COMP309, 2024

Project Report

Image Classification

Brett P

ID: 300635306

Contents

1 Introduction

Deep convolutional neural networks **(CNN's)** are foundational to modern computer vision. They have allowed for advancements in all areas from facial recognition to autonomous driving. This project explores three different CNN models, and their ability in image classification on a dataset of tomatoes, cherries, and strawberries. This is a complex task as the three fruits are all of similar colors. The three models architectures are:

- A custom CNN inspired by the VGG architecture[1].
- A ResNet50 model trained using transfer learning.
- An ensemble model built by stacking the two previous models.

Through experimenting with data augmentation, learning rates, and model tuning, each model's performance will be assessed on accuracy, loss and performance in order to find the optimal solution to the classification problem.

2 Problem Investigation

Before analysing the results of the 3 different models, we will first look at the insights gained from the data, and the choices these insights led to.

2.1 Exploratory Data Analysis (EDA)

Previously we have primarily conducted EDA on structured data, which allows for a straight forward pipeline, so for this dataset the primary focus was on distributions, outliers, and other characteristics such as color/brightness/noise. This was done using python, with the **PyTorch** library as well as utilizing the excellent 'CleanVision'[2] library to audit the dataset. These were the results:

2.1.1 Dataset Overview

The dataset consists of **4485** images categorized into three classes: tomatoes, cherries, and strawberries. Most images are of size 300x300 pixels and are stored in JPEG format.

2.1.2 Summary Statistics

Table 1 and 2 present the summary statistics for each fruit category, indicating that the data is overall uniformly distributed.

Category	**Num. Img**	**Mean Dimensions**
Tomatoes	1495	299.54x299.33
Cherries	1495	299.54x299.33
Strawberries	1495	299.54x299.33

Table 1: Summary Statistics of the Dataset

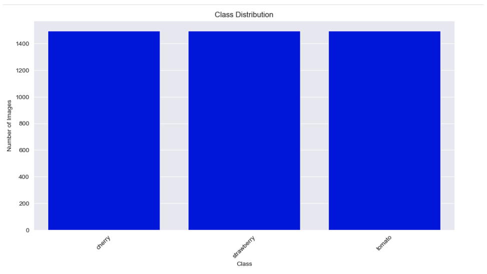


Table 2: Class Distributions

2.1.3 Data Visualization

After analyzing the initial summary statistics, it was clear that overall the data was relatively clean. This figure showing random images from the dataset further proves this point:

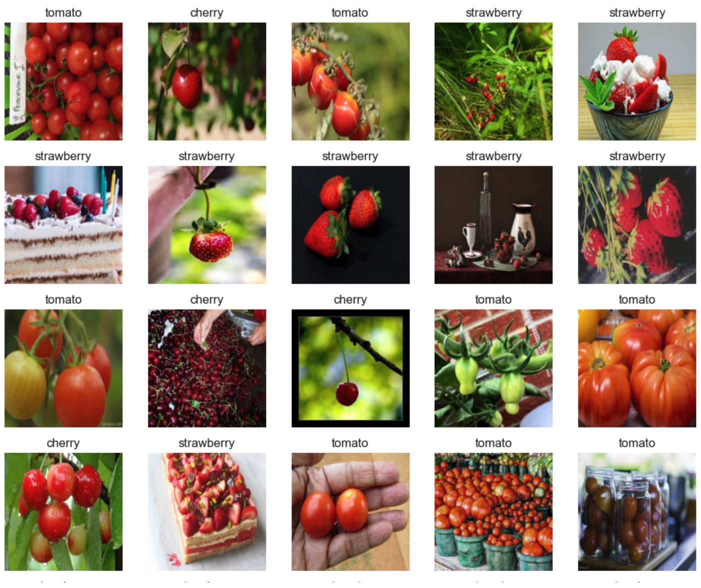
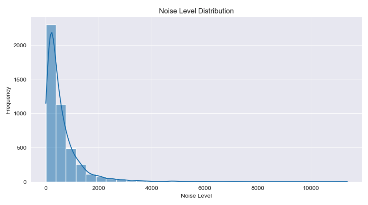
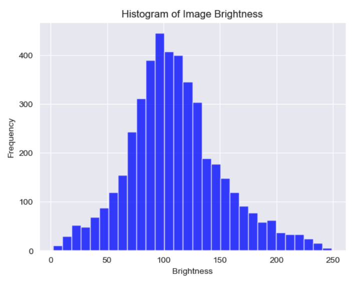


Figure 1: Example images over different classes

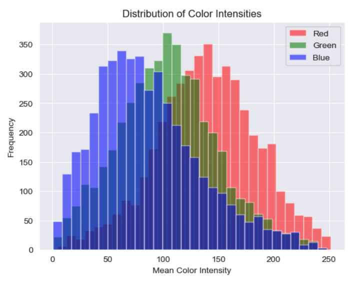
Using methods to analyze noise, brightness, and color outliers returned mixed sets of good and bad images (some that shouldn't be removed) . Overall the distributions of noise, brightness, and color intensities was as expected for this dataset, with normally distributed brightness, and color intensities (red being the most intense), and with a peak of 0 for noise level (skewed to the right).



(a) Noise Level



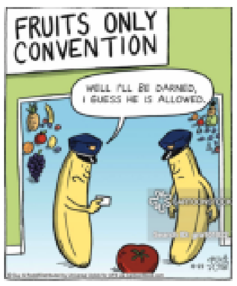
(b) Brightness



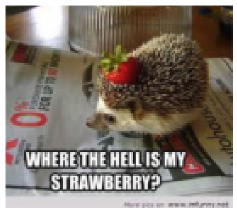
(c) Color Intensity

Figure 2: Informative Distributions

Figure 2 indicates overall integrity of the dataset, but doesn't clearly inform us on where the outliers like these (which could lead our models to learn incorrect patterns) are in the dataset:



(a) Banana??



(b) Hedgehog??

2.1.4 CleanVision

To solve the previous issue, the open-source tool **CleanVision** was used for locating issues and their respective indexes. This was the resulting table from the CleanVision Audit:

Table 3: CleanVision Audit Results

Issue Type	Number of Images
Odd Size	53
Exact Duplicates	39
Near Duplicates	32
Blurry	9
Dark	3
Low Information	3
Light	0
Odd Aspect Ratio	0
Grayscale	0

These numbers were manually verified to be correct by displaying the related images from the sets of issue types, e.g. the *exact duplicate* images:

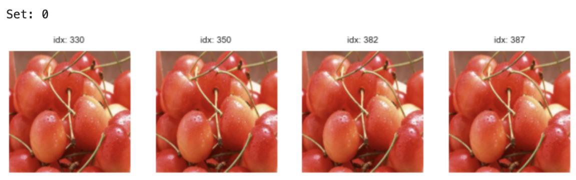


Figure 4: Verification that the CleanVision exact duplicates are infact duplicates

However, out of the 6 *issue* classes, the only relevant classes to our project were: Odd Size, Exact Duplicates, and Near Duplicates. This is because on manual inspection, the *Blurry* images weren't excessively blurry, and the Dark/Low information images contained 1 instance of each class (*looked to be put there purposely*). Leaving these features could help to increase model robustness.

2.2 Data Pre-processing

Effective data pre-processing is vital for improving the performance and generalization capabilities of machine learning models. This section outlines the steps taken to clean, augment, normalize, and split the dataset to prepare it for training.

2.2.1 Data Cleaning

Data cleaning involves identifying and fixing errors or inconsistencies in the dataset to make sure that the model learns from high-quality data. Using *CleanVision's* insights from the EDA:

Removal of Duplicates Duplicates were identified, verified and removed from the dataset to prevent the model from over fitting on redundant information. More specifically, 39 exact duplicates and 32 near duplicates were excluded *(whilst ensuring that the original images were left in place)*.

Removal of Odd Sized Images Upon manual inspection of the *odd sized* issue set, the odd images e.g. banana's, dog's, strawberry shortcake logo were removed. This ensures that the model doesn't learn patterns of features that aren't required.

Result Data cleaning using *CleanVision* removed 89 images leaving **4,396** images in total.

2.2.2 Data Transformation Pipeline

The primary preprocessing steps were applied in the form of a **torchvision.transforms** block, where image resizing, data augmentation, normalization and conversion to the tensor data type occurred. These techniques assist the model in learning, and help to avoid over fitting.

Image Resizing All images were resized to 224 x 224 pixels to give the **tensor.shape** value of *(3, 224, 224)*. This was done to ensure consistency and is the required image shape when using the ResNet50 model.

Random Horizontal Flips Randomly flipping images horizontally was used to simulate different orientations of the fruits. This allows the model to learn features no matter the fruit's direction and also could make predicting on a cleaner *test set* easier.

Random Rotations Applying random rotations (±10 degrees) introduced variations in the angle of the images. This allows the model to generalize better to images where fruits might appear at different angles.

Normalization Normalization is key for stabilizing and speeding up the training process. Each image was normalized using the following mean and standard deviation values for the RGB channels:

- **Mean**: [0.485, 0.456, 0.406]
- **Standard Deviation**: [0.229, 0.224, 0.225]

These values are commonly used for models pretrained on ImageNet[3] and help in aligning the input data distribution with those of the pretrained weights.

3 Methodology

This next section goes over the methodologies used in training the convolutional neural networks (CNNs) for image classification, looking at model architecture, loss function selection, optimization strategies, and evaluation metrics. *Note: this will be looking at techniques when optimizing the custom CNN, which was not the final model.*

3.1 Data Splitting and DataLoaders

The dataset was divided into training and validation sets to evaluate the model's performance on unseen data. An 80/20 split was used, which leaves 80% of the data for training and 20% for validation/test. This partitioning makes sure that the model is trained on a large portion of the data while also keeping a significant subset for performance evaluation. After preprocessing and splitting these are the number of images in each set:

- **Total Images**: 4,396 images
- **Training Set**: 3,516 images
- **Validation Set**: 880 images

DataLoaders torch.DataLoaders were applied on each set with a batch size of 32 and `num_workers` set to 4 to utilize parallel data loading. This means we will be conducting batch gradient descent in the optimization stage. The batch size of 32 is a commonly used size, and gained greater accuracy than smaller or larger sizes. The primary purpose of utilizing batches is to allow data to be processed in smaller, more manageable chunks; allowing the model to train faster and the optimizer to update weights more frequently.

3.2 Loss Function

A loss function is an integral part of a Neural Network; at its core, it measures how well the algorithm models the dataset. The goal is to minimize this value, which can be achieved through gradient descent and other optimization techniques during back propagation. For the three CNNs, the **Categorical Cross-Entropy Loss** function from *PyTorch*[4] was used.

Cross-Entropy Loss is suited for multi-class classification tasks as it measures the difference between the predicted probability distribution and the true distribution of labels. Mathematically, for a single instance/batch, it is defined as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(p_i)$$

where:

- C is the number of classes.
- y_i is the binary indicator (0 or 1) if class label i is the correct classification.
- p_i is the predicted probability of the input belonging to class i.

In general **Cross-Entropy Loss** works best with a *Soft-Max* loss function. But for this project it was found that it resulted in high accuracy and low loss with an alternative loss function which will be discussed further on. This could be due to **PyTorch's nn.CrossEntropyLoss()** inherently combing the *Soft-Max* activation in the its calculation as it requires *raw logits* when performing loss computation.

3.3 Optimization Methods

Optimization methods in deep learning are used to minimize a loss function *(Cross Entropy Loss in this case)*, which results in better predictions. This is generally done through some form of gradient descent *(Batch gradient descent)* where an optimizer attempts to find a local or global **optimum** by updating the network's weights during back-propagation.

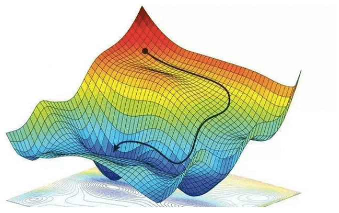


Figure 5: Gradient descent example

Due to this, selecting the correct optimizer is essential for improving classification accuracy and reducing the loss. For the *custom CNN* and *Ensemble* models, the **Adam**[7] optimizer with a learning rate of 0.001 and weight decay of 1×10^{-5} was found to produce the greatest accuracy and lowest loss. Adam combines the advantages of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp), by keeping per-parameter learning rates and adapting them based on the first and second moments of the gradients.
In contrast, for the *ResNet50* model, the **Stochastic Gradient Descent (SGD)**[6] optimizer with a learning rate of 0.01 and momentum of 0.9 was utilized. **SGD** is a traditional optimizer known for its simplicity and effectiveness, especially in large-scale and deep networks. The addition of momentum helps accelerate gradient vectors in the right directions, which leads to faster convergence.
It should be noted that in the training loop, a **learning rate scheduler** was applied to both optimizers, specifically **StepLR(self.optimizer, step_size=7, gamma=0.1)**. This scheduler reduces the *learning rate* by a factor of 0.1 every 7 epochs. Adjusting the learning rate this way helps to fine-tune the convergence process, which prevents the optimizer from overshooting minima and allowing for more stable convergence in later stages of training.

3.4 Regularization Strategies

To prevent over fitting and improve the generalization of the CNN models, a few regularization techniques were used:

- **Batch Normalization**: Implemented after each convolutional layer, batch normalization normalizes the output of the previous activation layer. This technique stabilizes and accelerates the training process, allowing for higher learning rates and reducing sensitivity to initialization.
- **Weight Decay**: Applied as a form of L2 regularization in the Adam optimizer, weight decay penalizes large weights by adding a term proportional to the square of the magnitude of the weights to the loss function. This stops the model from fitting over-complex patterns and helps in maintaining simpler models which generalize better to unseen data.
- **Dropout**: Dropout is another regularization technique that attempts to prevent over fitting by randomly setting input units to 0. Through further research of lecture material it was found that combining **combining Batch Normalization and Dropout layers** was not recommended as it can lead to unstable behavior and predictions. This lead to the Dropout layer being removed.

3.5 Activation Functions

Activation functions are mathematical equations that determine the output of a neuron in a neural network based on its input. The purpose is to introduce **non-linearity** to the output of a neuron, which is crucial when learning complex patterns such as in classifying images in a CNN. In all three of the CNNs, the **Rectified Linear Unit (ReLU)** activation was implemented. **Rectified Linear Unit (ReLU)** is defined mathematically as:

$$\text{ReLU}(x) = \max(0, x)$$

where x is the input to the neuron.

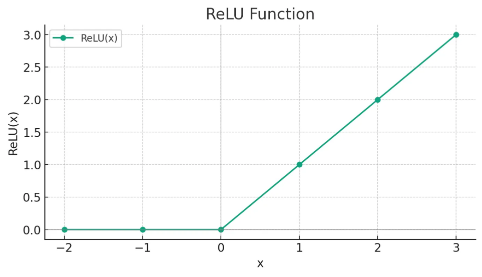


Figure 6: Graph of the ReLU activation function[8]

The ReLU activation function introduces sparsity by outputting zero for any negative input, which helps in reducing the likelihood of vanishing gradients and speeds up the convergence of the training process. Additionally, ReLU is computationally efficient as it involves simple thresholding at zero, which makes it suitable for deep neural networks with multiple layers. Using ReLU resulted in the greatest accuracy and lowest loss functions in the CNN's compared to the Sigmoid, or SoftMax functions.
Example Calculation: Applying ReLU to a set of inputs:

$$\text{ReLU}([-2, -1, 0, 1, 2]) = [0, 0, 0, 1, 2]$$

This transformation enables the network to model complex, non-linear relationships within the data, enhancing its classification capabilities.

3.6 Obtaining Further Images

To further enrich the dataset, over 300 new images were obtained. The images were appended to each class to first balance out the class counts, and then an additional 100 images were added to each fruit class. The images were obtained from the Fruit-360 dataset[9]. The images contained similar images which were rotated at different angles; helping to reduce overfitting.

3.7 Transfer Learning

To obtain the final model, transfer learning was implemented. Transfer learning involves using a pretrained model *(e.g., pretrained on the ImageNet dataset)* and leveraging the knowledge gained from one task to improve performance on another task, as illustrated in Figure 9.

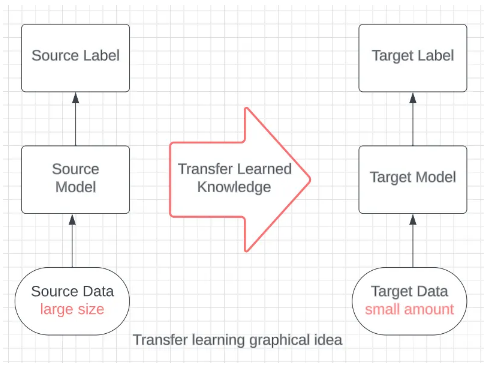


Figure 7: Transfer Learning visualization[10]

ResNet50 is a convolutional neural network well known for its depth and performance in image classification tasks. **ResNet50** was pretrained on the ImageNet-1k dataset with input images resized to 224 x 224 pixels, aligning with the preprocessing steps used in this project. In my approach, all layers of the pretrained ResNet50 model were frozen to retain the learned feature representations. Only the final fully connected layer was replaced with a new layer configured to output three classes, *corresponding to the fruit classification task.* This strategy utilizes the robust feature extraction capabilities of ResNet50 while still allowing the model to adapt specifically to our dataset. This resulted in highly improved accuracy, lower loss, and more efficient learning time.

3.8 Ensemble Learning

Ensemble learning is a machine learning technique where two or more models are stacked to gain improved prediction accuracy.

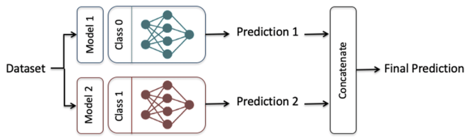


Figure 8: Ensemble Learning visualization

Stacking Custom CNN with ResNet50 Similar to the figure illustrated above, the two models were stacked together to form an ensemble aimed at improving prediction accuracy. This resulted in an improvement of 0.5-1%. However, the ensemble approach was not used as the final model as it significantly increased computational time (causing crashes when training) and led accuracies to be stuck at 100% when loading the .pth file.

4 Summary and Discussion

This section summarizes the performance of the different neural network models discussed in this report, comparing the baseline Multi-Layer Perceptron (MLP), ResNet50 with transfer learning, and the Custom CNN. The comparison looks at model structures, training times, classification accuracies, and discusses the observed outcomes.

4.1 Model Structures

4.1.1 Baseline Multi-Layer Perceptron (MLP)

The baseline model used in this project is a Multi-Layer Perceptron (MLP), a fundamental neural network architecture made up of fully connected layers. The MLP was structured with:

- **Input Layer**: Accepts flattened 224x224 RGB images, resulting in 150,528 input features.
- **Hidden Layers***: Two hidden layers with 512 and 256 neurons respectively, each followed by ReLU activation functions to introduce non-linearity.
- **Output Layer**: A final fully connected layer with 3 neurons corresponding to the three target classes, utilizing the Cross-Entropy Loss function for multi-class classification.

4.1.2 ResNet50 with Transfer Learning (Best CNN)

ResNet50, a deep residual network pretrained on the ImageNet dataset, using transfer learning was the best performing model. This is it's architecture and structure in the transfer learnt version:

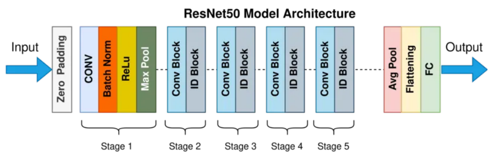


Figure 9: ResNet50 Architecture

- **Pretrained Backbone**: Uses the convolutional layers of ResNet50, which have learned strong feature representations from extensive image data.
- **Modified Fully Connected Layer**: The original fully connected layer was replaced with a new layer for the three target classes.
- **Layer Freezing**: All layers except the final fully connected layer were frozen to keep the pretrained features, this improved performance and reduced overfitting.

4.1.3 Custom Convolutional Neural Network (Custom CNN)

Before utilizing transfer learning with the final ResNet50 model, an initial **CNN** was created, based on the VGG architecture:

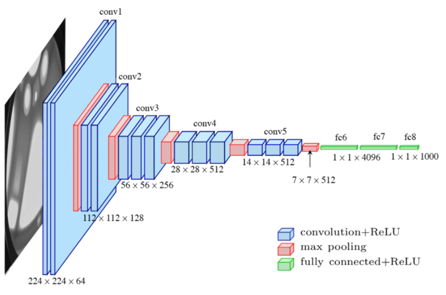
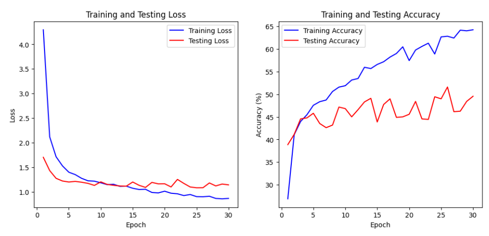


Figure 10: VGG Architecture *(inspiration for Custom CNN)*

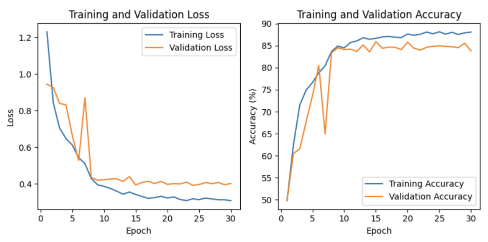
- **Convolutional Layers**: Five convolutional layers with increasing filter sizes (32, 64, 128, 256, 512), each followed by Batch Normalization and ReLU activation.
- **Pooling Layers**: Max Pooling layers after each convolutional block to reduce spatial dimensions.
- **Fully Connected Layers**: Three fully connected layers transitioning from 512 features to 256, and finally to 3 output classes.

4.2 Results Comparison

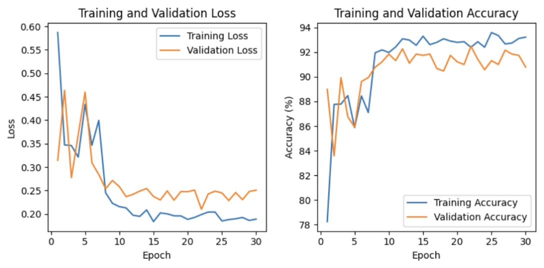
The performance of the three models was evaluated based on training time, classification accuracy, and loss metrics over 30 epochs. Here are the results:



(a) Baseline Performance



(b) Custom CNN Performance



(c) ResNet50 Performance

Figure 11: Model Performance Evaluation

4.2.1 Training Time (on Kaggle P100 accelerator)

- **MLP**: Total training time: 00:04:13
- **Custom CNN**: Total training time: 00:04:56
- **ResNet50 Transfer Learning**: Total training time: 00:05:01

4.2.2 Accuracy

- **MLP**: Best val accuracy: 51.9%
- **Custom CNN**: Best val accuracy: 86.0%
- **ResNet50 Transfer Learning**: Best val accuracy: 92.5%

4.2.3 Loss Metrics

- **MLP**: Best loss: 0.8705
- **Custom CNN**: Best loss: 0.3084
- **ResNet50 Transfer Learning**: Best loss: 0.1839

Table 4: ResNet50 Further Performance Metrics

Class	**Precision**	**Recall**	**F1-Score**	**Accuracy**
Cherry	0.93	0.89	0.91	88.79%
Strawberry	0.92	0.91	0.91	91.08%
Tomato	0.88	0.93	0.90	92.56%

4.3 Discussion

The training statistics of the three models shows the differences in their performance metrics. The **ResNet50** model achieved the highest validation accuracy of **92.5%** and the lowest loss of **0.1839**, outperforming both the **Custom CNN** and the **MLP** models. The **Custom CNN** had impressive performance with a validation accuracy of **86.0%** and a loss of **0.3084**, which surpasses the **MLP**, which had a validation accuracy of only **51.9%** and a loss of **0.8705**.

Training Time For training time, all models had similar results, with the ResNet50 model needing slightly more time (**00:05:01**) compared to the Custom CNN (**00:04:56**) and the MLP (**00:04:13**). This slight difference is outweighed by it's superior performance.

Classification report The classification report for the ResNet50 model highlights its performance over all classes, with high precision, recall, and F1-scores. These metrics show that the ResNet50 model generalizes well across each of the 3 different classes.

Striking Differences The most obvious difference between the poorly performing **Baseline** model and the high performing **ResNet50** model lies in in the shape of the loss function curves. The baseline model has a smooth decrease in loss over the epochs, whilst the ResNet's loss is *zig-zagged.* This could arrive from the MLP being stuck in a local optimum, whilst the more feature rich **ResNet** with **momentum** in it's optimizer descends from these local minimum to find optimal weights.

Overall The results highlight the power of transfer learning of using pretrained models. Aswell as how deep networks can achieve much higher classification accuracy and lower loss compared to a simple MLP.

5 Conclusion and Future Work

5.1 Conclusion

This study evaluated three neural network models: a Multilayer Perceptron (MLP), a Custom Convolutional Neural Network (CNN), and a ResNet50 model for image classification. The ResNet50 model achieved the highest validation accuracy of **92.5%** and the lowest loss of **0.1839**, significantly outperforming the Custom CNN (**86.0%** accuracy, **0.3084** loss) and the MLP (**51.9%** accuracy, **0.8705** loss). These results highlight the effectiveness of *deep convolutional networks* and transfer learning in complex classification tasks (*which some human's may even struggle with*). Additionally, the ensemble approach, which involved stacking the Custom CNN with ResNet50, was not used any further due to poor computational time and challenges in model loading, showing the trade-off between performance gains and usability.

5.2 Future Work

Future research could focus on improving the ensemble approach to balance computational efficiency with performance improvements. Alternatively with greater computational power, it could be worth stacking even more models to make the increase in accuracy outweigh the decrease in performance. This could involve exploring more efficient ensemble techniques or reducing model complexities to mitigate the issues encountered with stacking multiple models. Additionally it would be worth enhancing my skills in data exploration and data preprocessing of Image datasets, something that was a challenge in this project. Experimenting with other pretrained models apart from ResNet50, such as EfficientNet or DenseNet, might produce even better classification accuracies. Lastly, deploying the best-performing models in real-world applications and conducting testing across diverse datasets would validate their usefulness and adaptability to different image classification scenarios.

References

[1] Siddesh Bangar, *VGG-Net Architecture Explained*, Jun 29, 2022. Available: `https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f`

[2] Clean Vision *CleanVision Documentation*, 2024. Available: `https://cleanvision.readthedocs.io/en/latest/`

[3] PyTorch Normalization Values *PyTorch Documentation*, Available: `https://pytorch.org/vision/stable/models.html`

[4] PyTorch, CrossEntropyLoss, *PyTorch CrossEntropyLoss*, Available:`https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`

[5] An overview of gradient descent optimization algorithms, *Sebastian Ruder*, Available:`https://arxiv.org/pdf/1609.04747`

[6] PyTorch SGD, *PyTorch Documentation*, Available:`https://pytorch.org/docs/stable/generated/torch.optim.SGD.html`

[7] PyTorch Adam, *PyTorch Documentation*, Available:`https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`

[8] Meet Patel, *Understanding the Rectified Linear Unit (ReLU)*. Available:`https://medium.com/@meetkp/understanding-the-rectified-linear-unit-relu-a-key-activation-function-in-neural-network`

[9] Fruit 360 Dataset, *Kaggle, Version: 2024.08.04.0*, Available:`https://www.kaggle.com/datasets/moltean/fruits`

[10] David Fagbuyiro, *Guide To Transfer Learning in Deep Learning*, Apr 21, 2024, Available:`https://medium.com/@davidfagb/guide-to-transfer-learning-in-deep-learning-1f685db1fc94`