COMP 309 Project

Manuela Spies 300491313

Introduction

Please make sure to have installed the library *split-folders*, which can be installed via “pip install split-folders.”

For the COMP 309 final project, the task was to create a convolutional neural network (CNN) to categorize images into three categories: strawberry, tomato, and cherry. The dataset provided originally contained 1500 images for each category (4500 total). Following cleaning up the provided data, I created a baseline MLP model. I measured its loss and accuracy performance to determine how to modify the baseline to achieve better results and create a good CNN model.

Data Clean-Up and EDA

The data provided for this project contains several irrelevant or wrongly classified images (e.g., the object is neither cherry, strawberry or tomato, or multi-class). Though each class is balanced at 1500 images, the noise has to be removed. I kept my removal choices very broad (two or more known categories, not the class at all), but given that one human did this, I probably missed some. Examples of noise include the following:



There are also images with wrong dimensions. These have been resized or removed, depending on whether they are also noise or not. I did not delete any grey-scale images, as they may help recognize the shape (rather than just color). I then also deleted all images that were not perfect squares and scaled all up to 300x300.

This resulted in 1478 tomato images, 1474 strawberry images, and 1472 cherry images. To keep things equal across the different classes, I removed the least quality images from tomato and strawberry to get down to 1472.

Preprocessing

1) Remove the noise (see EDA)[1]
2) Test-Train Split: I used the Python library split-folders to split the data into test and train, with a 0.7 to 0.3 train-test split. I can manually change this through parameters later if I notice under or overfitting.
3) Image manipulation (Training only)
    a) Potential resizing. Each image is a perfect square due to the removal of noise and thus also non-square images. Via parameters, I can resize the size of the images to control how many pixels the model needs to calculate (fewer pixels means faster processing, but lower quality and vice versa).
    b) Add random rotation & random flips. This allows more data to work with, and the algorithm works with more angles and enriches the dataset. For this, I decided to add a random transformation between -90 and 90 degrees for each image
2) Transform to tensor (via ToTensor())

Baseline Model

The baseline model of my MLP uses the following parameters, chosen at random on what "feels right".

- 300x300 Images
- 80 Epochs
- 16 Batches
- 0.0001 learning rate
- 0.9 momentum
- 3 kernels

```
ass MLP(nn.Module):

  def __init__(self):
      super().__init__()

      CHANNEL_OUT_1 = 32
      CHANNEL_OUT_2 = 64
      N_CLASSES = 3
      N_CHANNELS = 3

      self.fc1 = nn.Linear(270000, CHANNEL_OUT_2)
      self.fc2 = nn.Linear(CHANNEL_OUT_2, CHANNEL_OUT_1)
      self.fc3 = nn.Linear(CHANNEL_OUT_1, N_CLASSES)
```

It has the following accuracy percentage on my test set. 52% for the total, while the classes go as follows: 45.6% for cherry, 50.1% for strawberry, 61.3% for tomato. This is a good baseline to improve on. I then developed a CNN based on the MLP, which provided the following results using the same parameters. The total accuracy starts at 58%. Cherry is 41%, strawberry is 62%, and tomato is 73%. From here, I'll attempt to improve the model through various means described in the methodology.

```
class CNN1(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(N_CHANNELS, CHANNEL_OUT_1, KERNEL_SIZE)
        self.conv2 = nn.Conv2d(CHANNEL_OUT_1, CHANNEL_OUT_2, KERNEL_SIZE)
        self.pool  = nn.MaxPool2d(2,2)

        self.fc1 = nn.Linear(341056, CHANNEL_OUT_2)
        self.fc2 = nn.Linear(CHANNEL_OUT_2, CHANNEL_OUT_1)
        self.fc3 = nn.Linear(CHANNEL_OUT_1, N_CLASSES)


    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        # x = nn.Softmax(x) # do it essentially propability between 0 and
        return x
```

[1] Looking back, it probably would have been better to do the split first, and then remove noise. However, by the time I realised that, it was not feasible to go back and redo the entire assignment with a "new" data set.

Methodology

Hyper-Parameter Settings

Due to the lengthy duration of the model at 80 epochs and my computer's capabilities, I couldn't increase that number. However, I decided to tune the other parameters (momentum, learning_rate, image_size) and choose the respective best. For each parameter (changed in order from the baseline), I chose the best, which has been bolded in the table below.

Parameter	**Baseline / Default**	**Other Attempts (Respective Total Accuracy)**
Learning Rate	0.0001 (54%)	**0.001** (65%), 0.00001 (42%), 0.0005 (62%)
Momentum	**0.9 (62%)**	1.0 (33%), 0.5 (55%), 0.1 (46%), 0.01 (51%)
betas	(0.9, 0.999) (71%)	(0.9, 0.1) (33%)
eps,	**1e-08 (74%)**	1e-07 (70%), 1e-09 (72%), 5e-08 (71%), 5e-09 (70%), 5e-07 (69%), 1e-10 (72%)
weight_decay	**0 (71%)**	1 (33%), 0.5 (33%)

I spent a lot less time on the betas and weight_decay, because it became very quickly clear to me that the default parameters did just well.

Loss Function

The loss function I ultimately decided to use is the Cross Entropy Loss Error. I tried to look into various types of classification loss functions, but most loss functions that I could find were geared towards regression models, not classification models.

Optimisation Method

I also tried various optimisation methods to find the best-scoring method. RMSprop scored 33%. Adagrad scored 63%. Adam scored 74%. SGD (which I used in the baseline) scored 65%. Obviously, given the it scored highest, I chose Adam. This did mean I didn't need momentum anymore, but did have other parameters that I'd then optimise. Please refer back to Hyper Parameter Settings for that.

Regularisation Strategy

I did not complete this.

Obtaining More Images & Use of Existing Models

I did not complete this.

Other Methodologies Used

I did not do this.

Result Discussion

From the baseline, I improved almost 20% on my own test set, from 52% to 70%. This is because of the more detailed, and thus more capable network. This comes at the cost of time; MLP takes circa 721 seconds for both training and testing, while my CNN takes 915 seconds; much longer.
Please refer to my baseline model and default/baseline parameters for my MLP model.

Conclusions and Future Work

All in all, I found this assignment very fascinating. My approach was very much a “get something working, then improve on it”, rather than try to build something from the ground up, and I believe that is why I wasn’t able to achieve all I wanted to do. That, in combination with time limitations, meant that I couldn’t look into regularisation (which I find very interesting), or using more data. The model itself is still very simple; that is a pro, because it’s much easier adjustable and acts more as a baseline for future projects, but negatively, still isn’t very advanced and still very basic. In the future, I’d like to see how the model performs on an entirely different set of images (e.g. rather than three categories of food, how about different styles of images (digital art, photography, paintings, for example!). I’m curious to see what differences that would lead to.