Flower Classification CNN Model

Table of Contents

Introduction

This final report summarizes the insights gained in creating a robust Convolutional Neural Network (CNN) model using the Flower Recognition dataset, sourced from Kaggle [1], for image classification of various flower classes.

This dataset encapsulates 4,317 flower images sourced from Flickr, Google Images, and Yandex Images. The collection categorizes these images into five distinct flower classes: daisy, dandelion, rose, sunflower, and tulip. Specifically, the daisy class comprises 764 images, the dandelion class contains 1,052 images, the rose class includes 784 images, the sunflower class contains 733 images, and finally, the tulip class holds 984 images. Notably, the images across each flower class exhibit diverse resolutions and proportions, typically around 320x240 pixels. Leveraging this dataset, the aim is to construct a robust CNN model proficient in accurately categorizing these varied flower images into their respective classes.

To effectively accomplish this objective through a CNN model, several key steps need to be executed. Initially, there is the critical task of loading and preprocessing images belonging to the five distinct flower classes. Resizing the images to a standardized dimension and normalizing pixel values will be conducted to manage their diverse proportions and resolutions. Subsequently, the preprocessed dataset will be partitioned into three subsets: training, validation, and testing, adhering to a 70-15-15 split, as outlined in the Final Project Overview Handout. Following this, the creation of a suitable CNN model architecture tailored for image classification becomes vital. Defining convolutional layers, pooling layers, activation functions, and fully connected layers constitutes a pivotal phase in this project. Once the model architecture is established, the subsequent step involves compiling the CNN model, necessitating the specification of the optimizer, loss function, and evaluation metrics.

Training the CNN model using the validation dataset is the subsequent stride, allowing for the validation of the model's performance to prevent overfitting. Evaluation metrics such as accuracy and loss will be thoroughly examined at this stage to assess the model's effectiveness. Based on the results and assessments, the following stage involves adjusting hyperparameters, modifying the model architecture, and/or implementing various techniques to enhance the model's performance. Finally, the trained model will be evaluated using the testing dataset to predict the classes of new flower images and evaluate the accuracy of these predictions.

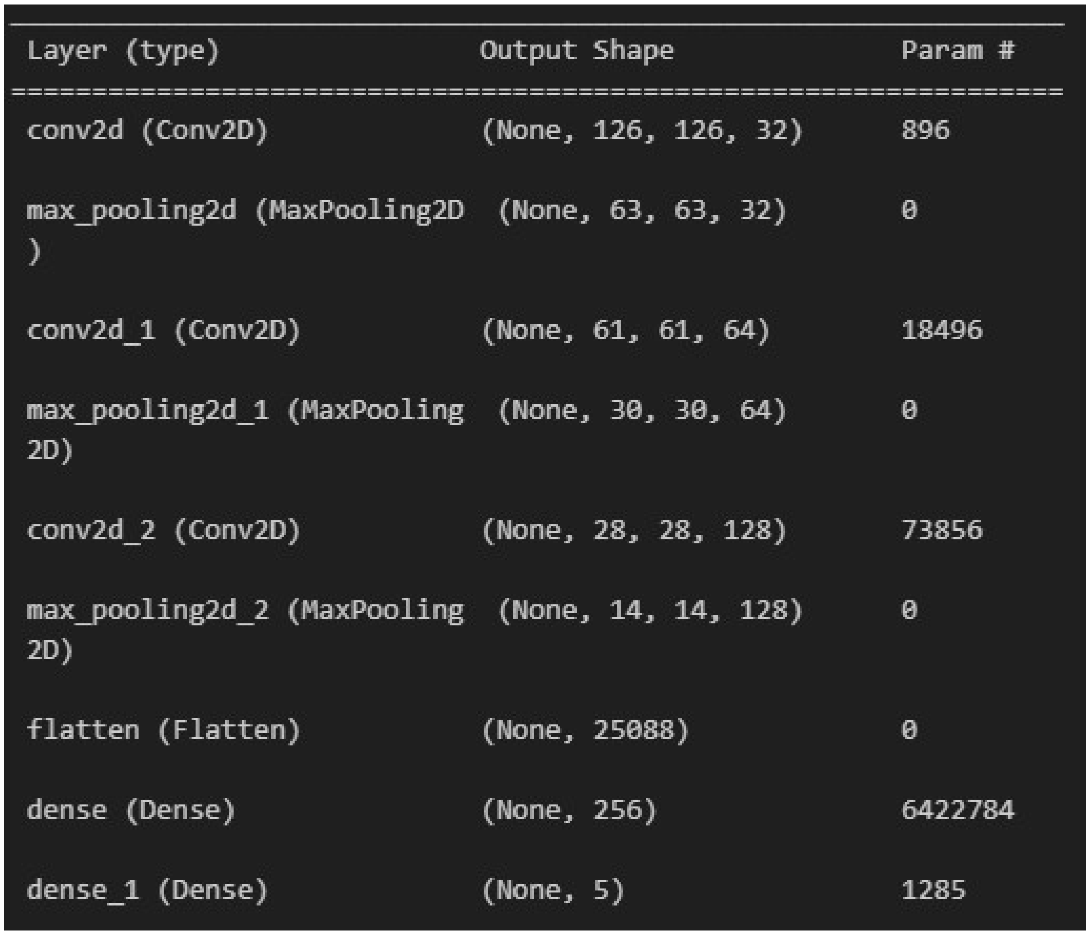
Methodology

Description of the Problem Set-Up

The problem being addressed in this project is image classification for various flower classes using a CNN model. The goal is to create a model capable of accurately identifying different types of flowers, such as daisy, dandelion, rose, sunflower, and tulip, based on the input images. 70% of the entire dataset has been utilized to train the CNN model for robust flower classification.

Network Design

For the CNN model I have implemented for my project, my network design is as follows.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 5)                 1285
```

Figure 1. Final Project Model Summary

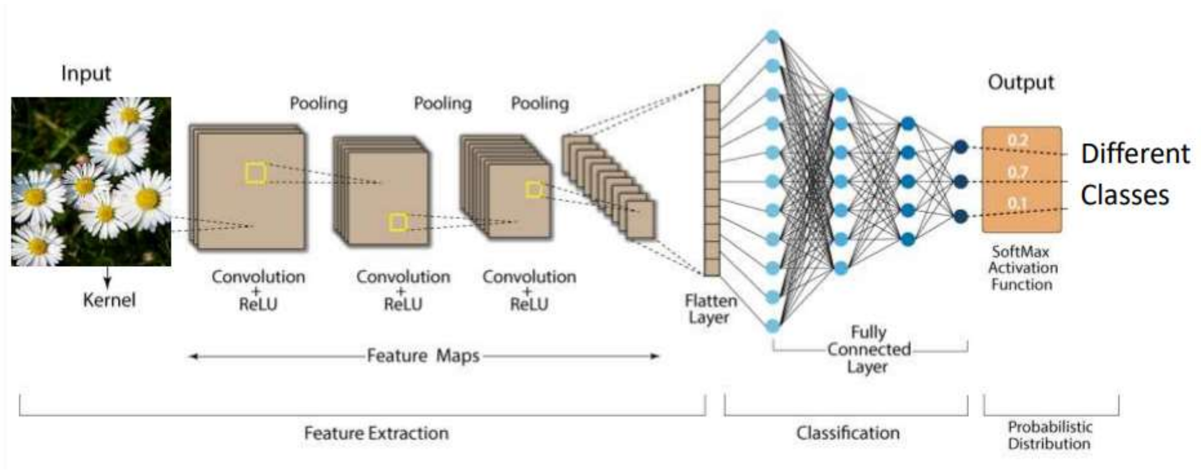


Figure 2. CNN Network Design [2]

For my project, I decided to implement a network design based on the diagram shown in the Final Project Overview Handout (shown above). One thing to note that differs my network diagram from that in the handout network diagram is that the output layer for the application of this project has five neurons, as there are five different classes, compared to the three neurons for the three different classes shown in the network diagram above (*Figure 2*).

For my network architecture, the input layer will take the shape of (128, 128, 3). I have resized the color images in the dataset to 128x128. Hence, 128 in this layer represents the height and width of the input image in pixels, and the 3 represents the number of color channels of the image: red, green, and blue (RGB).

Following the input layer, my network architecture has a Convolutional layer to extract the various features from the input image. In this Convolutional layer, the filter has been specified to 32, kernel size of 3x3 and an activation function of the rectified linear unit (ReLU).

The initial Convolutional layer [3] serves as the foundational step to extract diverse image features by performing convolution between the input image and defined filters of a specific size, in my case 32 filters was specified for my network architecture. This process involves sliding the filter, size of 3x3, over the image and computing the dot product, generating a feature map that captures critical information like edges and corners. The subsequent layer receives these feature maps, allowing CNN to progressively learn intricate image characteristics while maintaining the spatial correlation between pixels. The ReLU activation function is a widely used activation function in neural networks due to its simplicity and effectiveness in mitigating the vanishing gradient problem. It introduces non-linearity to the model by outputting the input directly if it is positive and outputting zero otherwise. Graph of the ReLU activation function is shown below (*Figure 3*).

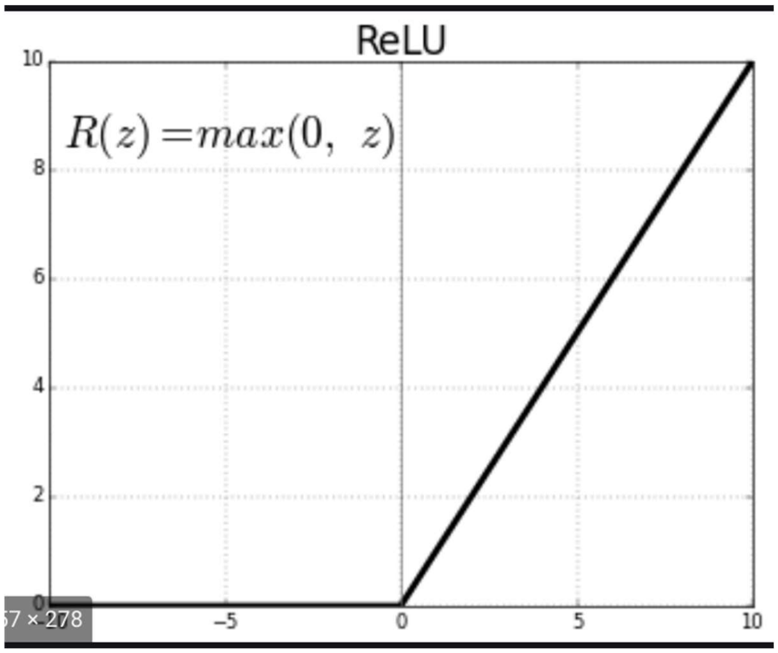


Figure 3. ReLU Activation Function [4]

Following the first Convolutional layer there is a Pooling layer [3]. In this Pooling layer, has a pool size of 2x2. The Pooling layer aids in reducing computational complexity and mitigating overfitting in CNN models. The pool size of 2x2 specifies the size of the pooling window that is applied to each feature map from the Convolutional layer. For each window, the maximum value within that window is retained, discarding the rest. This process reduces the spatial dimensions of the feature map by taking the maximum value in non-overlapping regions. The process aids in capturing the most dominant features from each section of the feature map while reducing its size, preserving important information and enhancing the network's ability to generalize to unseen data.

Next up, there is a second Convolutional layer following the first Pooling layer. In this Convolutional layer, the filter has been specified to 64, kernel size of 3x3 and an activation function of rectified linear unit (ReLU). This layer continues the process of feature extraction, initialized by the first Convolutional layer. More complex and abstract features from the image are extracted. Each filter operates as a feature detector, capturing different patterns and details from the input. The resulting output contains 64 new feature maps, each representing different learned aspects of the input image data. The ReLU activation function performs the same function as described above. The second Convolutional layer enables the network to capture higher-level features from the input data, contributing to its ability to discriminate between different classes of flowers in the dataset.

Like before, a second Pooling layer follows the second Convolutional layer. This Pooling layer has a pool size of 2x2. This layer serves the same purpose as before of down sampling the feature maps obtained from the preceding convolutional layer.

A third Convolution layer and third Pooling layer proceed with this layer respectively, serving the same purposes. Having multiple Convolutional and Pooling layers helps in learning hierarchical representations of the input data, enabling the network to extract more complex and abstract features. This step-by-step abstraction aids in creating a feature hierarchy that captures both low-level and high-level features. This feature hierarchy promotes better generalization, enabling the network to recognize similar patterns even in unseen or distorted data. The initial layers of the Convolutional layer tend to detect basic features like edges, corners, and textures. As the network progresses deeper, higher-level features are learned, combining these basic features to recognize more complex patterns or objects. Moreover, with each additional layer, the down sampling in each Pooling layer helps in controlling overfitting and decreasing the model's sensitivity to small variations in the input data [3].

Following the feature extraction stage, the Flatten layer acts as a bridge between the convolutional/pooling layers, which extract features, and the fully connected layers, which perform classification based on these features by transforming the data from a 2D spatial arrangement into a 1D sequence (as seen in *Figure 2*).

After the Flatten layer, a Dense layer [5] follows with 256 units/neurons and with the ReLU activation function. Each neuron in this layer receives input from all the neurons in the preceding Flatten layer, the more units/neurons allow the model to learn more complex patterns from the data. The ReLU activation function performs the same function as described previously, as it helps in mitigating the vanishing gradient problem and accelerates the convergence of the training process. Moreover, each neuron in the Dense layer has weights and biases, which are adjusted during training through backpropagation, allowing the model to learn the optimal values to make predictions.

Lastly, another Dense layer follows the previous Dense layer with 5 unit/neurons and with the Softmax activation function. This Dense layer is the output layer in the CNN. For classification tasks with five classes (daisy, dandelion, rose, sunflower, and tulip), this layer has an equal number of units to the number of classes being predicted. Each neuron in this layer produces an output corresponding to the probability of the input belonging to a particular class. The neuron with the highest probability indicates the predicted class. The Softmax activation function (*Figure 4*) converts the raw output values from the previous layer into probabilities for each class. The output values are normalized such that they all fall between 0 and 1 and sum up to 1, representing the probability distribution across the classes.

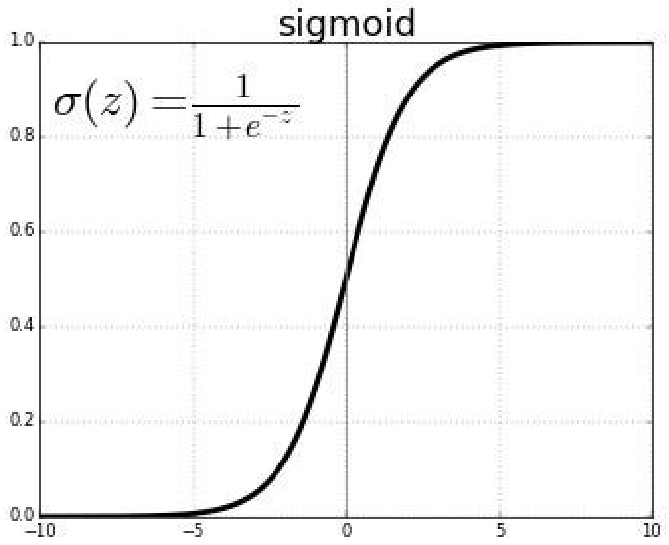


Figure 4. Softmax Activation Function [6]

Model Training and Testing

Once the CNN model is created, it requires to be trained using the Keras library's compile function. When the input image has gone through all the layers of the feature extraction process (forward pass), predictions of the class of the input image is compared to the actual class of the input image using a loss function, in this case, the categorical cross entropy loss function. This loss function is closely related to the Softmax function. It is used in classification tasks where each sample belongs to only one class (multi-class classification) [7]. This loss function helps in quantifying the difference between the predicted and actual class labels. The equation to calculate that using the categorical cross entropy loss function is shown below.

$$E(y, p) = -y \cdot \log p = -\sum_i y_i \log p_i$$

Figure 5. Categorical Cross-Entropy Equation [7]

Calculating the gradient of the loss function with respect to the weights and biases, is the start of backpropagation. The gradient will indicate the direction and magnitude of the changes needed to minimize the loss and are propagated backwards through the network. The Adam optimizer I have implemented in my CNN network will utilize these gradients and the learning rate (0.001) to update the weights and biases in the network. [8] The learning rate will help control the step size of the weight updates, to prevent overshooting or slow convergence.

Further, I have set the metrics variable in the Keras library's compile function to accuracy as that will be the measure, I will be using to evaluate the performance of my CNN model. Accuracy measures the proportion of correctly classified samples out of the total samples in the dataset. It is calculated as the ratio of the number of correctly predicted instances to the total number of instances in the dataset.

For the training process, I split the dataset into three subsets: training, validation, and testing, adhering to a 70-15-15 split. 70% of the data (training set) will be used to train the model's weights and biases by presenting the input data and their corresponding labels, 15% of the data (validation set) will be used during the training phase to fine-tune the model's hypermeters [9]. This data will be used to evaluate the model's performance. Regarding the testing scheme, the 15% of the data will be allocated for testing exclusively for evaluating the final trained model's performance. It's essential to keep the test set separate from the training and validation sets to ensure unbiased evaluation. Images outside of the data can provide additional insights into the model's performance, however, that was not done to test the model.

Moreover, I have set the batch size and epochs to 32 and 22 respectively. The entire process of forward pass, loss computation, backward pass and weight and bias updates are repeated iteratively across multiple epochs – 22 times in this case. Each epoch represents one complete cycle through the entire dataset. The batch size, 32 in my CNN model, is the number of training images utilized in one iteration. Instead of updating the model's weights after processing the entire dataset (as in full-batch training), training is performed on smaller subsets or batches of data. These batches enable faster computations and smoother convergence as the model updates its weights more frequently.

Results and Discussion

Model Performance

Below is a graph of the Training and Validation Loss over the Epochs for my CNN model. The training loss curve (in blue) represents how well the model is learning, and the validation loss curve (in orange) represents how well the model is generalizing. The relationship between the two curves can be used to diagnose the behavior of the machine learning model and help with improving the learning and/or performance of the model.

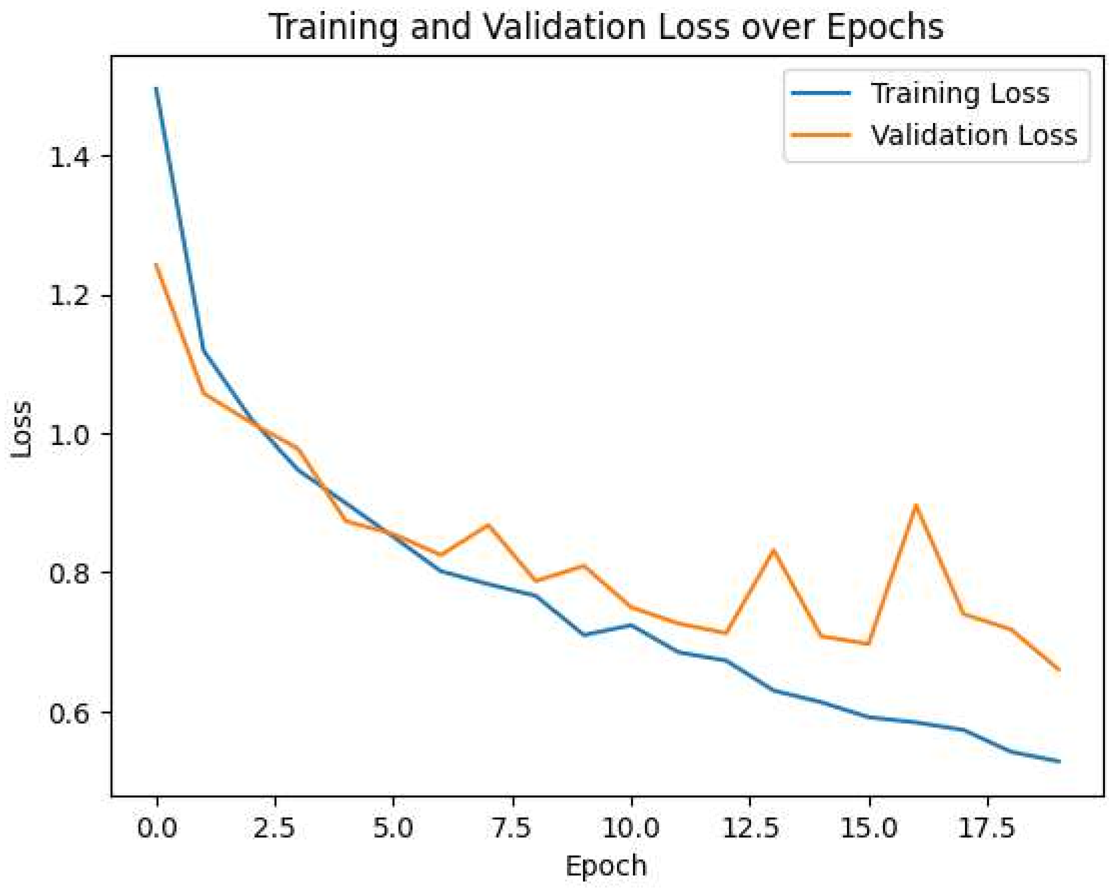


Figure 6. Training and Validation Loss over Epochs Graph

Underfit loss curves often are flat lines, have very noisy values, have a high loss and/or the training loss continuous to decrease until the end of training [10]. This means that the model was unable to learn the training dataset at all. On the opposite side of the spectrum, overfit loss curves present themselves as the training loss curve continues to decrease over the epochs and the validation loss curve decreases to a point and begins to increase again. An overfit loss indicates that the model has learned the training dataset too well [10]. A good fit loss curve is when the training loss curve is decreasing to a point of stability, and the validation loss curve decreases to a point of stability and has a gap with the training loss. This means that the model is trained well without overfitting, maintains a low and stable loss, and generalizes effectively to unseen data.

Hence, the graph above (*Figure 6*) is impacted by how the model is trained. The table below specifies the training parameters which significantly impact the model's training process and overall performance.

Model Training Parameters

Training Parameters:

Parameter	**Value**
Learning Rate	0.001
Optimizer	Adam
Loss Function	Categorical Cross Entropy
Number of Epochs	22
Batch Size	32
Metrics	Accuracy

Figure 7. Table of Training Parameters

Furthermore, my CNN model's model summary is shown in *Figure 1*. As seen, my model architecture consists of 3 Convolutional layers each followed by a Pooling layer. These six layers make up the Extraction stage of the architecture. Following the last Pooling layer there is a Flatten layer, followed by two Dense layers – making up the Classification stage. Parameters set during training and their application were described above, under the Methodology section, subsection Network Design of this report.

Classification Results

The train loss and accuracy from evaluating the model was 0.2657 and 91.0626%, respectively. *Figure 6* shows the loss of the training dataset, and *Figure 8* shows the accuracy of the training dataset. The trends that the loss of the training data is decreasing, and that the accuracy of the training data is increasing indicate that the model is learning the patterns in the training dataset effectively. It's important that the model doesn't overfit by observing if the training accuracy and loss continue to improve without significantly diverging from the validation dataset curves in both *Figure 6 and 8*.

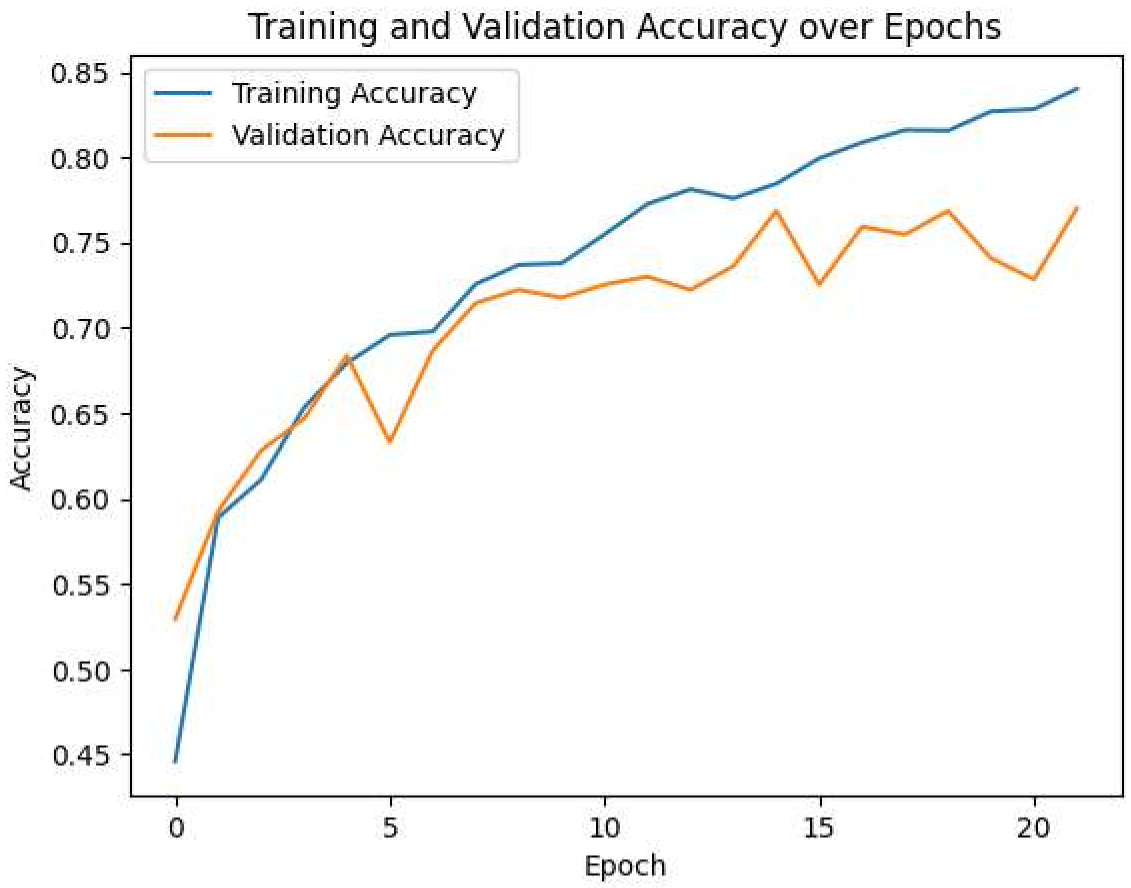


Figure 8. Training and Validation Accuracy over Epochs Graph

The validation loss and accuracy from evaluating the model was 0.7326 and 78.3951%, respectively. A similar trend as the training loss and accuracy is seen in *Figure 6 and 8*. The consistency between training and validation metrics, indicating good generalization. A smaller gap between training and validation loss indicates better generalization, showing that the model isn't overfitting.

From both the training and validation losses, small losses are obtained signifying that the model is minimizing errors. From both the training and validation accuracies, we see that a high accuracy was obtained signifying that the model learned useful patterns to classify the flower images correctly.

In terms of testing, a confusion matrix is a great way to visualize and assess the classification results of the CNN model. It shows how many of the predictions made by the model were correct and how many were incorrect for each of the flower classes. *Figure 9* is a confusion matrix for the model predicting the classes of the test images. 499 test images were classified correctly, and 149 test images were classified incorrectly.

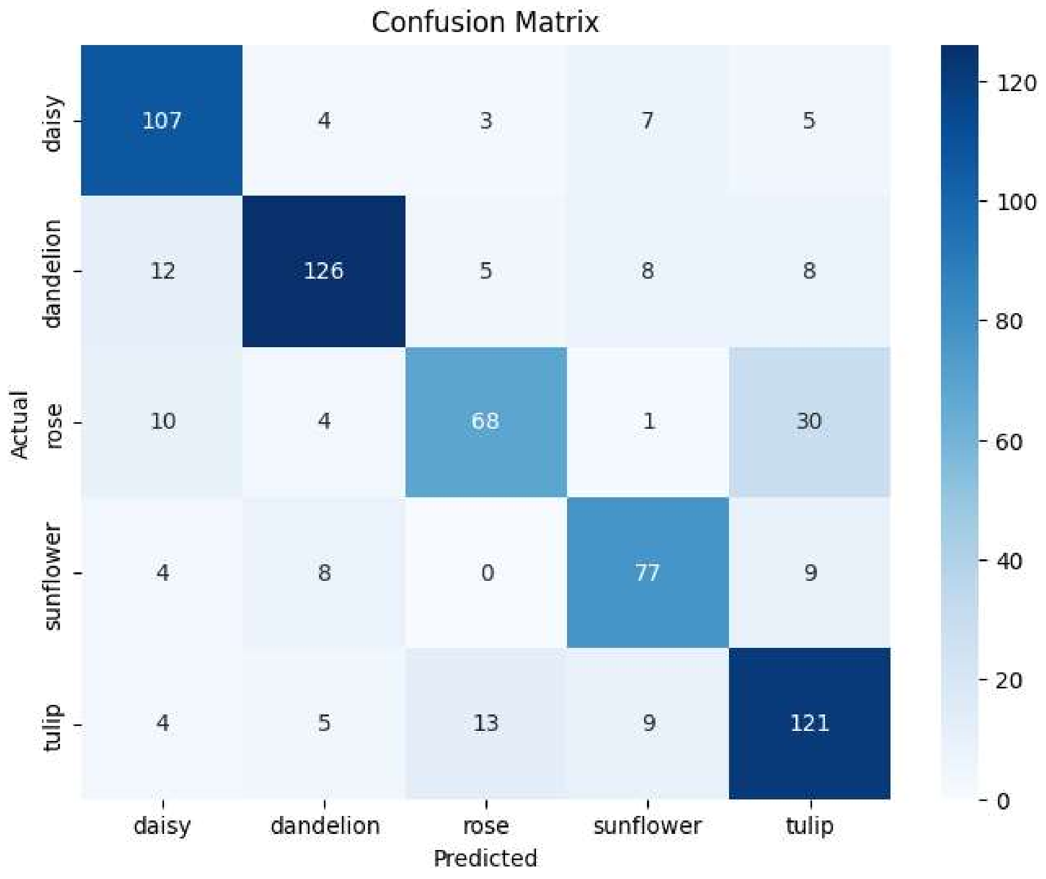
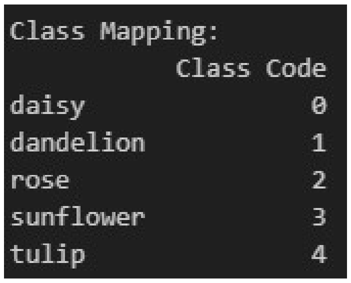


Figure 9. Confusion Matrix

Using observations for the performance of the CNN model via *Figure 6 and 8* and the confusion matrix (*Figure 9*) to understand how the classes of flowers were predicted using the model, the resulting test loss and accuracy of the model were 0.6463 and 77.0062%, respectively.

Figure 11 showcases the model classifying some of the test flower images. *Figure 10* will aid in encoding the flower classes predicted in *Figure 11*. The actual test image is shown on the left side, on top of which the actual test class label is shown. The right side shows the probability distribution of the flower classes, outputted from the CNN model for each of the flower classes, from which the highest probability is taken to predict the class of the image (shown on top of the probability distribution).



```
Class Mapping:
           Class Code
daisy               0
dandelion           1
rose                2
sunflower           3
tulip               4
```

Figure 10. Class Mapping of the Different Flower Classes

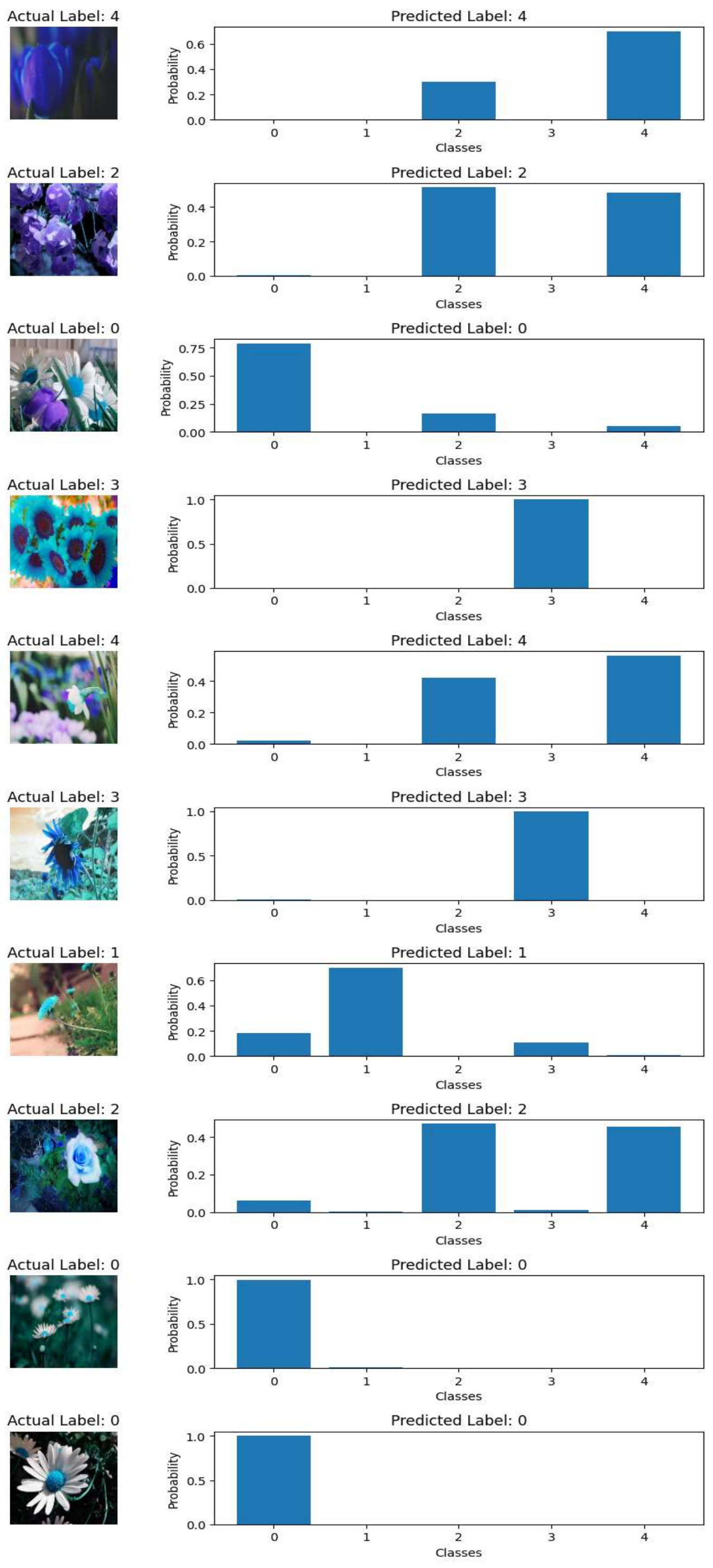


Figure 11. Displaying the Images with their Actual and Predicted Classes

The model achieved relatively moderate accuracy and a loss of 0.6463 on the test set. An accuracy of 77.0062% indicates that the model correctly predicted the class for roughly three-quarters of the test images. The model could be limited by the size or diversity of the dataset. A larger and more diverse dataset might improve the model's ability to generalize, and/or fine-tuning the model's architecture can also aid. Furthermore, the task of distinguishing between different flower classes can be challenging, especially if there are subtle differences between certain classes or images contain noise or variations.

Prediction on Images Outside of the Dataset

Predicting the generalizability of the model to images outside the dataset is challenging without additional information about the external images. The model may have a hard time distinguishing the image between daisy, rose and sunflower flower classes, as there were less images for these classes for the training of the model. This is reflected in the distribution of the images in the entire dataset in *Figure 12*.

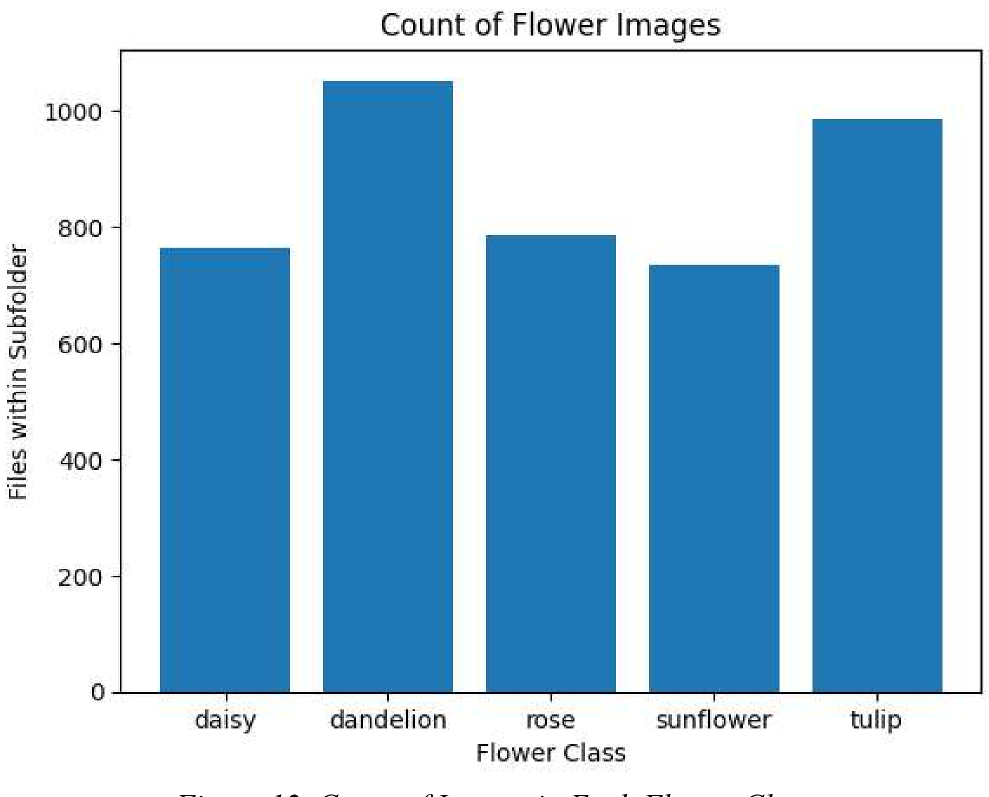


Figure 12. Count of Images in Each Flower Class

However, given that the dataset used for training was diverse and representative of real-world scenarios, and the model exhibits robust performance on the test set (an accuracy of 77.0062% on the test set), there's a reasonable chance that it might generalize well to unseen images. Yet, this is not a guarantee, and actual performance on entirely new data needs to be evaluated separately.

Conclusion

All in all, the final project for ENEL 525 – Machine Learning for Engineers (Fall 2023) required classifying flower images into five categories: daisy, dandelion, rose, sunflower, and tulip. To achieve this task a CNN model was created, with three sets of a Convolutional layer and Pooling layer – creating the Feature Extraction stage of the CNN model, followed by a Flatten layer, and 2 Dense layers – creating the Classification stage of the CNN model.

The CNN model created was trained using 70% of the Flower Recognition dataset from Kaggle. [1] The test evaluation of the model resulted in a test loss of 0.6463 and an accuracy of 77.0062%.

The CNN model was designed to extract relevant features from the images and classify them into their respective classes. It utilized convolutional layers to detect patterns, followed by pooling layers to down sample the feature maps and reduce computational complexity. Fully connected layers were used for classification.

While the model demonstrated moderate performance with an accuracy of 77.0062%, there is room for improvement. Additional strategies, such as collecting more diverse data and/or fine-tuning the model architecture could potentially enhance its performance.

In conclusion, the CNN model showcased reasonable performance in classifying flower images but could benefit from further refinements to achieve better accuracy and robustness. The results serve as a starting point for potential enhancements and further exploration in improving the model's classification capabilities.

References

[1] A. Mamaev, "Flowers Recognition," Kaggle, 16 July 2021. [Online]. Available: https://www.kaggle.com/datasets/alxmamaev/flowers-recognition/data. [Accessed 8 December 2023].

[2] E. a. S. E. Department, "Final Project Overview," 6 December 2023. [Online]. Available: https://d2l.ucalgary.ca/d2l/le/content/543044/viewContent/6190930/View. [Accessed 8 December 2023].

[3] "Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network," upGrad blog, [Online]. Available: https://www.upgrad.com/blog/basic-cnn-architecture/. [Accessed 8 December 2023].

[4] P. Chima, "Activation functions: Relu & Softmax," Medium, 6 April 2020. [Online]. Available: https://medium.com/@preshchima/activation-functions-relu-softmax-87145bf39288. [Accessed 8 December 2023].

[5] "The Concepts of Dense and Sparse in the Context of Neural Networks," Baeldung on Computer Science, 24 May 2023. [Online]. Available: https://www.baeldung.com/cs/neural-networks-dense-sparse. [Accessed 8 December 2023].

[6] A. Suman, "Activation Function," Medium, 29 August 2020. [Online]. Available: https://medium.com/analytics-vidhya/activation-function-c762b22fd4da. [Accessed 8 December 2023].

[7] L. David, "Activation, Cross-Entropy and Logits," 30 August 2021. [Online]. Available: https://lucasdavid.github.io/blog/machine-learning/crossentropy-and-logits/. [Accessed 8 December 2023].

[8] "How does Backpropagation in a Neural Network Work?," [Online]. Available: https://builtin.com/machine-learning/backpropagation-neural-network. [Accessed 8 December 2023].

[9] T. Shah, "About Train, Validation and Test Sets in Machine Learning," Towards Data Science, 10 July 2020. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7. [Accessed 8 December 2023].

[10] J. Brownlee, "How to Use Learning Curves to Diagnose Machine Learning Model Performance," Machine Learning Mastery, 6 August 2019. [Online]. Available: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/. [Accessed 8 December 2023].