# 1. Introduction

## 1.1. Motivation

Classification tasks that humans perform naturally, such as distinguishing between similar objects, present significant challenges for computers. Extracting meaningful patterns from noisy and diverse image datasets requires robust learning methods. Deep learning has led to major advancements in image classification, enabling neural networks to achieve results. However, designing and fine-tuning a CNN involves critical choices, including selecting appropriate activation functions, loss functions, regularization strategies, hyperparameters and computational power.

## 1.2. Focus

This project focuses on building and training neural network models to classify images into three categories: cherry, strawberry, and tomato. This involves; implementing pre-processing techniques to enhance data quality, developing a baseline Multi-Layer Perceptron (MLP) to compare with more complex models, building and training a CNN model, and leveraging transfer learning with ResNet-18 to enhance model performance.

## 1.3. Aim

The primary aim of this project is to design, train, and evaluate neural networks for accurate image classification of cherry, strawberry, and tomato images. The study will compare the performance of a baseline MLP model with an advanced CNN model, which is then compared through optimizing transfer learning with ResNet-18. The focus is on identifying optimal strategies for achieving robust model performance.

## 1.4. Scope

This project investigates the image classification pipeline, including data pre-processing, augmentation, model selection, and optimization. It will involve testing different hyperparameters, loss functions, batch sizes, and optimization strategies. The report assesses transfer learning techniques to understand their impact on performance. The key focus is on comparing MLP vs CNN performance, evaluating transfer learning with ResNet-18, and optimizing for both accuracy and computational efficiency.

# 2. Background and Problem Statement

Neural networks are powerful tools for handling complex data. They consist of interconnected neurons organized into layers, where each neuron processes input data through an activation function and produces an output [1]. Neural networks learn patterns through feedforward propagation and are trained by adjusting weights and biases using backpropagation to minimize the loss function [1]. These principles form the foundation of both MLPs and CNNs, which are commonly used in machine learning.

## 2.1. Multi-Layer Perceptron (MLP)

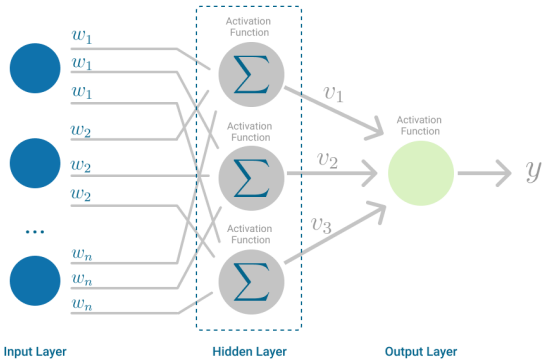



*Figure 1: Multi-Layer Perceptron Layout*

MLPs are a class of neural networks consisting of fully connected layers, where each neuron connects to every neuron in the next layer [1].

They use nonlinear activation functions to capture complex relationships in data. While MLPs are effective for structured data and small-scale tasks, they struggle with high-dimensional data like images [1]. MLPs treat input images as flattened vectors, which leads to a loss of spatial relationships between pixels [1]. For this reason, the MLP in this project serves as a baseline model as it provides insight into the limitations of traditional neural networks for image classification.

### 2.2. Convolutional Neural Network

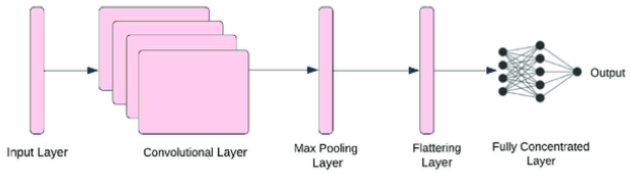



*Figure 2: CNN Layout*

CNNs are designed to handle grid-like data such as images by applying convolutional filters that learn patterns across spatial dimensions [2]. CNNs consist of convolutional layers, which extract feature maps by sliding kernels over the input image, pooling layers, down-sample feature maps to reduce dimensionality and improve computational efficiency, and fully connected layers, which combine extracted features to make final predictions [2]. CNNs are widely used in tasks such as image classification, object detection, and segmentation due to their ability to capture spatial hierarchies [3]. However, CNNs require large datasets and computational power to perform well [3], which poses challenges for smaller datasets like the one used in this project.

### 2.3. Transfer Learning with ResNet-18

Transfer learning addresses the challenge of limited data by leveraging pre-trained models trained on large datasets [3]. These models capture general visual patterns, such as edges, textures, and shapes, which can be transferred to new tasks [3]. For this project, we chose ResNet-18 which is a deep CNN with residual connections.

Residual connections mitigate the gradient problem by enabling gradients to flow through deeper layers [3]. This allows for efficient training even with deeper architectures. By freezing the earlier layers of ResNet-18 and retraining only the final fully connected layer, the model adapts to the classification of cherries, strawberries, and tomatoes. This approach saves time and computational resources while improving performance by using the general features learned.

### 2.4. MLP vs CNN vs Transfer Learning

The project compares the performance of three models: MLP, CNN, and CNN with transfer learning. Each model offers different trade-offs. MLP is simple and fast but it is unable to capture spatial patterns in images. CNN is effective at learning patterns from images but requires large datasets for optimal performance. Transfer Learning (ResNet-18) provides the benefits of CNNs while mitigating data scarcity issues, allowing for faster convergence and better generalization. This combination of models offers a comprehensive view of the trade-offs between simplicity, effectiveness and computational efficiency.

## 3. Methodology

### 3.1. Exploratory Data Analysis

Before constructing the classification models, Exploratory Data Analysis (EDA) was performed on the provided dataset. The dataset contained a total of 6,000 images across three classes: tomato, cherry, and strawberry, with 2,000 images per class. Of these, 4,500 images were designated for training purposes. The brief indicated that most images had been resized to 300x300 pixels. However, during the EDA, we

discovered inconsistencies in the dataset. The actual number of images per class was 1,495 rather than the expected 1,500. A total of 51 images were incorrectly resized to 300x300(17 cherry, 18 strawberry, and 16 tomato).

```
Number of Cherry images: 1495
Number of Strawberry images: 1495
Number of Tomato images: 1495
```

*Figure 3: Num. Class Images in Dataset*


*Figure 4: Irrelevant Image from Strawberry DS*

We manually removed irrelevant or mislabeled images (e.g. cartoons, merchandise, and images not representative of the target objects), ensuring the dataset more accurately reflected the categories. This decision assumed that removing noisy data would improve model performance by providing cleaner input.

```
Number of Extracted Cherry images: 1478
Number of Extracted Strawberry images: 1466
Number of Extracted Tomato images: 1475
```

*Figure 5: Num. Class Images Left after removal*

Additionally, we explored various image transformation and augmentation techniques, including:

- Sharpening: Enhanced edge definition using a kernel to increase feature distinction.
- Blurring: Applied smoothing with a 3x3 kernel to reduce noise.
- Contrast adjustment: Improved visibility of image details, making class features more prominent.
- Grayscale conversion: Removed color channels to simplify image information and reduce complexity.
- Edge detection (Canny and Sobel filters): Highlighted object outlines to aid in distinguishing key features.

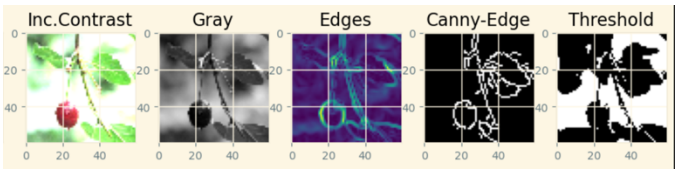



*Figure 6: Image augmentation techniques EDA*

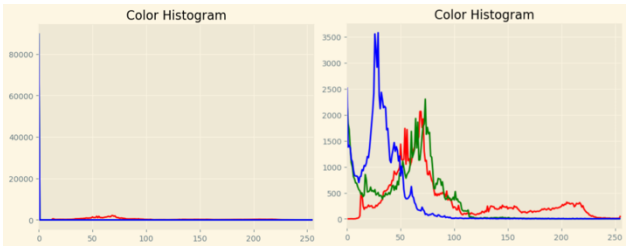



*Figure 7: Channel distribution between normal image and extracted RGB image*

These transformations aimed to improve the clarity of objects within the images and reduce background noise, aiding the model's understanding of each class. Image augmentation techniques [4], such as flipping, rotating, and zooming, were applied to enhance the dataset, helping the model generalize better to unseen data. This classical augmentation technique [4], involved geometric transformations and photometric shifts, generated additional training samples addressing the issue of limited dataset size.

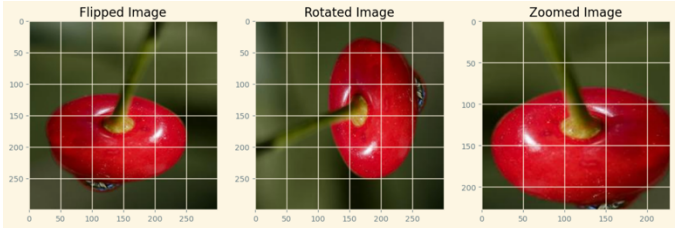



*Figure 8: Augmented Image Techniques*

We also experimented with Local Binary Pattern (LBP), a feature extraction method that captures texture information, as a potential way to improve the model's ability to recognize fine details within the images. Lastly, we explored the impact of removing specific color channels (fig7), blue and green, on the effectiveness of image transformations to determine if focusing on fewer color dimensions could enhance model accuracy.

While these augmentation and transformation techniques improved the dataset's quality, one limitation is that objects in images are not always prominently placed in the foreground, and may overlap with other objects, reducing the effectiveness of some techniques. Acknowledging this limitation means that the majority of these image manipulation tasks can only point the model in the right direction for understanding the underlying object, but not boost the model indirectly.

### 3.2. Preprocessing

The preprocessing step focused on ensuring consistency and preparing the dataset for optimal training performance. During the EDA, I identified that some images were incorrectly scaled, either not meeting the 300x300 dimension or containing noisy, irrelevant data.

The first processing method I used was for images that did not meet the 300x300 size requirement to be removed from the dataset when read in the code. After this cleanup, the datasets were equalized in size by limiting each class to the smallest dataset.

The second processing method I used was for all images to be rescaled to 200x200 pixels. This choice was made to reduce the model's computational load and improve training speed without significantly sacrificing image quality.

```
Number of cherry images: 1478
Number of strawberry images: 1466
Number of tomato images: 1475

Number of cherry images: 1474
Number of strawberry images: 1460
Number of tomato images: 1472

Balanced number of cherry images: 1460
Balanced number of strawberry images: 1460
Balanced number of tomato images: 1460
```

*Figure 9: Num. initial images, removed <300 images and balanced images*

Additionally, images were converted to float32 format [5], reducing memory consumption and improving processing speed [5]. While the uint8 format is often sufficient, the higher precision provided by float32 allowed for better accuracy in the model's predictions [5].

### 3.3. Data Augmentation

We implemented a total of nine different models, each focusing on a unique image transformation technique to explore how each type of augmentation affected performance. The goal was to evaluate which preprocessing or transformation techniques were most effective for improving classification accuracy across three fruit classes: cherry, strawberry, and tomato. The models (techniques) were:

- Augmentation Model
- Background Manipulation Model
- Blurred Model
- Canny Edge Model
- Sobel Edge Detection
- Contrast Model
- Linear Binary Pattern Model
- RGB Channel Removal Model
- Threshold Model

### 3.4. Purpose and Strategy

The motivation behind building these nine models was to explore the impact of various preprocessing and data augmentation strategies

on the neural networks' performance. By building multiple models and evaluating their performance on different versions of the preprocessed data, I was able to analyze the trade-offs between feature extraction techniques and augmentation strategies. The results from these models will guide the choice of image transformation and augmentation steps for the final CNN model.

### 3.5. Baseline (MLP)

The purpose of the MLP model was to establish baseline performance. This provided a reference point to compare against more advanced models like CNN and transfer learning. By measuring the performance of the MLP model we could determine whether the complexity of CNNs or data augmentation techniques improved the accuracy beyond what a simpler model could achieve.

MLP is useful for baseline comparisons as they treat input data as flattened vectors, without accounting for spatial relationships (like pixel arrangement) inherent in image data. This approach helps reveal how important these spatial features are for accurate classification. Each of the nine MLP models focused on different transformation techniques.

#### 3.5.1. MLP Results Across Models

| Transformation | Test Loss | Test Accuracy |
|---|---|---|
| Augmentation | 1.0374 | 45.56% |
| Background | 1.5605 | 36.99% |
| Blurred | 1.2463 | 45.05% |
| Canny-Edge | 1.8647 | 39.12% |
| Contrast | 1.1040 | 32.27% |
| LBP | 1.3053 | 36.99% |
| RGB | 1.1060 | 38.36% |
| Sobel Edge | 2.7348 | 42.01% |
| Thresholding | 2.2530 | 36.83% |

*Figure 10: Results of the different MLP models*

#### 3.5.2. Observations

Canny Edge Detection and Blurred models produced relatively higher test accuracy (~45%). This suggests that these transformations helped the MLP model focus on more relevant features. Contrast adjustment and thresholding performed poorly, likely due to a loss of important information during the transformation process. Even the best-performing MLP models, such as those using augmentation or blurred, struggled to exceed 45% accuracy. This aligns with expectations, as MLPs do not capture spatial dependencies in image data effectively. Some models such as Sobel Edge and Canny Edge achieved high accuracy during training but struggled on the test set, indicating potential overfitting.

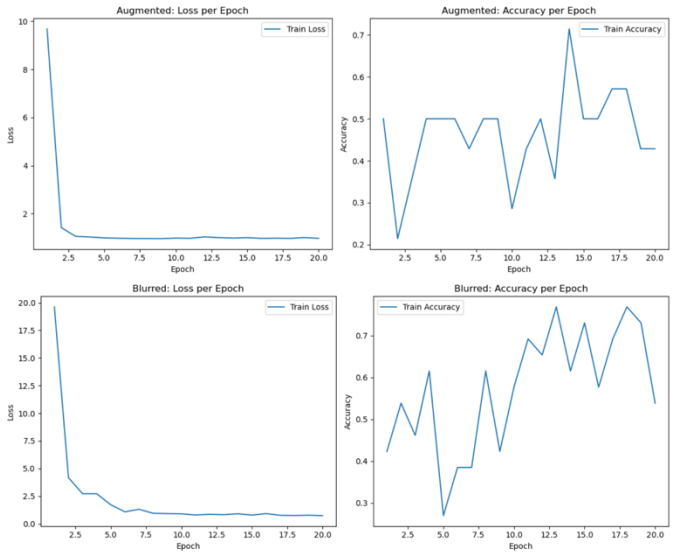



*Figure 11: Graphs of Augmented and Blurred Accuracy and Loss in Initial MLP*

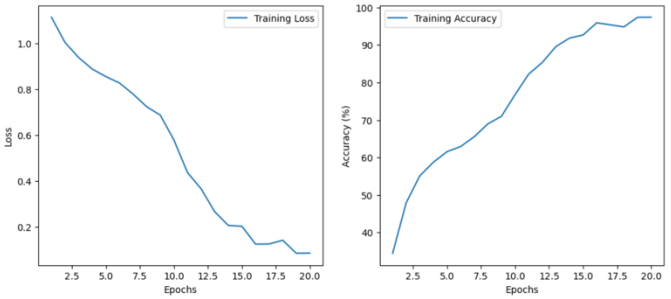
The MLP results highlighted the impact of different preprocessing techniques, with some transformations and augmentation submitting better performance than others. These insights reinforced the need for more advanced models such as CNNs or transfer learning which are better suited to take these techniques and apply them to models that would capture these complex patterns.

## 3.6. CNN Model

### 3.6.1. Initial CNN Model

The first CNN model used the blurred image transformation technique to serve as a baseline for future CNNs. It was designed with three convolutional layers followed by pooling operations, dropout for regularization, and two fully connected layers for classification. The primary goal was to improve the model's performance over the MLP baseline by leveraging CNN's ability to extract hierarchical features from image data.

The training performance of the CNN showed steady improvement across 20 epochs, with accuracy increasing from 34.44% in the first epoch to 97.39% by the final epoch on the training set. However, the test accuracy achieved was 55.86%, indicating potential overfitting and room for further improvements through hyperparameter tuning, and additional data augmentation.




```
Epoch [17/20], Loss: 0.1241, Accuracy: 95.37%
Epoch [18/20], Loss: 0.1404, Accuracy: 94.81%
Epoch [19/20], Loss: 0.0834, Accuracy: 97.36%
Epoch [20/20], Loss: 0.0846, Accuracy: 97.39%
Test Accuracy: 55.86%
```

*Figure 12: Initial CNN Model Results - Blurred*

### 3.6.2. Tuning CNN Model

From here we went on to investigate tweaking the initial CNN. This would be done by investigating multiple techniques, which we would then apply to different image transformation and augmentation models to find a better accuracy.

#### 3.6.2.1. Activation Functions

To enhance the performance of the CNN, we experimented with different activation functions; ReLU, Leaky ReLU, and ELU. The goal was to identify the activation function that best aids in the model's convergence and generalization, resulting in improved accuracy.

| Function | Train Acc. | Test Acc. |
|---|---|---|
| ReLU | 93.34% | 54.79% |
| Leaky ReLU | 98.86% | 53.27% |
| ELU | 96.09% | 51.67% |

*Figure 13: Results of Activation Function Tests*

The ReLU-based model performed well in training but did not generalize as expected suggesting some degree of overfitting. The Leaky ReLU model achieved similar training accuracy but faced challenges in improving test performance, indicating that the extra gradient did not significantly enhance generalization. The ELU function helped the model converge smoothly during training, the final test accuracy showed that it didn't significantly outperform the ReLU or Leaky ReLU models on unseen data.

While each activation function contributed to the effective training of the CNN model, ReLU showed the highest potential for this task in terms of both simplicity and convergence speed.

#### 3.6.2.2. Cross-Validation

Cross-validation is used to evaluate the generalization ability of machine learning models. Instead of relying on a single train-test split like the initial CNN and MLP models were using, cross-validation helped assess the performance across multiple subsets of the data, providing robust estimates of accuracy. We applied 3-fold cross-validation to better evaluate the CNN's model's performance. Each fold splits

the dataset into a training set and validation set, ensuring that every sample is used for validation exactly once. This helps mitigate overfitting and provides a more comprehensive view of how the model performs on unseen data.

| Type | Accuracy |
|---|---|
| No Fold | 55.86% |
| Fold (3) | 58.22% |

*Figure 14: Results of Cross-Validation Tests*

Using cross-validation provided a more reliable estimate of this performance by ensuring the model was trained and validated on multiple subsets of the data which reduced overfitting and provided a more stable indication of how the model might generalize to new data.

#### 3.6.2.3. Loss Functions

Investigated loss functions as they are a crucial component in training a neural network as they guide the optimization process by measuring how far off the model's predictions are from actual target labels. We investigated two loss functions, CrossEntropyLoss and NLLLoss.

| Function | Train Acc. | Test Acc. |
|---|---|---|
| CrossEntropyLoss | 99.51% | 56.47% |
| NLLLoss | 98.24% | 55.10% |

*Figure 15: Results of Loss Function Tests*

CrossEntropyLoss achieved a high training accuracy, suggesting the model learned the data well however the test accuracy indicates that it may be overfitting. NLLLoss indicates a strong accuracy but faces challenges generalizing to unseen data.

#### 3.6.2.4. Mini-Batch

Mini-batch sizes affect both the convergence speed and generalization. A mini-batch is a small subset of the dataset used to compute gradients and update the model's parameters in each iteration. Choosing the right batch size can improve the model performance and training efficiency.

| Size | Train Acc. | Test Acc. |
|---|---|---|
| B = 32 | 99.38% | 56.39% |
| B = 64 | 97.85% | 54.64% |
| B = 128 | 92.07% | 57.69% |

*Figure 16: Results of Mini-Batch Tests*

A batch size of 32 shows that small batch sizes generally offer more granular updates, helping the model converge faster, but may introduce noisy gradient estimates as the training accuracy was high though the test accuracy suggests potential overfitting. A batch size of 64 shows that increasing the batch size reduced the training noise but slightly lowered both training and test accuracies. A batch size of 128 shows that with a large batch size, the training stabilized further but convergence was slower. The test accuracy improved slightly indicating that large batch sizes may reduce overfitting but at the cost of longer training.

#### 3.6.2.5. Optimization

Optimization algorithms help in adjusting the model's parameters to minimize the loss function during training. Different optimizers influence the convergence rate, stability and final accuracy. We tested Adam, RMSProp and SGD optimization algorithms.

| Algorithm | Train Acc. | Test Acc. |
|---|---|---|
| Adam | 96.39% | 55.85% |
| RMSProp | 99.35% | 52.82% |
| SGD | 51.84% | 59.97% |

*Figure 17: Results of Optimization Tests*

Adam has fast convergence during training but showed moderate generalization to test data. Its adaptive learning rate makes it efficient for noisy gradients but may not generalize well to unseen data. RMSProp was able to handle objectives well and achieve good training accuracy however it struggled with overfitting which can be shown

in the test accuracy. SGD with momentum showed more stable generalization achieving the highest test accuracy, however, it required more epochs to converge.

### 3.7. Tuned CNN Model

Building on the initial CNN model and the insights gained from experimenting with various techniques, we applied the best-performing approaches to the original CNN and all other CNN models (augmentation, background, blurred, canny edge, contrast, LBP, RGB, Sobel edge and threshold). For consistency and optimal performance, we configured the models as follows:

- Optimization Technique: Adam
- Batch Size: 32
- Loss Function: CrossEntropyLoss
- K-Fold Cross Validation: 3 folds
- Activation Function: ReLU

FINAL MODEL

```
class AugmentedModel(nn.Module):
    def __init__(self, num_classes=3):
        super(AugmentedModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.dropout = nn.Dropout(0.25)
        self.fc1 = nn.Linear(64 * 37 * 37, 128)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = x.view(-1, 64 * 37 * 37)
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

[18]

*Figure 18: Tuned Augmentation CNN Model*

| Model | Validation Accuracy |
|---|---|
| Augmentation | 74.76% |
| Background | 39.00% |
| Blurred | 56.69% |
| Canny Edge | 40.71% |
| Contrast | 63.49% |
| LBP | 40.27% |
| RGB | 41.51% |
| Sobel Edge | 48.97% |
| Threshold | 39.09% |

*Figure 19: Results of Tuned CNN Models*

```
Epoch [12/20], Loss: 0.0900, Accuracy: 96.89%
Epoch [13/20], Loss: 0.0809, Accuracy: 97.03%
Epoch [14/20], Loss: 0.0861, Accuracy: 96.93%
Epoch [15/20], Loss: 0.0600, Accuracy: 98.00%
Epoch [16/20], Loss: 0.0629, Accuracy: 97.95%
Epoch [17/20], Loss: 0.0456, Accuracy: 98.53%
Epoch [18/20], Loss: 0.0615, Accuracy: 98.04%
Epoch [19/20], Loss: 0.0471, Accuracy: 98.33%
Epoch [20/20], Loss: 0.0636, Accuracy: 97.95%

Accuracy: 74.76%
```

*Figure 20: Results of Augmentation Tuned CNN*

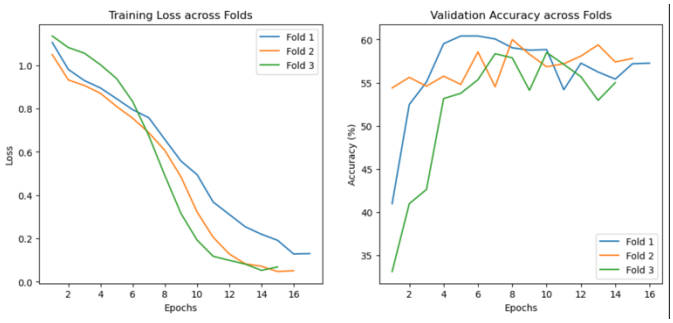



*Figure 21: Results of Blurred Tuned CNN*

The augmented models generally performed better than the baseline models, with the Augmentation and Contrast models yielding the highest validation accuracies respectively. However, other models such as Background and Threshold faced challenges in maintaining consistent performance, as evidenced by early stopping and lower validation accuracies. These results highlight the importance of fine-tuning transformation techniques to achieve optimal performance in CNN models.

### 3.8. Transfer Model

Following the optimization of the CNN, we explored additional techniques to further enhance the best-performing model, the Augmentation Model. Specifically, we implemented transfer learning by leveraging a ResNet-18 model. Transfer learning allowed the utilization of a pre-trained ResNet-18 model to extract high-level features from images, integrating it with my existing dataset for classification tasks. This approach improves

training efficiency, reduces the likelihood of overfitting, and enhances accuracy, especially with smaller datasets. We applied the same image resizing, normalization, augmentation and filtering techniques used in my tuned CNN model.

We used the ResNet-18 architecture with all layers except the final fully connected layer frozen to retain the learned features of the pre-trained model, reducing training time and preventing overfitting. The final fully connected layer was replaced with a custom layer to classify the three classes; cherry, strawberry and tomato. The model was trained using Adam optimizer with a learning rate of 0.001, batch size of 32, and 3-fold cross-validation to ensure robust evaluation.

| Step | Validation Accuracy | Last Epoch |
|---|---|---|
| Fold 1 | 90.11% | 20 |
| Fold 2 | 90.21% | 12 |
| Fold 3 | 89.45% | 16 |
| FINAL avg. | 89.92% | |

*Figure 22: Results of the Transfer Model*

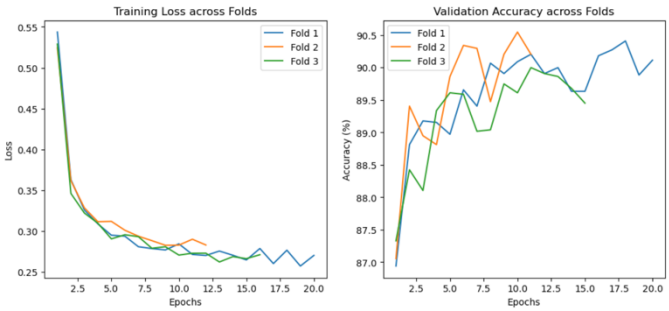



*Figure 23: Graph Results of Loss and Accuracy In Transfer Model*

The transfer learning model showed steady convergence and high performance across all folds, with early stopping applied to avoid overfitting, based on loss. Across all folds the model achieved validation accuracies around 90% on the majority of folds. Training loss decreased steadily, with validation accuracy stabilizing across early epochs, indicating the model learned efficiently from the data. Applied early stopping, which occurred on 3 folds which prevented unnecessary training and saved time by stopping when no further improvement was made.

## 4. Discussion

MLP served as the starting point which offered an easily interpretable but limited model. MLP only achieved moderate test accuracies with the highest around 45% for the augmentation and blurred models. These results confirm the MLP's inability to effectively capture hierarchies present in image data. The CNN significantly outperformed the MLP, showing a training accuracy of 97.39 and a test accuracy of 55.86% for just the Blurred model. The model's ability to learn hierarchical patterns improved image classification however there were signs of overfitting as test performance was behind the training results. By exploring activation functions, cross-validation, batch sizes, and optimizers, we were able to fine-tune the CNN and achieve improved results. The augmentation model with Adam optimizer and a batch size of 32 was the best-performing version, achieving 74.76% validation accuracy. Cross-validation provided more stable results and reduced overfitting, as evidenced by the average validation accuracy across folds being higher than the test accuracy from the initial CNN. Transfer learning however was the most effective strategy, with ResNet-18 achieving an average validation accuracy of 89.92% across all folds. The ability to leverage pre-trained layers for feature extraction provided benefits.

While we explored various augmentation and transformation techniques, such as background manipulation, edge detection, and blurring, these methods were not consistently effective. Many transformed models struggled with generalization, as indicated by lower validation and test accuracies. Although some techniques, like augmentation and contrast, enhanced

performance, others (background, thresholding) gave inconsistent results with several models experiencing early stopping due to poor validation performance. These findings highlight the limitations of relying solely on image manipulation and emphasize the importance of leveraging advanced architectures like CNNs and transfer learning to achieve robust classification outcomes.

## 5. Conclusions

The progression from MLP to CNN to ResNet-18 illustrates the trade-offs between simplicity, performance and computational efficiency. While the MLP provided a baseline and confirmed the need for advanced models, the CNN demonstrated the importance of tuning hyperparameters to achieve better generalization. However, transfer learning was the most efficient and accurate approach, as it was more effective in limited and small datasets. This highlights the value of leveraging pre-trained models, especially with smaller datasets, to achieve robust results while minimizing training time. The majority of image augmentation and transformation techniques used were mostly ineffective, apart from a select few. Utilizing classical augmentation techniques involving flipping and rotating images proved worthwhile in a limited dataset however Sobel and Canny Edge detection proved to be inefficient due to the amount of background and foreground noise in each image. Further analyzing image manipulation techniques would be useful for future work, where stacking augmentation techniques on top of each other, i.e. Sobel-Edge, LBP, flipped image, and rotated image, might provide CNN models with a better understanding of the features it's trying to understand. The findings emphasize that while advanced models require careful optimization, transfer learning offers an efficient high-performance image classification.

## 6. References

[1] S. Jaiswal, "Multilayer Perceptrons in Machine Learning: A Comprehensive Guide," *Datacamp.com*, Feb. 07, 2024. https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning

[2] IBM, "What are Convolutional Neural Networks? | IBM," *www.ibm.com*, 2024. https://www.ibm.com/topics/convolutional-neural-networks

[3] C. Gu and M. Lee, "Deep Transfer Learning Using Real-World Image Features for Medical Image Classification, with a Case Study on Pneumonia X-ray Images," *Bioengineering*, vol. 11, no. 4, p. 406, Apr. 2024, doi: https://doi.org/10.3390/bioengineering11040406.

[4] N. E. Khalifa, M. Loey, and S. Mirjalili, "A comprehensive survey of recent trends in deep learning for digital images augmentation," *Artificial Intelligence Review*, Sep. 2021, doi: https://doi.org/10.1007/s10462-021-10066-4.

## 7. References for Code Implementation

- https://chatgpt.com (used to help with basic implementation and debugging across code)
- https://www.youtube.com/watch?v=w1UsKanMatM (Explanation on residual networks and how it is implemented)
- https://www.youtube.com/watch?v=s_hDL2fGvow&list=PLZsOBAyNTZwYx-7GylDo3LSYpSompzsqW&index=6 (understanding of Python image processing)
- https://www.youtube.com/watch?v=kSqxn6zGE0c (understanding Python image processing for EDA and pre-processing)