Flower Classification

A. Abstract

In recent years, deep learning techniques have garnered significant attention for multimedia applications. This study aims to develop and compare the performance of three Convolutional Neural Network (CNN) models—ResNet18, MobileNetV2, and Densenet121—for flower image classification. Each model will be trained and tested on three distinct datasets, resulting in a total of nine models. Additionally, two models will be developed using transfer learning. To improve the accuracy and robustness of the models, we apply preprocessing techniques such as image resizing, center cropping, and normalization. The models' performance will be evaluated using metrics including accuracy, precision, recall, F1 score, and the confusion matrix. Our results demonstrate that these models achieve high accuracy, with the exception of Densenet, and are effective in classifying flower images, offering potential for broader applications in the field of artificial intelligence. Furthermore, we have implemented t-SNE visualization for all the models.

B. Introduction and Problem Statement

Flowers play a crucial role in ecosystems by providing food and medicinal resources. The Flower Classification project seeks to leverage image recognition technology to classify various flower species, aiding in conservation, cultivation, and pharmaceutical applications. Deep learning techniques are favored for flower identification over manual classification, as they efficiently handle large datasets and accurately categorize images, identifying subtle patterns and features for improved precision. However, training an algorithm on a dataset containing only one type of flower may reduce its accuracy when classifying multiple species. Deep learning models continually improve through training and optimization, becoming adaptable and efficient for real-time classification with minimal human intervention.

Associated Challenges: In a multi-label, multi-class flower classification project, several challenges must be addressed to ensure accurate predictions. Large batch sizes can lead to biased training, negatively impacting model accuracy. Balancing the number of samples per class within each batch is critical to overcome this issue. For instance, Keqi Zhou's paper applied stratified segmentation to handle class imbalances effectively. Another challenge is that models may struggle to generalize when exposed to flower images taken in different environments or lighting conditions, or when differentiating between visually similar flowers. This issue arises when the model is trained on a specific set of images that fail to represent the full range of real-world variations. Data augmentation techniques, such as random cropping, flipping, and rotation, have been used to mitigate these challenges and improve model accuracy. When encountering unfamiliar images, the model may struggle to make accurate classifications. Optimizer choice also plays a significant role in model performance, with its effectiveness depending on dataset size and complexity. In Vani's research, various optimizers were assessed to improve accuracy using a customized learning rate. Transfer learning, which uses pre-trained models as a starting point, may not always be optimal for new datasets with more classes, requiring fine-tuning of parameters and modifications to the architecture to achieve better accuracy. Tan, Chuanqi, and Sun highlighted the importance of transfer learning in solving data insufficiency issues. Furthermore, Xue Ying's work on regularization demonstrated how it can help prevent overfitting and improve generalization when using transfer learning on datasets with a large number of classes.

Pros and Cons of Existing Solutions: Larger batch sizes speed up training but may cause suboptimal convergence or oscillation around the loss function's minimum. While image diversity is essential for model generalization, training on too many similar images may lead to overfitting. Optimizers can significantly enhance performance by reducing convergence time and improving accuracy, though selecting the right one often requires extensive experimentation. Pre-trained models may not be ideal for new datasets, requiring extensive fine-tuning or modification. Balancing datasets and preventing overfitting can be particularly challenging with multi-class datasets.

Proposed Solutions and Further Challenges: To address the issues of imbalanced datasets, techniques like stratified or weighted sampling can be employed to ensure equal class representation during training. Shifting from flower classification to data recognition can help manage multi-class images more effectively. Increasing dataset diversity by including images taken in various settings will improve the model's ability to generalize to real-world scenarios. Optimizer selection should be based on dataset size and model complexity, with performance comparisons aiding the final decision. Fine-tuning pre-trained models and modifying their architectures will be crucial for achieving optimal accuracy on new datasets. To improve classification across diverse flower types in real-world scenarios, the model must account for variations in lighting, background, and other factors. Handling noisy or incomplete data and dealing with class imbalances are additional challenges that need addressing to improve model accuracy. Efficient management of computational resources is also necessary for training and testing large models or datasets.

Work Done: In this project, we aimed to classify flower

images using datasets with varying class numbers, channel counts, and image sizes. We tested three widely used models—ResNet18, MobileNetV2, and Densenet121—by tuning their hyperparameters and optimizing them to showcase the output of each model. Each of the three models will be tested on three different datasets, generating a total of nine models. Additionally, two trained models will serve as base models for transfer learning on other datasets. All models were trained using different train-test splits, with the Adam optimizer and cross-entropy loss function applied. Most models were trained for at least 20 epochs, with a default learning rate of 0.001. We also developed a custom dataset class to map and transform images according to project requirements. The results are presented through accuracy and loss graphs, with t-SNE used for better visualization of data splits.

B.1. Related Work

The paper [1] introduces an efficient automatic flower classification system utilizing deep feature extraction and feature selection techniques. It employs two popular pretrained CNN models, AlexNet and VGG16, for feature extraction, with the mRMR feature selection method used to identify the most relevant features. The proposed approach demonstrated superior accuracy compared to previous works across various experiments conducted on the Flower17 and Flower102 datasets. The deep features selected in this method achieved the highest evaluation scores, with AlexNet outperforming VGG16 in terms of results. However, performance degradation was observed when the dataset was split into training, validation, and test sets.

The study in [2] presents a deep learning-based method for flower segmentation, detection, and classification using CNNs. This method follows a two-step approach, which simplifies the classification task and enhances accuracy. It achieves the best classification accuracy for flower classification to date across three different datasets. The study also introduces innovative techniques, including the use of transferred weights and data augmentation methods, which significantly improve the robustness and accuracy of the CNN. This work contributes to the advancement of deep learning-based methods for flower classification and offers potential for other image classification applications.

Another study [4] focuses on flower image classification using deep CNNs and data augmentation techniques. This approach outperformed others in the literature when tested on the Oxford-17 Flowers and Oxford-102 Flowers datasets. The proposed system is particularly useful in areas with limited labeled data, and the study demonstrated the use of t-SNE for visualizing images and the distribution of extracted features. Additionally, the study compared the performance of different classifiers and found that while SVM showed less performance degradation with larger datasets, it was slower than the Multilayer Perceptron classifier.

C. Proposed Methodology

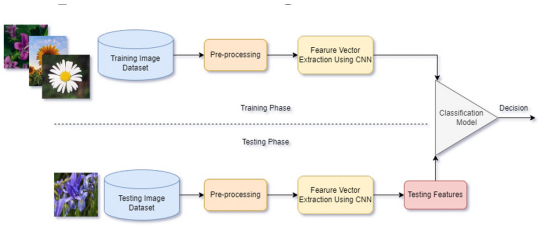

Figure 1. Your image caption here

C.1. Dataset

Dataset I: features flower images from Utkarsh Saxena's "Flower Classification dataset" on Kaggle, organized into five categories: Rose, Daisy, Sunflower, Lavender, and Lily. The images exhibit varying resolutions, with the majority measuring 225x225 pixels. All images are in RGB format and have been preprocessed to eliminate noise and resize appropriately. The dataset includes approximately 5,000 JPEG images for training, with a relatively balanced distribution across each class. The range of aspect ratios and resolutions presents a significant challenge for flower classification tasks [3].

Dataset II: consists of 11,200 flower images (in JPEG format) with pixel dimensions ranging from 178x256 to 648x500, representing seven distinct classes from Nadyana's "Flowers" dataset on Kaggle. This large collection provides sufficient data for training a deep learning model. Each image is annotated with labels that identify the species, creating clear targets for the model to predict. This organized labeling enhances the ability to train a robust and precise flower classification model [2].

Figure 2. Examples of Dataset-1 Images

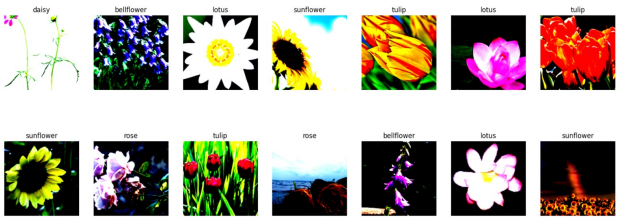

Figure 3. Examples of Dataset-2 Images

Dataset III:, accessible on Kaggle, is the most versatile of the datasets selected, comprising 13,716 images (in JPEG format) sized at 256x256 pixels, spanning 14 different classes. The images were captured under various conditions and lighting scenarios. This dataset supports the identification of multiple flower types, including carnations, irises, bluebells, golden English flowers, roses, fallen nephews, tulips, marigolds, dandelions, chrysanthemums, black-eyed daisies, water lilies, sunflowers, and daisies [1].

Name	Total Images	Classes
Flowers Dataset I	5k	5
Flowers Dataset II	11.2k	7
Flowers Dataset III	13.7k	14

The following preprocessing steps were applied to the images. The Resize transformation resizes the images to 256x256 pixels, reducing computational complexity and enhancing generalization. The CenterCrop transformation crops the central portion of the resized images to retain key features while removing background noise. The ToTensor() transformation converts the images into PyTorch tensors and scales the pixel values, preparing them for deep learning models. Finally, the Normalize transformation standardizes the input data by subtracting the mean and dividing by the standard deviation, thereby improving training performance and accuracy.

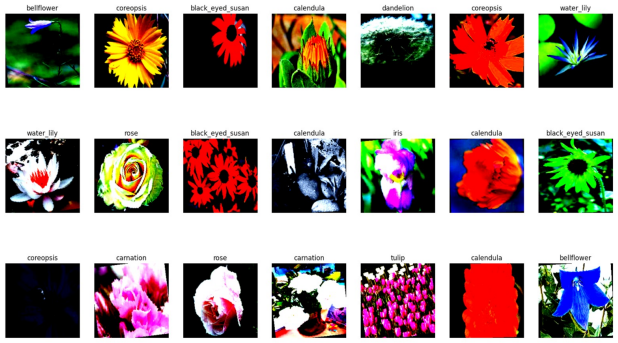

Figure 4. Examples of Dataset-3 Images

C.2. CNN Models

We have investigated ResNet18 due to its ability to address the vanishing gradient problem. ResNet18 is a powerful model thanks to several key components. The convolutional layers, which range from 18 to 152 layers or more, extract features from the input image, while residual connections facilitate efficient gradient propagation during training. The core feature of ResNet is the residual block. As the network deepens, it becomes more difficult to learn the true output, H(x), due to the large number of layers. Instead of directly learning the underlying mapping, the network learns the residual mapping, R(x). The inclusion of batch normalization layers helps to reduce internal covariate shifts, improving both accuracy and convergence speed. However, ResNet18 may not perform optimally if some flower classes lack sufficient image variety to detect complex patterns.

MobileNetV2, on the other hand, uses depthwise separable convolution layers and inverted residual blocks for flower image classification. Depthwise separable convolution includes both depthwise and pointwise convolutions, optimizing computation and reducing the parameter count without compromising accuracy. Inverted residual blocks utilize shortcut connections to preserve spatial resolution and enhance the model's representational capacity.

DenseNet121 is a convolutional neural network (CNN) architecture recognized for its distinctive dense connectivity, where each layer is connected to every preceding layer. This unique design facilitates improved gradient flow, enabling more efficient information propagation throughout the network. DenseNet121 consists of 121 layers, which include convolutional and pooling layers, and is known for its high performance in image classification tasks. Notably, it achieves excellent accuracy while maintaining a lower parameter count compared to traditional CNNs, making it a computationally efficient option. This characteristic enhances its ability to generalize effectively across different image recognition applications, making DenseNet121 a powerful choice for a wide range of computer vision tasks.

C.3. Optimization Algorithms

In our flower image classification project, we employed two commonly used optimization algorithms: Adam and SGD. While both algorithms aim to minimize the loss function, they differ in their approach. SGD updates the weights and biases of a neural network based on the gradient of the loss function with respect to those parameters, using a fixed learning rate.

In contrast, Adam adapts the learning rate for each parameter based on the mean and variance of the gradient, which can accelerate convergence. Overall, Adam is generally considered a more robust optimization algorithm, particularly for deep learning models with many parameters

or noisy data. However, SGD remains a popular choice for simpler models or datasets with relatively smooth gradients.

The choice of optimizer depends on the specific requirements of the image classification task, as both Adam and SGD have their advantages and limitations. For instance, Adam produced higher accuracy for ResNet18 and DenseNet121, typically converging faster, while SGD usually converges to more optimal solutions. In the case of MobileNetV2, the choice of optimizer plays a critical role due to its compact and efficient architecture with densely connected layers. While Adam is often preferred for its faster convergence, it can sometimes lead to sub-optimal solutions when training MobileNetV2 on complex datasets like flower classification. This is attributed to Adam's adaptive learning rate, which may cause it to settle in a local minimum or overfit, especially when fine-tuning the pre-trained model.

To mitigate these issues, SGD with momentum was explored for training DenseNet121. SGD, despite its slower convergence, often achieves better generalization on the validation set by allowing the model to converge to a more optimal solution. By leveraging the DenseNet121 architecture's strength in feature reuse and optimizing it with SGD, the model demonstrated improved performance for flower classification, achieving higher validation accuracy and reduced overfitting compared to Adam.

Thus, incorporating SGD into the training of DenseNet121 ensures that the model performs robustly, particularly for tasks involving fine-grained image classification like flower species differentiation.

D. Results

The primary objective of our project was to develop and optimize three convolutional neural networks (ResNet18, MobileNetV2, and Densenet121) for flower image classification. To achieve this, we followed a structured process that included dataset preprocessing, hyperparameter tuning, and model validation to ensure optimal performance.

Training Parameters Used				
Epoch	Train-Validation-Test	Loss fun.	Optim.	Batch
30	0.8/0.1/0.1	Cross-Entropy	Adam	32

To fine-tune our models, we carefully adjusted the hyperparameters. For all three models, we used the Adam optimizer with a learning rate of 0.001 and trained each model—ResNet18, MobileNetV2, and DenseNet121—for 30 epochs with a batch size of 32. The Cross-Entropy Loss function was selected for all models. A key decision in our project was the choice of optimizer, as this can significantly influence the results. We tested both Adam and SGD optimizers on ResNet18 and DenseNet121 and found that Adam provided better accuracy for these models, as it tends to converge faster and adaptively adjusts learning rates for individual parameters, which is beneficial for deeper architectures like ResNet18 and DenseNet121. However, for MobileNetV2, Adam did not perform as well because MobileNetV2 is designed as a lightweight architecture where careful gradient updates are crucial. SGD proved to be more effective for MobileNetV2, as it provides better control over the optimization process and avoids overfitting in models with fewer parameters.

For MobileNetV2, we observed that when trained on a large dataset, the model did not achieve the expected high accuracy. This issue was addressed by implementing a series of mitigation strategies. We fine-tuned the model by reducing the learning rate to prevent overfitting and ensure better convergence. Additionally, we experimented with different optimizers, switching from Adam to SGD, which helped improve the model's ability to find optimal solutions. These changes, combined with adjustments in batch size and training duration, contributed to the improvement of MobileNetV2's performance on the large dataset, bringing the accuracy closer to that of ResNet18.

To evaluate the effectiveness of our models, we employed various techniques. We monitored the training loss and accuracy over time, and plotted the validation loss and accuracy to track performance. Additionally, we assessed the models using key metrics such as precision, recall, and F1-score, providing a more comprehensive understanding of their performance and identifying areas for improvement. With the optimized hyperparameters and validation techniques, we achieved satisfactory performance in the flower image classification task.

During the development phase, we leveraged the real-time processing capabilities of graphics processing units (GPUs) to facilitate the training of large convolutional neural networks (CNNs). Initially, we conducted tests on cloud-based platforms like Kaggle and Colab. After evaluating their performance, we decided to continue using Kaggle as the primary platform for the remainder of the project. One additional advantage of using this platform was the ability to directly import the dataset from Kaggle.

E. Main Results

	ResNet-18			**MobileNetV2**			**DenseNet-121**		
	D1	D2	D3	D1	D2	D3	D1	D2	D3
Accuracy (%)	83.40	85.36	81.63	82.57	86.52	91.84	83.29	88.57	92.86
Precision (%)	82	87	85	83	87	94	82	89	93
Recall (%)	87	85	82	83	87	92	87	89	93
F1 Score	83	86	81	83	86	92	84	89	93

Overall, Densenet121 exhibited robust performance across all datasets, consistently achieving metrics in the

83–93% range. Additionally, we generated confusion matrices for each class in all datasets, providing deeper insights into each model's classification effectiveness and further guiding our tuning efforts.

Transfer learning is a technique where pre-trained models, initially trained on large datasets, are adapted to solve new tasks with smaller, task-specific datasets. This approach leverages learned features from the original task, saving time and computational resources. It is particularly effective in domains like computer vision, where general features learned from large datasets (such as ImageNet) can be fine-tuned for specific tasks, improving performance even with limited data.

In this experiment, we tested ResNet, MobileNet, and DenseNet on Dataset 1, which has five classes, in two configurations: trained from scratch and with transfer learning. Models were fine-tuned with a learning rate between 0.1 and 0.01 and a batch size of 32 for 30 epochs. The pre-trained weights were from ImageNet, and the models were evaluated using precision, recall, F1 score, and accuracy metrics to assess their performance.

Model	Precision (%)	Recall (%)	F1 Score	Accuracy (%)
ResNet	82	87	83	83
ResNet Transfer Learning	91	90	90	90
MobileNet	88	88	88	87
MobileNet Transfer Learning	91	90	90	90.40
DenseNet	82	87	84	83.3
DenseNet Transfer Learning	91	91	91	90.81

Table 1. Metrics Comparison: Transfer Learning Vs Train from Scratch Models on Dataset 1

Table 1 shows significant improvements in performance after applying transfer learning. ResNet's accuracy increased from 83% to 90%, MobileNet's from 87% to 90.4%, and DenseNet's from 83.3% to 90.81%. The precision, recall, and F1 scores also showed notable gains, confirming the effectiveness of transfer learning in boosting model performance on Dataset 1.

Transfer learning led to substantial improvements in all models, enhancing accuracy by 7% to 8%. These gains reflect the ability of pre-trained models to generalize better to new tasks, improving both classification and overall model performance. Transfer learning not only improved accuracy but also refined the models' ability to classify instances accurately across all five classes in the dataset.

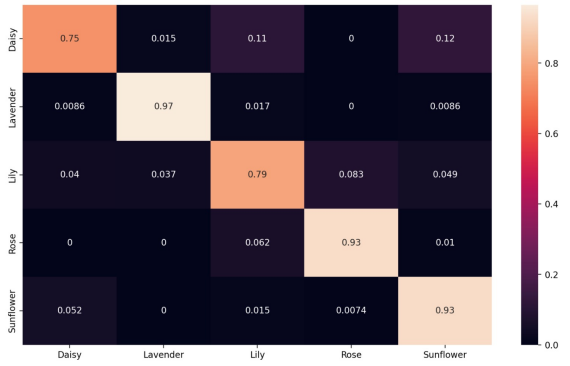

Figure 5. Resnet Dataset 1 Confusion matrix

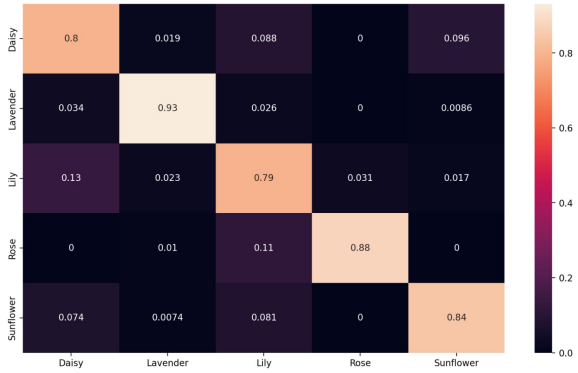

Figure 6. MobileNet Dataset 1 Confusion matrix

Figure 7. Merged image showing both images side by side.

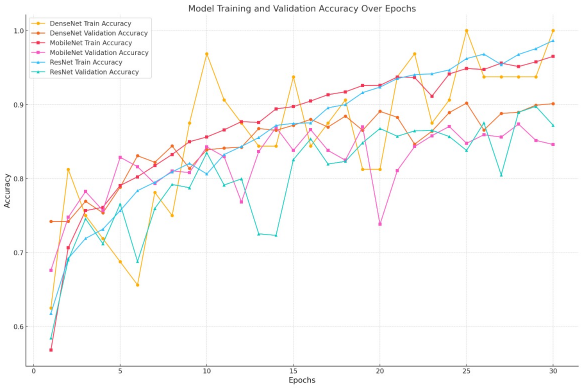

Figure 8. Comparison of Accuracy Across Different Models (Dataset 1, Classes)

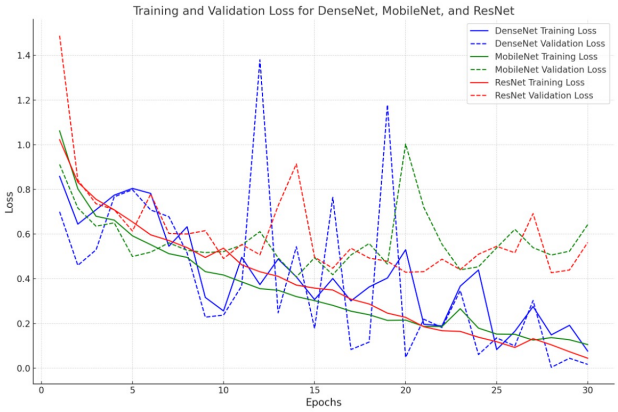

Figure 9. Comparison of Loss Across Different Models (Dataset 1, Classes)

E.1. Ablative Study

In artificial intelligence (AI), hyperparameters are parameters set before the training process begins and are crucial for model optimization. They include parameters like learning rate, batch size, and the number of epochs. The learning rate determines the size of the steps the model takes towards the minimum of the loss function. Batch size defines how many training examples are processed before the model's internal parameters are updated. A smaller batch size can result in noisier updates, which can help avoid local minima but also increase training time. On the other hand, a larger batch size offers smoother and more stable gradients, which can lead to faster convergence, but may also result in poorer generalization or higher memory consumption.

Figure 10. Impact of Varying Batch Sizes on ResNet Accuracy

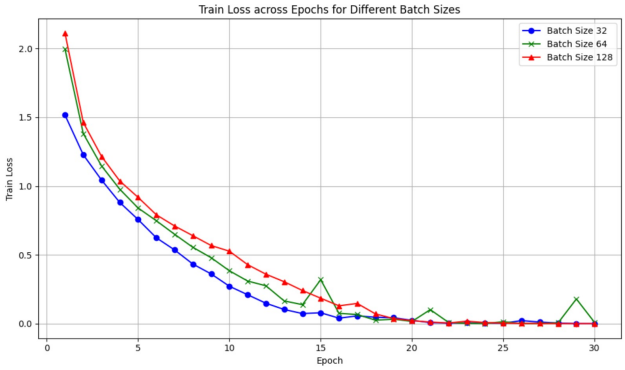

Figure 11. Losses Across Different Batch Sizes in ResNet Model Training

The comparison of train accuracy and train loss across varying batch sizes (32, 64, 128) reveals interesting trends in model behavior. For a batch size of 32, the model exhibited steady improvements in both accuracy and loss, with the train accuracy reaching 100% and the train loss diminishing to 0.00092 by the 30th epoch. With a batch size of 64, the model showed a slower improvement in accuracy, reaching 99.99% by epoch 30, but the loss remained slightly higher at 0.00926. When using a batch size of 128, the model achieved 99.99% accuracy, with a train loss of 0.0016 at the final epoch. The larger batch sizes resulted in more stable accuracy trends but may have sacrificed quicker convergence seen in smaller batch sizes. This suggests that batch size impacts the speed and stability of learning, with trade-offs between accuracy and convergence time.

The learning rate is a crucial hyperparameter in training neural networks, significantly influencing convergence speed and model performance. In the ResNet model with a batch size of 128, a learning rate of 0.1 initially results in rapid improvements in training accuracy, achieving a total accuracy of 90%. While the model shows consistent progress in training, the validation accuracy fluctuates, indicating potential overfitting or instability due to an overly aggressive learning rate. A learning rate of 0.01 provides more stable and consistent learning, with both training and validation accuracy increasing gradually, leading to better generalization. This configuration achieved a total accuracy of 69.39%. In contrast, a learning rate of 0.0001 significantly slows down the training process, with much lower training and validation accuracy, resulting in a total accuracy of 26.53%. This suggests that such a small learning rate is insufficient for effective optimization within the given number of epochs. Overall, the choice of learning rate plays a critical role in balancing training speed and model accuracy, emphasizing the need for careful tuning to achieve an optimal trade-off between convergence and generalization.

The comparison of model performance across different batch sizes (32, 64, and 128) with varying learning rates

Table 2. Hyperparameter Optimization - Learning Rate

Model and Dataset	Batch	Learning Rate	Loss	Accurac (%)
ResNet dataset 3	32	0.1	0.00092	86.734
ResNet dataset 3	32	0.01	0.1073	88.755
ResNet dataset 3	64	0.1	0.00926	88.775
ResNet dataset 3	64	0.01	0.93762	61.224
ResNet dataset 3	128	0.1	0.0016	90
ResNet dataset 3	128	0.01	0.29215	69.387
ResNet dataset 3	128	0.0001	2.11173	26.53

reveals distinct trends in both loss and accuracy, as shown in the table. For a batch size of 32, the model achieved the best performance with a learning rate of 0.1, resulting in the lowest loss of 0.00092 and an accuracy of 86.73%. This indicates that smaller batch sizes with higher learning rates contribute to better optimization and faster convergence.

For a batch size of 64, the model showed a slight decline in performance with the learning rate of 0.1, yielding a loss of 0.00926 and an accuracy of 88.78%. However, when the learning rate was reduced to 0.01, the model's performance deteriorated significantly, with a loss of 0.93762 and an accuracy of only 61.22%. This suggests that a larger batch size with a lower learning rate may hinder the model's ability to converge effectively.

With a batch size of 128, the model showed a more stable trend with the learning rate of 0.1, achieving a loss of 0.0016 and an accuracy of 90%. However, as the learning rate was reduced to 0.01, the model's accuracy dropped to 69.39% with a higher loss of 0.29215. When the learning rate was further decreased to 0.0001, the performance significantly worsened, with a loss of 2.11173 and an accuracy of just 26.53%, demonstrating that very small learning rates negatively impact learning, especially with larger batch sizes.

Overall, the results show that smaller batch sizes (32) with higher learning rates (0.1) yield the best performance, while larger batch sizes (128) with excessively low learning rates (0.0001) result in severe performance degradation.

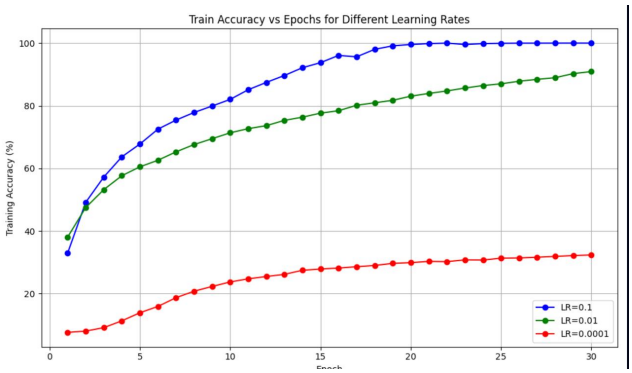

Figure 12. Comparison of Train Accuracy Across Different Learning Rates vs. Epochs on Dataset 3 (14 Classes) - ResNet

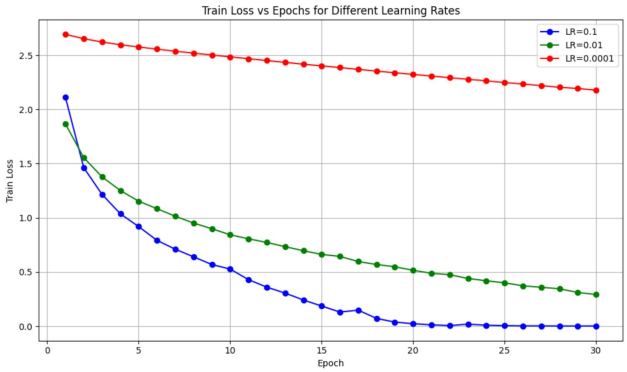

Figure 13. Comparison of Train Loss Across Different Learning Rates vs. Epochs on Dataset 3 (14 Classes) - ResNet

References

[1] Marquis03. Flower classification : 14 types of flower image classification. Kaggle, 2024. Available at: https://www.kaggle.com/datasets/marquis03/flower-classification. 3

[2] Nadyana. Flowers dataset. Kaggle, 2024. Available at: https://www.kaggle.com/datasets/nadyana/flowers. 2

[3] Utkarsh Saxena. Flower classification — 10 classes. Kaggle, 2024. Available at: https://www.kaggle.com/datasets/utkarshsaxenadn/flower-classification-5-classes-roselilyetc. 2

F. Appendix

D1: Dataset I
D2: Dataset II
D3: Dataset III
ROC: Receiver Operating Characteristic Curve
AUC: Area Under the ROC Curve