COMP309/AIML421 — ML Tools and Techniques Project

Student ID: 300569298

Student Name: Shiyan Wei

1. Introduction

This report presents the project aimed at constructing an image classification model. The primary objective is to differentiate between images of tomatoes, cherries, and strawberries by leveraging the power of Convolutional Neural Networks (CNNs). This project is implemented in Python and utilizes various packages, with PyTorch being a central component.

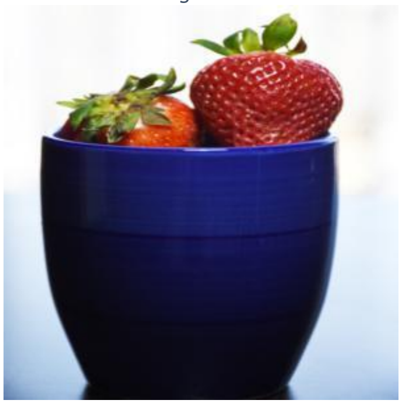
2. Problem investigation

- There are 4,500 images in the dataset, with 1,500 images belonging to each of the classes (tomato, cherry, and strawberry). The dataset is balanced, and there are no concerns about class imbalance.

- While the images have been initially processed to a size of 300 x 300 pixels, during the exploratory data analysis phase, I discovered that some images are still smaller than this size. This discrepancy could potentially cause issues during model training. Therefore, I am considering resizing all images again to ensure uniformity.

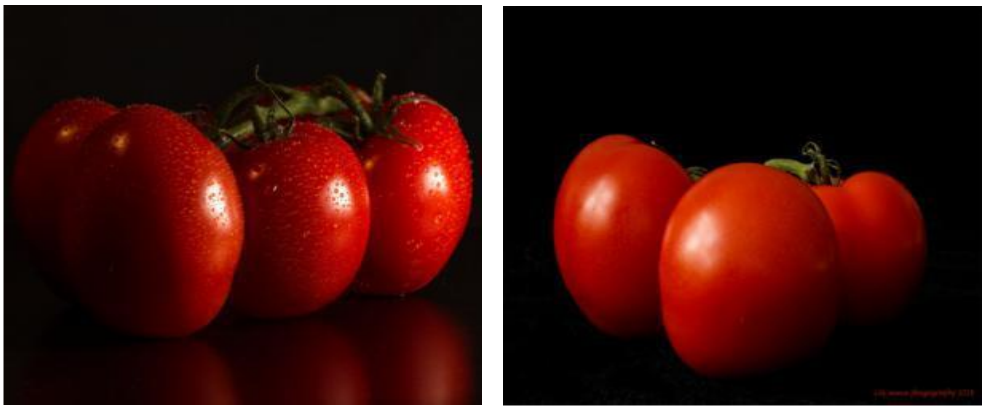
- Upon a more detailed examination of the dataset, it has come to my attention that certain target images are obstructed by various objects, as depicted in Figure 1. This obstruction often obscures the complete shape and structure of the objects, making them challenging to identify.

Figure 1



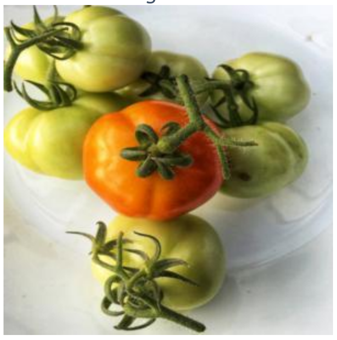
- Additionally, it's crucial to acknowledge that certain images display variations in viewpoint, as exemplified in Figure 2. Although these images may represent the same object category, they do so from distinct angles and perspectives, introducing complexities in accurate identification. To address this, I intend to employ image augmentation techniques to enrich the dataset with additional variations.

Figure 2



- Furthermore, I have noticed that the colours variations in the images can have an impact on image identification. Tomato, cherry, and strawberry have different colours and different sizes in real life. In Figure 3, we can see the green and orange tomatoes. To address this issue, I intend to normalize the colours of the images during preprocessing.

Figure 3



3. Data-processing

3.1. Image Resizing
To reduce computational complexity during model training, we have resized all images from their original dimensions of 300 x 300 pixels to a more manageable 64 x 64 pixels resolution. This resizing not only improves model training efficiency but also reduces the overall training time.

3.2. Image Augmentation
We only have 4,500 images available for training the model and no additional test data, we needed to partition the given dataset into two subsets. This division has led to a reduction in the amount of data available for analysis in the training set. To compensate for this limitation and to enrich the diversity of our training dataset, we've employed image augmentation techniques on the original images. For examples, flipping randomly the images horizontally, which helps the model generalize better by seeing variations of the same object, additionally, we apply random rotations of up to 10 degrees to the images, these techniques can generate additional training samples to enriching the dataset and making the model robust to different orientations.

3.3. Image Normalize
To standardize the size and distribution of images, we have applied image normalization to the dataset. This crucial step enhances both the stability and performance of the model. We have opted for a widely accepted approach by using common mean and standard deviation values as parameters for normalizing the RGB channels of each loaded image. (Cook, 2021)

4. Methodology

4.1. How you use the given images
To utilize the provided images effectively, given the absence of a validation set, we have performed a data split. The available image data has been divided into two subsets: a training set and a test set. Specifically, the training data includes 90% of the dataset, while the remaining 10% is the test data. Instead of relying on scikit-learn, we chose to employ PyTorch for data set splitting. This decision was motivated by the transformation of the image data set into numpy arrays, rendering PyTorch more suitable for this task. Additionally, PyTorch offers a richer set of image augmentation tools and models tailored for image classification tasks. Furthermore, this partitioning strategy allows us to have a small but essential test dataset for evaluating the model's performance, while also ensuring that we provide a more diverse and comprehensive set of image features to the model during the training process.

4.2. The loss functions.

The loss function is a measure of the difference between the model's predicted value and the true label. Using a loss function can help the model is to minimize this difference. The choice of using 'nn.CrossEntropyLoss()' as the loss function is suitable for a multi-class classification problem like image classification as we would like to identify images from three categories(tomato, cherry and strawberry). Additionally, this function is a combination of softmax and cross-entropy loss. Softmax is a function that produces output in a neural network like CNNs, so that can be compatible with the CNNs model we will use as our final model, which can provide a stable and effective procedure for dealing with issues. (Cloud, 2023)

4.3. The optimization method

For optimization strategies, I conducted an extensive evaluation of various optimizers to enhance the model's performance. I experimented with the SGD optimizer, because I used it in the baseline mlp model and it returns a good performance in the model, however, it is not suitable for CNNs, it turns loss increasing and accuracy reducing while I trained my model, it did not return satisfactory results in terms of model performance. Finally, I employ the Adam optimizer as the final optimization method. It gives me better performance than SGD and achieves lower loss and higher accuracy throughout the training process. Additionally, I added a learning rate scheduler to the model's architecture. This scheduler dynamically adapts the learning rate within the optimizer at regular intervals, precisely every 3 epochs. This adaptive learning rate scheduling significantly contributes to improving model convergence and overall training performance.

4.4. Regularization Strategies, Activation Functions, and the Benefits of Pre-trained Models.

Typically, we employ regularization techniques and activation functions to mitigate or prevent model overfitting. Various methods are available for achieving this objective. In my initial CNN model, I constructed multiple layers in the model, with each convolution layer followed by batch normalization and ReLu activation functions to implement regularization. I also employed pooling and dropout layers to reduce the spatial dimensions of the feature maps, and used the final fully connected layers for image identification. However, the initial model didn't perform well on the test set, achieving only 47% accuracy. To improve its performance, I added more convolution layers to fine-tune the model, which increased accuracy to 58%. Nevertheless, further improvements seemed to plateau, and training time also increased significantly.
In my final CNN model, I made a different choice. Instead of adding custom layers, I opted for transfer learning using the ResNet 18 model. ResNet 18 is a pre-trained model with 17 convolution layers and 1 fully connected layer, trained on a vast dataset, making it proficient in feature extraction. I combined ResNet 18 for feature extraction with a custom fully connected layer in my model.
This approach significantly boosted the test accuracy, exceeding 80%. While it did require a longer training time, I found it well worth the investment to achieve this level of accuracy, especially when compared to the earlier results.

4.5. Hyper-parameter settings

Regarding the design of the hyperparameters, I made two significant changes. First, I adjusted the learning rate of the optimizer, reducing it from 0.001 to 0.0001. This modification was implemented to ensure more stable and precise weight updates during training. Second, I reduced the batch size from 64 to 32. This change aimed to strike a balance between memory efficiency and model performance, as well as to improve the model's convergence speed. These hyperparameter adjustments collectively contributed to better training dynamics and ultimately led to improved model performance.

4.6. Get more images

I downloaded 200 images for each class from Flickr to enrich the training set and resized the image for 300x300 to match the given data.

4.7. The use of existing models.
I build the final CNN model with ResNet18, a pre-trained model from ImageNet. While this approach doubled the training time compared to my original model, it also doubled the accuracy and improved overall performance by nearly 40%. Given these significant improvements, I believe it was a worthwhile decision to replace my original model with this pretrained one.

5. Summary and discussion

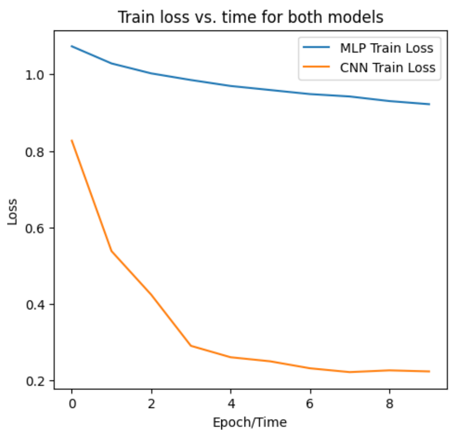
5.1. Describe the structure and overall settings of your chosen baseline MLP, and your best CNN.

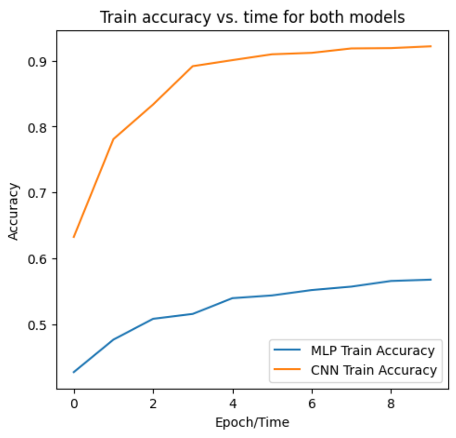
The baseline MLP model is structured with two fully connected layers. It takes three essential arguments: input_size, which specifies the dimensions of the input (e.g., 64x64x3 for a 3-channel RGB image); hidden_size, determining the size of the hidden layer; and num_classes, which defines the number of output classes. The model employs the Rectified Linear Unit (ReLU) activation function after the hidden layer to introduce non-linearity. It's important to note that this basic model doesn't incorporate dropout layers for regularization, which is often added in more complex models to prevent overfitting. Also, it apply loss function with 'nn.CrossEntropyLoss()' and optimizer with Stochastic Gradient Descent (SGD) and a learning rate of 0.001.

In contrast to the MLP model, the CNN model has a relatively simpler structure. The CNN model leverages the pre-trained ResNet-18 model for feature extraction and adds custom pooling and output layers for image classification. Additionally, the optimizer is switched to 'Adam' from SGD, and the learning rate is adjusted to 0.0001 to ensure a more stable model. These modifications result in an improved performance of the CNN model, achieving better results while maintaining the same loss function, 'nn.CrossEntropyLoss.' This series of enhancements not only boosts performance but also simplifies the model structure, making it more comprehensible and maintainable.

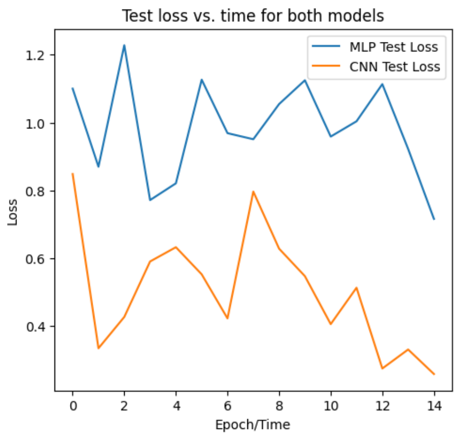
5.2. Compare the results of your MLP and CNN, in terms of the training time and the classification performance (for validation/test data). Plots of loss and accuracy versus time, for train and test sets, would be useful graphics to show.

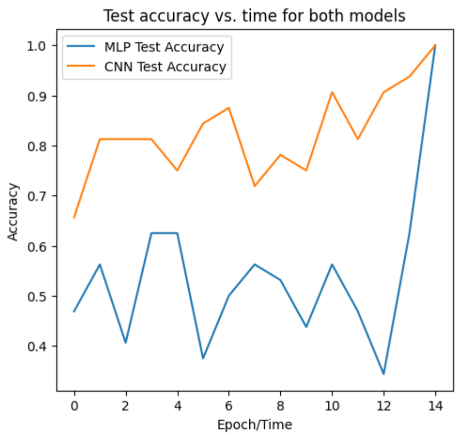
- The observation that the training loss of the Convolutional Neural Network (CNN) is significantly lower and the training accuracy is substantially higher than that of the Multilayer Perceptron (MLP) indicates that the CNN is a better fit for the model. This suggests that the CNN has learned the training data more effectively and has a better ability to capture complex patterns and features in the data.





- While we assess both the MLP and CNN on validation datasets, we can observe that the Multilayer Perceptron (MLP) having higher loss and lower accuracy than the Convolutional Neural Network (CNN) on the validation dataset is significant. It suggests that the CNN initially outperforms the MLP in terms of validation performance. However, I noticed that the MLP's accuracy approaches that of the CNN over time, which can be intriguing and requires further analysis.





- In summary, it is evident that the Convolutional Neural Network (CNN) consistently outperforms the Multilayer Perceptron (MLP) on both the training and test datasets.

5.3. Note any striking differences, and discuss why you think they occurred

- **Striking Difference**: the CNN consistently outperformed the MLP on both the training and test sets, displaying lower loss and higher accuracy.
  Explanation: this difference might be attributed to fundamental architectural distinctions between MLP and CNN. CNNs are specifically designed for tasks like image classification, leveraging convolutional layers to extract spatial features, while MLPs lack this capability. As a result, CNNs excel in capturing intricate patterns and structures in the image data, leading to their greater performance.

6. Conclusions and future work

In conclusion, this project effectively harnessed Convolutional Neural Networks (CNNs) to tackle image classification. Throughout the project, several key components led to improvements in performance. Notably, data preprocessing techniques, such as image resizing, augmentation, and normalization, contributed to preparing the data effectively. Transfer learning, using a pre-trained ResNet-18 model, further boosted the model's capabilities by extracting powerful features from the images. Optimizing the model's architecture, including switching to the Adam optimizer and fine-tuning the learning rate, played a pivotal role in enhancing the model's performance and training efficiency.
To take this project further, potential areas of interest include adjusting hyperparameters to optimize not only model performance but also training time. This could involve fine-tuning learning rates, batch sizes, and regularization techniques, as well as experimenting with different optimization algorithms to strike the right balance between speed and accuracy.

References

Cloud, S. (2023 , 6 7). *PyTorch Equivalence for softmaxcrossentropywithlogits*. Retrieved from saturncloud: https://saturncloud.io/blog/pytorch-equivalence-for-softmaxcrossentropywithlogits/#:~:text=In%20simple%20words%2C%20it%20is,predicted%20and%20actual%20probability%20distributions

Cook, B. (2021, 10 21). *Normalizing Images in PyTorch*. Retrieved from sparrow: https://sparrow.dev/pytorch-normalize/