvement in model performance over epochs.

The accuracy and macro F-score were calculated for each model on a test dataset, revealing their performance on unseen data. Confusion matrices were plotted to identify the most confusing classes for each model.

Overall, the project successfully achieved its objectives of exploring CNNs, transfer learning, and fine-tuning for flower classification. Further improvements can be made to address overfitting and enhance model generalization. The project serves as a foundation for future work in the field neural networks and CNN.

THANKS