Flower Image Classification Using CNN

Cuong Duc Nguyen
University of Skövde
Skövde, Sweden
Cuongnguyense@yahoo.com

Georgios Kokolakis
University of Skövde
Skövde, Sweden
giorgoskokolakis.97@gmail.com

***Abstract*—In this paper, we present a system for classifying images of flowers using deep convolutional neural networks (CNNs). The system is trained and tested on a dataset of images of flowers, where each image is labeled with the species of flower it depicts. We evaluate the performance of the system using metrics such as accuracy, precision, and recall. Our system achieves high accuracy in classifying images of flowers, demonstrating the effectiveness of CNNs in this task. Overall, the proposed system is a valuable tool for automating the identification of flower species, which can have applications in fields such as conservation, agriculture, and research. Additionally, the findings in this paper could be extend to other image classification tasks such as plant recognition in general.**

Keywords—CNN, Image Classification, Layers, Convolutional, Pooling, Activation Function, Evaluation Metrics, Confusion Matrix.

I. INTRODUCTION

Flower image classification is a vital task in fields such as botany, horticulture and ecology. Flowers are used as a model system in various scientific studies, thus accurate identification of flower species is crucial in data collection and analysis. In recent years, the rise of deep learning has led to a significant improvement in image classification performance. Among the deep learning architectures, Convolutional Neural Networks (CNNs) have been widely used to achieve state-of-the-art results in many image classification tasks.

In this work, we propose a CNN-based model for classifying flower images. The proposed model uses transfer learning, which is a powerful technique to leverage the knowledge learned by a pre-trained model, fine-tuning it on a new dataset. We have collected a dataset of flower images and labeled them according to their species. The model is trained and tested on this dataset, and the performance is measured by various evaluation metrics such as accuracy, precision and recall. Our goal is to achieve high classification performance, to further its practical applications in various fields such as conservation, botanical research and commercial agriculture.

II. THE DATASET

The dataset used comprises of 4242 images of flowers that were collected from different sources such as flickr, google images and yandex images. The images are labeled into five different classes, namely chamomile, tulip, rose, sunflower, and dandelion, with around 800 images per class. These images can be used to train models for recognizing different plant species from photographs. However, it is important to note that these images are not of high resolution, being approximately 320x240 pixels and also have different proportion [1].

III. THE PROCESS

Input: The first step is to provide the CNN with the input data, such as an image or a video. The input data is typically preprocessed and normalized to ensure that the network can effectively learn from it.

Convolution: The next step is the convolution operation, where a set of filters (also called kernels or weights) are applied to the input data. These filters are used to extract features from the input data, such as edges, textures, and patterns. The convolution operation produces feature maps that represent different aspects of the input data.

Non-Linearity (Activation): Next step, the feature maps are passed through an activation function, which introduces non-linearity into the model. Commonly used activation functions are ReLU, sigmoid and Tanh.

Pooling: After the convolution and activation step, a pooling operation is applied to the feature maps. The purpose of pooling is to down-sample the feature maps and reduce their dimensions, which helps to control overfitting and reduce the number of parameters in the network.

Repeating steps 2-4: These three steps are repeated multiple times in the network, with each repetition resulting in a deeper and more abstract representation of the input data.

Fully Connected Layers: The output of the final convolutional/pooling layer is passed through one or more fully connected layers. These layers are used to make final predictions or decisions based on the learned features.

Output: Finally, the output of the CNN is generated, typically in the form of class scores or probabilities.

Backpropagation and Optimization: Based on the output the network makes a prediction, the prediction is then compared to the actual output (label) and the error is calculated using a loss function, optimizer then use backpropagation to optimize the model parameters, making the network to reduce the error. [2]

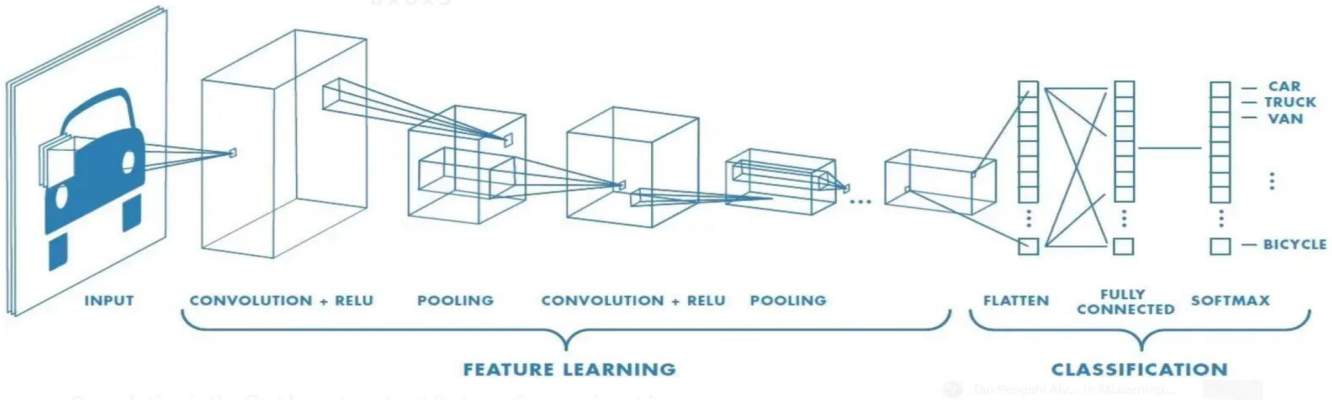

Figure 1 Complete Flow of CNN to process an input image and classifies the objects based on values .

IV. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) have been widely used for image classification tasks because of their ability to automatically learn features from images. The key advantage of CNNs is their ability to learn hierarchies of features, with lower-level features such as edges and textures being learned in the early layers and higher-level features such as shapes and parts of objects being learned in the deeper layers. This hierarchical structure allows CNNs to automatically learn relevant features for image classification, which can be difficult or time-consuming to design manually.

Another advantage of CNNs is that they can effectively process images of different scales, orientations, and translations, making them robust to image variations. This is achieved by the use of convolutional layers, which apply filters to small regions of the input image and are able to maintain the spatial relationships between pixels. This allows CNNs to learn spatial hierarchies of features, enabling them to detect patterns and features regardless of their position in the image.

Additionally, CNNs use pooling layers to reduce the dimensionality of the data while preserving the most important information. This reduces the computational cost of the model and improves its generalization capabilities.

Finally, CNNs are able to take advantage of the large amount of labeled data and computational resources available today, which allows them to be trained on large and complex datasets. This is particularly useful for image classification tasks, where large amounts of labeled data is required to train accurate models.

Overall, CNNs are well-suited for image classification tasks because they can automatically learn relevant features from images, handle image variations and preserve the most important information, and are able to take advantage of large amounts of labeled data and computational resources [3].

We will use the Sequential model to add the necessary layers to our CNN. The Sequential model is a way to define the architecture of the neural network by creating a linear stack of layers. The Sequential model is a simple and easy-to-use model provided by the Keras library, which allows you to define the architecture of your network by adding layers one by one.

A. Conv2D

A CNN is typically composed of multiple layers, including one or more convolutional layers (Conv2D) as well as other types of layers such as pooling layers, fully connected layers and normalization layers.

Conv2D layers are a specific type of layer that are used in CNNs to learn spatial hierarchies of features from images. These layers work by applying a set of filters to small regions of the input image, creating a set of feature maps. Each filter is designed to detect a specific feature or pattern in the image, such as edges or textures. By applying multiple filters, the convolutional layer can learn a variety of different features from the image.

The role of Conv2D layers in a CNN is to learn the local patterns of the images and extract the features from the images. These features are then passed to the next layer, which can be either another convolutional layer or a pooling layer. The pooling layer reduces the dimensionality of the data, while preserving the most important information. This process is repeated through multiple layers and the features learned by each layer are combined to form a more robust and abstract feature representation of the image.

Finally, the output of the last convolutional/pooling layer is passed to a fully connected layer (also known as Dense layer), which uses the features learned by the previous layers to make a final prediction about the image.

The main advantage of Conv2D layers is that they are able to learn local patterns in the image, preserving the spatial information of the input. For example, Conv2D layers are able to detect edges and textures regardless of their position in the image, which allows the network to be robust to translations and rotations. Conv2D layers also share their parameters across different regions of the input image, reducing the number of parameters required to learn and also reducing overfitting.

Conv2D layers have also the ability to increase the depth of the representation in CNNs, which means creating more feature maps, and hence, adding more layers to learn more abstract and complex features. This process called stacking, is essential for the CNNs to build a robust feature representations to classify images.

Furthermore, Conv2D layers allow the network to learn features that are sensitive to the spatial relationships between

pixels, which is crucial for tasks such as image classification. By using Conv2D layers, the network can learn features such as shapes and parts of objects, which are crucial for classifying images correctly [4].

B. Activation Functions

In a Conv2D layer, the activation function is used to introduce non-linearity into the network, allowing it to learn complex and abstract features from the input. The activation function is applied element-wise to the output of the convolution operation, creating a new output called the activation map.

The most commonly used activation functions in Conv2D layers are ReLU (Rectified Linear Unit) and its variants such as LeakyReLU, PReLU, and ELU (Exponential Linear Unit). ReLU is a simple and computationally efficient activation function that replaces all negative values with zero, which speeds up the training process and can also help reduce overfitting.

LeakyReLU is a variant of ReLU that allows small negative values to pass through, which can help prevent the dying ReLU problem (when all the neurons output zero).
PReLU is another variant of ReLU that learns the value of the negative slope of the function.
ELU is similar to ReLU but instead of replacing negative values with zero, it replaces it with an exponential function, which allows for the network to learn smoother and more robust features.

Other activation functions like Sigmoid, tanh and Softmax can also be used but they are less commonly used as they may lead to slow down the training process and overfitting.

Overall, the choice of activation function depends on the dataset, problem, and the specific requirements of the model. In practice, ReLU and its variants are often used in Conv2D layers due to their computational efficiency, stability, and effectiveness in preventing overfitting [5].

C. Max Pooling

In a CNN, max pooling is a technique used to down-sample the spatial dimensions of the feature maps, which can reduce the computational cost of the model and improve its generalization capabilities. The max pooling operation is typically applied after one or more convolutional layers and works by dividing the feature map into a set of non-overlapping regions, or pooling windows, and then taking the maximum value of each region.

The max pooling operation is implemented in Keras using the MaxPooling2D layer. The layer is typically added after one or more convolutional layers, and it takes several arguments such as pool_size and strides to control the size and stride of the pooling window.

The pool_size parameter controls the size of the pooling window, which is typically set to (2, 2) to reduce the spatial dimensions by a factor of 2 in each direction.

The strides parameter controls the step size of the pooling window, which is typically set to the same value as the pool_size to ensure that the pooling window does not overlap with itself [4].

D. Flatten & Dense Layers

We finally add the Flatten and Dense layers to complete our CNN. In a CNN, the Flatten and Dense layers are used to prepare the feature maps learned by the previous layers for final classification.

The Flatten layer is used to convert the feature maps from a 2D array to a 1D array. It does this by unrolling the 2D array into a long 1D array, which can be used as input to the final fully connected layers of the network. The flatten layer is usually added after one or more convolutional and max pooling layers.

The Dense layer is a fully connected layer that is used to make the final prediction. It is typically added after the Flatten layer and takes the 1D array of features learned by the previous layers as input. The Dense layer applies a linear transformation to the input, which is then followed by a non-linear activation function. The Dense layer has a number of units (also known as neurons) that indicate the number of output classes.

V. Data Augmentation

Data augmentation is a technique used to artificially increase the size of a dataset by applying various transformations to the images, such as rotation, translation, flipping, and scaling. The goal of data augmentation is to increase the diversity of the training data, which can help improve the robustness and generalization of the CNN model [6].

When working with image data, it is common to apply data augmentation techniques such as:

- Rotation: images can be rotated by a random angle to make the model more robust to rotations in the input.
- Translation: images can be translated by a random amount to make the model more robust to translations in the input.
- Flipping: images can be horizontally or vertically flipped to make the model more robust to symmetry in the input.
- Scaling: images can be scaled by a random factor to make the model more robust to scale variations in the input.
- Zooming: images can be zoomed in or out to make the model more robust to zoom variations in the input.
- Lighting: brightness and contrast can be adjusted to make the model more robust to lighting variations in the input.

VI. The Final Model

We compile the model by setting the optimizer, loss function and evaluation metrics. 'Adam' optimizer is used, it's default learning rate is 0.001. The loss function used is categorical_crossentropy which is commonly used for multi-

class classification problems. The evaluation metric is 'accuracy' which is the ratio of correctly classified samples to total number of samples.
The data was split in training and test sets and a 20% test size was used. We then used datagen which is a generator that yields batches of data for training. It takes the training data X_train and labels y_train, and a batch size of 128. This generates data for the training process in a batch of 128 at a time. The number of epochs was set to 50. The fit() method trains the model on the generator and also performs validation on X_test and y_test data set at the end of each epoch, with a batch size of 128 and runs for a total of 50 epochs.

VII. Evaluation

For evaluation metrics we utilized the confusion matrix. The confusion matrix provides a useful way to summarize a large amount of data and can be used to calculate various evaluation metrics such as accuracy, precision, recall and F1-Score, which are useful to analyze the performance of the model [7].

Here is a general explanation of the terms that are used to describe the matrix:

- True Positives (TP): These are the cases in which we predicted yes (they have the condition), and they do have the condition.
- True Negatives (TN): We predicted no, and they don't have the condition.
- False Positives (FP): We predicted yes, but they don't actually have the condition. (Type I error)
- False Negatives (FN): We predicted no, but they actually do have the condition. (Type II error)

Accuracy, precision, recall and F1-score are evaluation metrics that are commonly used to measure the performance of a classification model. They can be calculated using the values in a confusion matrix.

- Accuracy is the ratio of correctly predicted observation to the total observations.
- Accuracy = (TP + TN) / (TP + TN + FP + FN)
- Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.
- Precision = TP / (TP + FP)
- Recall (Sensitivity or True Positive Rate) is the ratio of correctly predicted positive observations to the total actual positive observations.
- Recall = TP / (TP + FN)
- F1-Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances). The greater the F1 Score, the better is the performance of our model.
- F1-Score = 2*(Recall * Precision) / (Recall + Precision)

Our model achieved an accuracy score of 77% on the test set. The best precision score was identified on Daisy flowers with a score of 0.91. The flower with the highest recall as well as the highest F1-score was the Sunflower which was 0.91 and 0.89 respectively. Overall, the model seems to correctly identify Sunflowers and Daisies better than the rest.

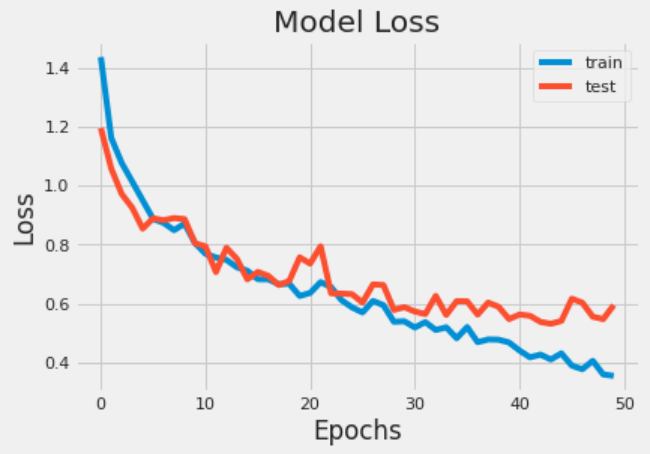

Figure 2. Model loss over the Epochs on training and testing sets

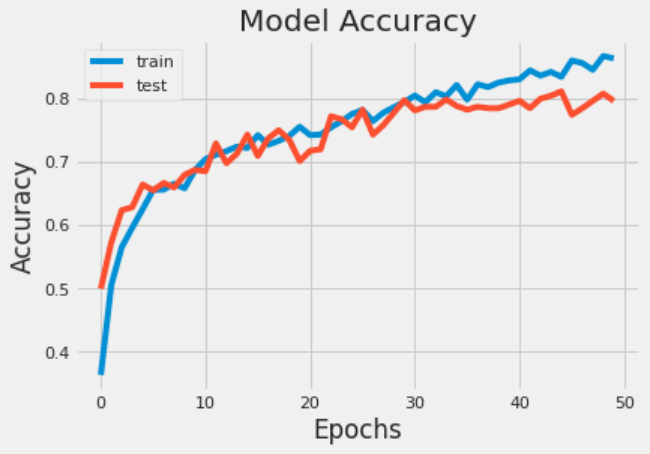

Figure 3. The Models' Accuracy over the Epochs on the training and testing sets

Future Work

Although our model performs adequately, there are several steps we could follow, given more time, that would improve our models' performance.

Namely, we could collect more data. Having a large and diverse dataset is crucial for training accurate image classification models. If your dataset is small or not diverse enough, collecting more data can help improve the model's performance.

We could also choose a more powerful model architecture. There are several pre-trained models available like ResNet, VGG, Inception etc. Using a more complex and powerful model architecture can improve the performance of the model.

Also, tuning our hyperparameters might increase our accuracy. Tuning the hyperparameters of the model, such as the learning rate, batch size, and number of epochs can improve performance. Grid search or random search can be used to find the optimal hyperparameters.

Using Ensemble methods by combining multiple models can also improve performance. For example, by training multiple models and then taking a majority vote, we can improve the overall accuracy.

Furthermore, leveraging the power of Cloud and GPUs, because the computational power of cloud and the utilization of Graphics Processing Units (GPUs) can greatly speed up the training process, and therefore improve the performance of the model.

It's worth noting that these methods are not mutually exclusive, and often, a combination of these methods is used to achieve the best results. It's also important to keep in mind that accuracy is not the only metric to consider when evaluating the performance of a model and that the choice of method(s) will depend on the specific problem and the available resources.

In the future, we could also try to classify different objects besides flowers with our model to evaluate its performance. Specifically, among other things the model can be used to classify:

- Animals: Dogs, cats, horses, birds, etc.
- Vehicles: Cars, trucks, buses, motorcycles, airplanes, etc.
- Plants: Trees, grass, etc.
- Fruits and vegetables: Apples, bananas, tomatoes, etc.
- Household items: Chairs, tables, lamps, etc.
- Fashion and apparel: Clothing, shoes, jewelry, etc.
- Food: Pizza, sushi, sandwiches, etc.
- Body parts: faces, eyes, hands, etc.
- Scenes: Beach, city, mountain, etc.
- Objects and tools: Scissors, hammer, computer, etc.

REFERENCES

[1] "Flowers Recognition." https://www.kaggle.com/datasets/alxmamaev/flowers-recognition (accessed Dec. 23, 2022).

[2] Prabhu, "Understanding of Convolutional Neural Network (CNN) — Deep Learning," *Medium*, Nov. 21, 2019. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 (accessed Jan. 11, 2023).

[3] T. Guo, J. Dong, H. Li, and Y. Gao, "Simple convolutional neural network on image classification," in *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, Mar. 2017, pp. 721–724. doi: 10.1109/ICBDA.2017.8078730.

[4] M. Agarwal, S. Gupta, and K. K. Biswas, "A new Conv2D model with modified ReLU activation function for identification of disease type and severity in cucumber plant," *Sustain. Comput. Inform. Syst.*, vol. 30, p. 100473, Jun. 2021, doi: 10.1016/j.suscom.2020.100473.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 2015, pp. 1026–1034. doi: 10.1109/ICCV.2015.123.

[6] S. Y. Feng *et al.*, "A Survey of Data Augmentation Approaches for NLP." arXiv, Dec. 01, 2021. Accessed: Jan. 11, 2023. [Online]. Available: http://arxiv.org/abs/2105.03075

[7] M. Heydarian, T. E. Doyle, and R. Samavi, "MLCM: Multi-Label Confusion Matrix," *IEEE Access*, vol. 10, pp. 19083–19095, 2022, doi: 10.1109/ACCESS.2022.3151048.