Module 2: Customize a Pre-trained Model for CV Classification

Trang Tran

CPS, Northeastern University

EAI 6010 | Applications of Artificial Intelligence

Dr. Dustin Garvey

June 3, 2024

Introduction

Computer vision image classification is a machine learning technique that has gained significant popularity in recent years and serves a wide array of applications. In the context of deep learning, transfer learning involves taking a model trained on one task and customizing it for another. Unlike building a model from scratch, transfer learning involves pulling an existing model off the shelf and tuning it for a specific use case. This assignment will explore the use of a pre-trained computer vision model from PyTorch for transfer learning on a new image dataset, demonstrating how leveraging pre-existing knowledge can enhance performance and efficiency in new tasks.

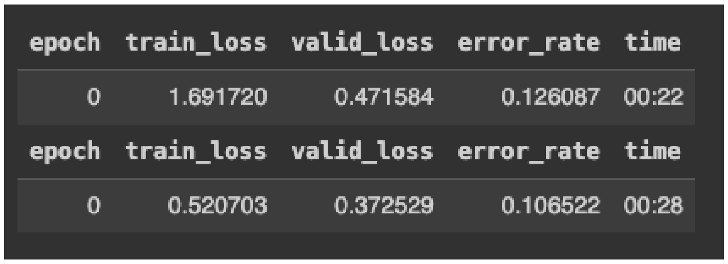
Data Selection and Methods

In this project, we utilized a dataset comprising 2301 fruit images sourced from Kaggle, categorized into ten classes: Apple, Orange, Avocado, Kiwi, Mango, Pineapple, Strawberries, Banana, Cherry, and Watermelon[1]. With approximately 230 images per category, the dataset ensures a balanced training resource for each classification.

PyTorch's EfficientNetV2 model was chosen for this dataset due to its reputation for balancing accuracy and computational efficiency. EfficientNetV2 often exhibits strong performance with fewer parameters and lower computational cost compared to other models, as highlighted in both the PyTorch documentation and the paper titled "EfficientNetV2: Smaller Models and Faster Training"[2][3].

To prepare the dataset for modeling, FastAi's features were harnessed to generate a data block, facilitating the normalization of the dataset view and the creation of a data loader. Subsequently, the Vision Learner (formerly known as CNN Learner) was employed to establish a base learner using the pre-trained EfficientNetV2 model. The validation metric ‘error_rate’ was chosen to monitor performance during training.

Fine-tuning the model involved two approaches: first, freezing all layers except the new output layer and conducting training epochs solely on the output layer; second, unfreezing the entire model and running the specified number of epochs. Initial results from a single epoch run of the model with the newly trained fruit dataset demonstrated promising performance and accuracy, with error rates ranging from only 10-12%, suggesting that the pre-trained model may be well-suited for this classification problem (Figure 1).



```
epoch  train_loss  valid_loss  error_rate  time
    0    1.691720    0.471584    0.126087  00:22
epoch  train_loss  valid_loss  error_rate  time
    0    0.520703    0.372529    0.106522  00:28
```

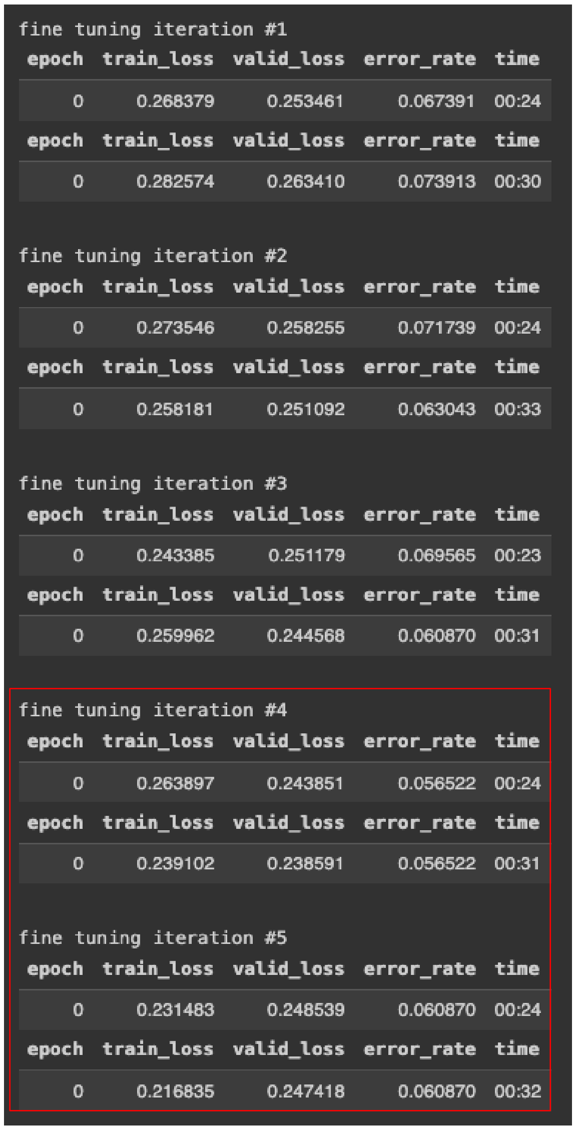
Figure 1: First single epoch trained with the dataset

Assignment Prompts

Can you create and document a scenario where over-training occurred?

Overfitting, a common challenge in neural network training, occurs when a model becomes excessively tailored to the training dataset, resulting in poor performance on unseen data such as the test or validation sets.

Despite the abundance of data, initial observations revealed signs of overtraining. This was evident from the continuous decrease in training loss, contrasted by the stagnation or increase in validation loss after approximately the 4th iteration during the fine-tuning process with the second recommended learning rate (Figure 2). Additionally, the error rate exhibited an increase. These findings indicate that the model had reached a stage where further training became counterproductive, underscoring the importance of vigilance for overfitting during model training.

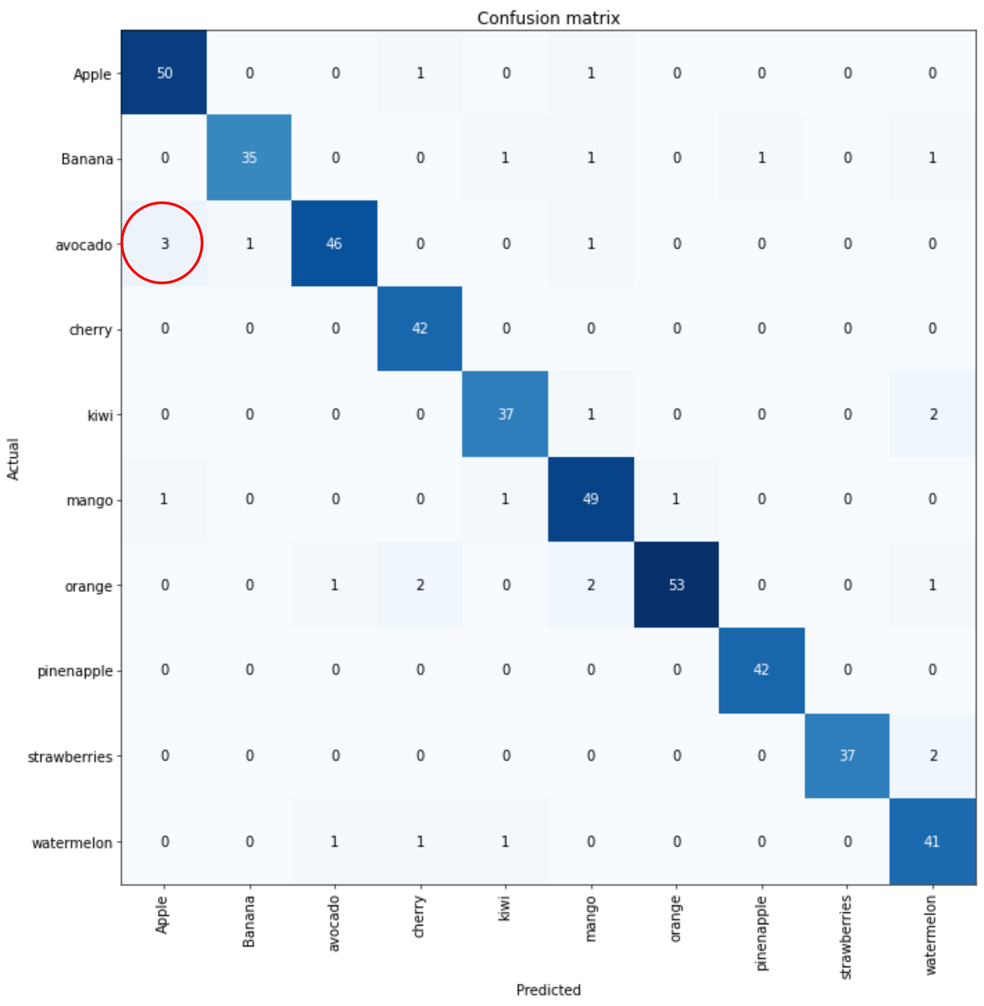


```
fine tuning iteration #1
 epoch  train_loss  valid_loss  error_rate  time
     0    0.268379    0.253461    0.067391  00:24
 epoch  train_loss  valid_loss  error_rate  time
     0    0.282574    0.263410    0.073913  00:30

fine tuning iteration #2
 epoch  train_loss  valid_loss  error_rate  time
     0    0.273546    0.258255    0.071739  00:24
 epoch  train_loss  valid_loss  error_rate  time
     0    0.258181    0.251092    0.063043  00:33

fine tuning iteration #3
 epoch  train_loss  valid_loss  error_rate  time
     0    0.243385    0.251179    0.069565  00:23
 epoch  train_loss  valid_loss  error_rate  time
     0    0.259962    0.244568    0.060870  00:31

fine tuning iteration #4
 epoch  train_loss  valid_loss  error_rate  time
     0    0.263897    0.243851    0.056522  00:24
 epoch  train_loss  valid_loss  error_rate  time
     0    0.239102    0.238591    0.056522  00:31

fine tuning iteration #5
 epoch  train_loss  valid_loss  error_rate  time
     0    0.231483    0.248539    0.060870  00:24
 epoch  train_loss  valid_loss  error_rate  time
     0    0.216835    0.247418    0.060870  00:32
```

Figure 2: Fine-tuning iterations of the second adjusted model

What training methods did you find helpful-useful to prevent overtraining and why?

An image augmentation technique was implemented in this dataset to address the issue of overtraining, and it did a good job. The confusion matrix in Figure 3 presents some insights into the validation predictions of the second/ adjusted model. The highest error is three wrong predictions between ‘avocado’ and ‘apple’.



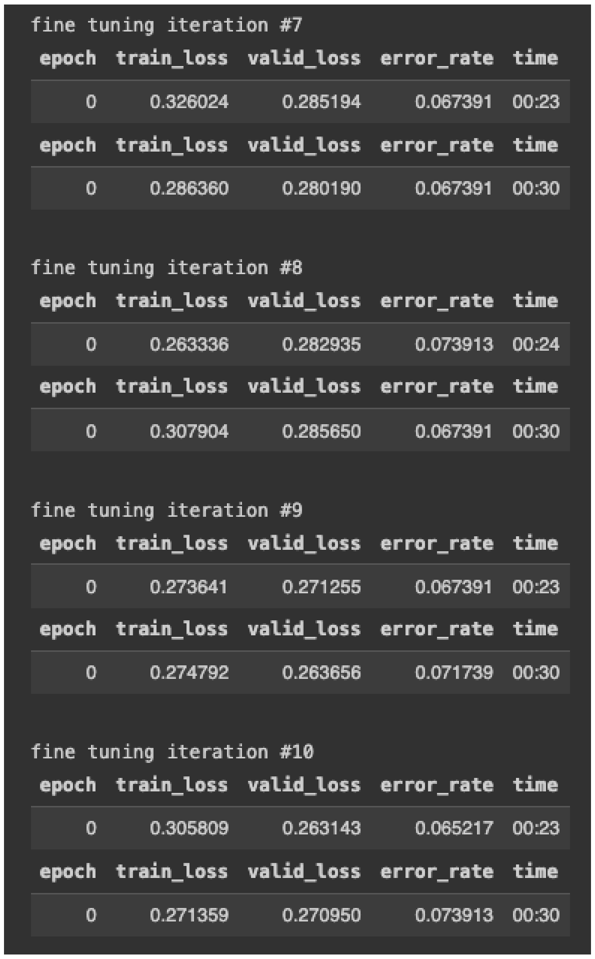
Confusion matrix

Actual \ Predicted	Apple	Banana	avocado	cherry	kiwi	mango	orange	pinenapple	strawberries	watermelon
Apple	50	0	0	1	0	1	0	0	0	0
Banana	0	35	0	0	1	1	0	1	0	1
avocado	3	1	46	0	0	1	0	0	0	0
cherry	0	0	0	42	0	0	0	0	0	0
kiwi	0	0	0	0	37	1	0	0	0	2
mango	1	0	0	0	1	49	1	0	0	0
orange	0	0	1	2	0	2	53	0	0	1
pinenapple	0	0	0	0	0	0	0	42	0	0
strawberries	0	0	0	0	0	0	0	0	37	2
watermelon	0	0	1	1	1	0	0	0	0	41

Figure 3: Confusion matrix of the second adjusted model

Did you reach a point where it was clear that you should stop training?

Compared to the initial model training, which reached a saturation point around the 8th-9th iteration during fine-tuning (Figure 4), the second adjusted model was conducted for only five single-epoch iterations and reached the saturation point earlier only after the fourth iteration (Figure 2). This adjustment obviously resulted in improved outcomes, with a time-consuming process and lower validation losses and error rates. The training loss appeared to be approaching the threshold of minimum loss (Figure 2 & 5).



```
fine tuning iteration #7
 epoch  train_loss  valid_loss  error_rate  time
     0    0.326024    0.285194    0.067391  00:23
 epoch  train_loss  valid_loss  error_rate  time
     0    0.286360    0.280190    0.067391  00:30

fine tuning iteration #8
 epoch  train_loss  valid_loss  error_rate  time
     0    0.263336    0.282935    0.073913  00:24
 epoch  train_loss  valid_loss  error_rate  time
     0    0.307904    0.285650    0.067391  00:30

fine tuning iteration #9
 epoch  train_loss  valid_loss  error_rate  time
     0    0.273641    0.271255    0.067391  00:23
 epoch  train_loss  valid_loss  error_rate  time
     0    0.274792    0.263656    0.071739  00:30

fine tuning iteration #10
 epoch  train_loss  valid_loss  error_rate  time
     0    0.305809    0.263143    0.065217  00:23
 epoch  train_loss  valid_loss  error_rate  time
     0    0.271359    0.270950    0.073913  00:30
```

Figure 4: Fine-tuning iterations of the initial model

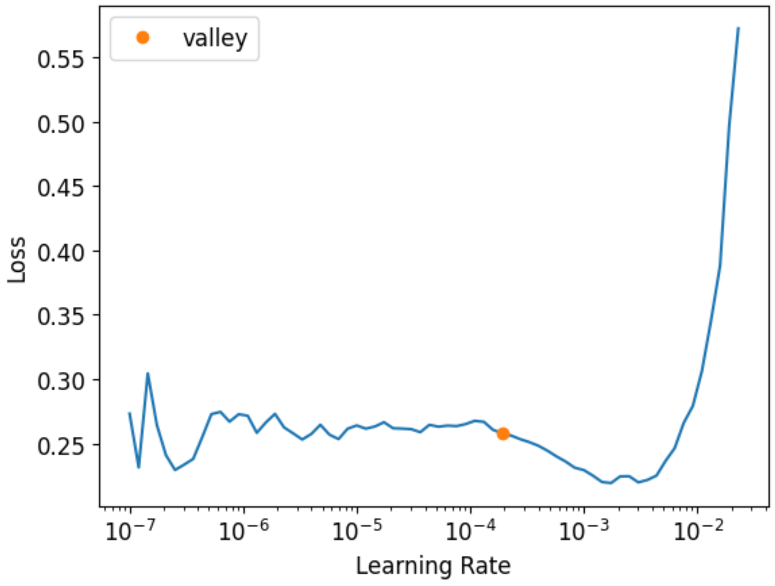
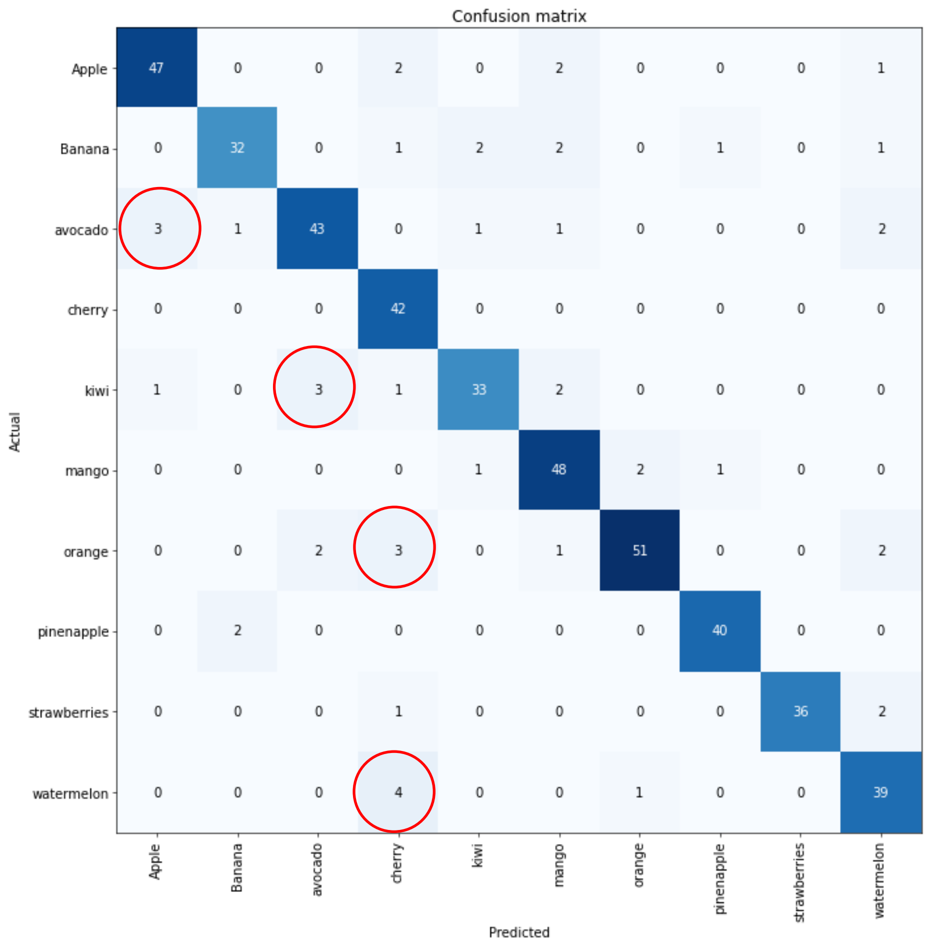


Figure 5: Learning rate and Loss for the second adjusted model

What image classes had the best-worst performance and for the worst performing classes, what is your recommended path for improvement?

Figure 6 highlights the worst performance of the validation predictions. However, it's important to note that this evaluation is based on the results extracted from a single training epoch, which may not fully capture the model's potential. To improve performance, we need to address the class imbalance split for the train-validation set, fine-tune pre-trained models with multiple iterations, and do hyperparameter tuning (adjust learning rate).



Actual \ Predicted	Apple	Banana	avocado	cherry	kiwi	mango	orange	pinenapple	strawberries	watermelon
Apple	47	0	0	2	0	2	0	0	0	1
Banana	0	32	0	1	2	2	0	1	0	1
avocado	3	1	43	0	1	1	0	0	0	2
cherry	0	0	0	42	0	0	0	0	0	0
kiwi	1	0	3	1	33	2	0	0	0	0
mango	0	0	0	0	1	48	2	1	0	0
orange	0	0	2	3	0	1	51	0	0	2
pinenapple	0	2	0	0	0	0	0	40	0	0
strawberries	0	0	0	1	0	0	0	0	36	2
watermelon	0	0	0	4	0	0	1	0	0	39

Figure 6: Confusion matrix after the 1st single epoch

References

1. Abdulnabi K. (n.d.). Fruit classification (10 Class). Kaggle. [Dataset]. Retrieved June 3, from https://www.kaggle.com/datasets/karimabdulnabi/fruit-classification10-class
2. Models and pre-trained weights. PyTorch documentation. Retrieved June 3, from https://pytorch.org/vision/stable/models.html#classification
3. Tan, M., & Le, Q. V. (2021). EfficientNetV2: Smaller Models and Faster Training. ArXiv. /abs/2104.00298
4. EAI6010.81433.202435. Module 2: Lesson 4: Choosing Between Pre-trained and Untrained Models.