Elliott Rose
300540768

COMP 309 FINAL PROJECT – IMAGE CLASSIFICATION

Intro:

For humans, distinguishing between objects is effortless; we quickly identify their colour, material, and shape through experience—years of learning what happy expressions look like or recognizing a square's shape. In contrast, computers lack this natural ability and struggle with such tasks. This project aims to simulate that "experience," enabling computers to differentiate between images of three fruits: Cherry, Strawberry, and Tomato. This will be achieved by developing a Convolutional Neural Network (CNN).

Problem investigation:

To effectively manipulate the data, I first needed to familiarize myself with it through several exploratory data analysis (EDA) steps outlined below.

1. ***Image Dimensionality Check.***
   The initial step involves checking the dimensions of the images in the dataset. Inconsistent image sizes can negatively impact model accuracy and complicate training. I assessed all the unique widths and heights, and my findings are shown below.

(168, 300, 3), (193, 262, 3), (300, 300, 3), (169, 299, 3), (167, 301, 3), (229, 220, 3), (182, 277, 3), (199, 254, 3), (225, 225, 3), (203, 249, 3), (232, 217, 3), (194, 259, 3), (230, 219, 3), (163, 310, 3), (211, 238, 3), (236, 214, 3), (193, 261, 3), (168, 301, 3), (199, 253, 3), (197, 256, 3), (283, 178, 3), (900, 870, 3), (278, 181, 3), (183, 275, 3), (228, 221, 3), (276, 183, 3), (251, 201, 3), (234, 215, 3), (277, 182, 3), (279, 181, 3), (138, 366, 3), (957, 800, 3)

2. ***Colour channel.***
   Some of the images in the data had a colour channel value of 1, which means that it is a greyscale image. These images are not ideal for the training of the model as some images contain various colours of the fruits (for example some green strawberries).
3. ***Alpha level.***
   Some images were in RGBA format, which could adversely affect training if not all images are uniform in format.
4. ***Noisy/Useless images.***
   In the data given, there are some images that do not contain any of the 3 target classes and are therefore useless in helping training of the model.
5. ***Removing incorrectly sized images.***
   In this step, the mode image dimension of 300x300 pixels was identified as the standard for image removal with a count of 8864. Any images not matching this dimension were moved to an "unwanted" folder. It was deemed more effective to remove incorrectly sized images rather than resize them, as most of these images were found to be unsuitable for the model, thus addressing the issue of irrelevant images simultaneously.

6. ***Removing non RGB images.***
   I wrote a function, process_and_augment_images(train_data_dir),
   that processes each image in the dataset to check its mode. If the mode is RGB, the original image is returned. If the mode is "L" or "RGBA," the image is converted to RGB before being returned.
7. ***Data Augmentation.***
   Basic data augmentation was performed using simple techniques: rotation, flipping, and colour variation. For each image, one of these augmentations was randomly selected and applied, with the resulting image saved as a new file in its corresponding directory. For example, 'cherry_0001.jpg' would turn into 'cherry_0001_aug.jpg' and be saved back into the cherry subdirectory, creating a duplicate of the original with one of three augmentations (as mentioned previously).

Methodology:

In this section, I will outline the methodology employed for training the Convolutional Neural Network (CNN). This includes a detailed description and justification of various steps taken throughout the process. Key aspects covered will include train/test split, basic MLP model, CNN model, and tuning the CNN model.

Train/test split.

The initial step in my process involved splitting the data into training and testing samples. This was achieved using DataLoaders to load the data in batches.

First, I generated a list of indices for all images in the dataset. Then, I utilized the "random_split" function to randomly divide these indices into two groups—one for training and one for testing—based on specified proportions. By creating separate lists for "train_indices" and "test_indices," I ensured the division was random and unbiased. These indices act as pointers to the actual data, enabling the creation of data loaders, "train_loader" and "test_loader," for accessing the corresponding images and labels during model training and evaluation. This random splitting process was crucial for the model's learning, allowing it to assess performance on unseen data while training on a distinct portion of the dataset.

Basic MLP model.

The next step involved building a basic Multi-Layer Perceptron (MLP) model to serve as a baseline. With no convolutional layers, the MLP's accuracy was relatively low (~31.91%), yet it provided valuable insights into the workings of the more complex CNN. This MLP was built in PyTorch, featuring basic configurations and a feedforward function.

CNN model.

Convolutional Neural Networks (CNNs) excel in image analysis by using filters to capture image dependencies, making them highly effective for this task. Their advantages over traditional networks, like MLPs, include fewer parameters and reusable weights, allowing for a deeper understanding of image content. The CNN model I created shares some configurations with the MLP but includes additional adjustments: it consists of two convolutional layers, where the first takes three inputs (RGB channels) and produces 16 feature maps, and the second generates 32 outputs. Additionally, a pooling layer down-samples the feature maps, retaining essential information while reducing dimensionality.

Elliott Rose
300540768

Tuning CNN.

Tuning the CNN is essential for improving model accuracy. In this section, I will document my experiments with various tuning techniques and their impact on performance.

1. ***Hyperparameter training.***
    a. ***Learning rate.***
    The learning rate determines how quickly the model learns. A lower rate requires more epochs and risks plateauing, while a higher rate may converge faster but can lead to instability. I tested low, medium, and high learning rate values to find the optimal balance for my model.
    b. ***Number of Epochs.***
    The number of epochs defines how often the model passes through the neural network. While more epochs can improve learning, too many may lead to overfitting and slow down processing. For my model, I experimented with various epoch counts to find the optimal balance.
    c. *Number of hidden layers.*
    The number of hidden layers affects the model's accuracy, as additional layers enable it to capture more complex patterns. However, too many layers risk overfitting, so I capped the number of hidden layers at 10.
2. ***Optimizers:***
    Optimizers influence model accuracy by affecting convergence speed. While SGD converges more slowly, it often introduces beneficial randomness, leading to better performance in my case. In contrast, ADAM typically converges faster and adjusts learning rates optimally, making it less sensitive to initial hyperparameter values. Despite these advantages, SGD yielded superior results for my model.
3. ***Loss functions:***
    After exploring various loss functions for multiclass classification, I found Cross Entropy Loss to be the most suitable for this task. While Weighted Cross Entropy and Focal Loss may work well for class-imbalanced datasets, Cross Entropy Loss proved to be the best fit for my needs, so I proceeded with it exclusively.
4. ***Adding more data:***
    Adding more data gives the model a broader base to learn from, theoretically improving accuracy. To achieve this, I augmented the existing data (as detailed in the data augmentation step), effectively doubling the dataset for training.

Summary:

This study employed two neural network models: a baseline Multi-Layer Perceptron (MLP) and a tuned Convolutional Neural Network (CNN). The MLP consisted of a two-layer structure, with five hidden units in the first layer using ReLU activation and dropout for regularization, followed by an output layer aligned with the number of classes. In contrast, the CNN was tailored for image data, featuring two initial convolutional layers with ReLU activations and max pooling, then moving into fully connected layers with dropout. This architecture was chosen intentionally: the MLP served as a comparative baseline, while the CNN was optimized for image recognition, leveraging convolutional and pooling layers to capture spatial features. These configurations, combined with careful initialization and regularization, impacted each model's performance.

Elliott Rose
300540768

In terms of performance, the MLP trained much faster but showed significantly lower accuracy on the test data, as it lacked the convolutional layers integral to the CNN's image-processing capability.

Conclusion and Further Work:

Overall, my model achieved relatively low accuracy because I only implemented basic machine learning techniques for training and tuning, primarily due to time constraints. However, it still outperformed the baseline MLP and was better than random guessing, representing a notable improvement. While my model doesn't match the capabilities of the human brain, it does outperform a simple guess.

For future work, I recommend optimizing the algorithm further to reduce accuracy loss while enhancing performance. This could be accomplished through improved preprocessing steps and additional model tuning. For instance, incorporating early stopping during training would allow the model to cease training automatically once it reaches its lowest loss, saving both time and computing resources.