BBM409 : Introduction to Machine Learning Lab.
Fall 2025
Instructor: Assoc. Prof. Dr. Hacer Yalım Keleş
TAs: Sümeyye Meryem Taşyürek* & Sevginur İnce

Assignement 3

Due date: Tuesday, 09-12-2025, 11:59 PM.

The objective of this assignment is to cultivate your comprehension and familiarity with Convolutional Neural Network (CNN) concepts and transfer learning techniques in the context of CNN for image classification tasks.

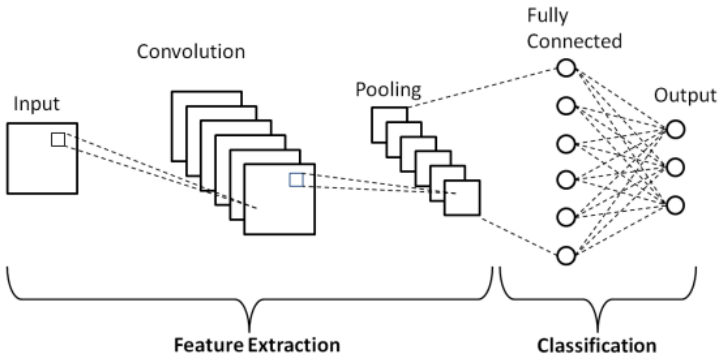
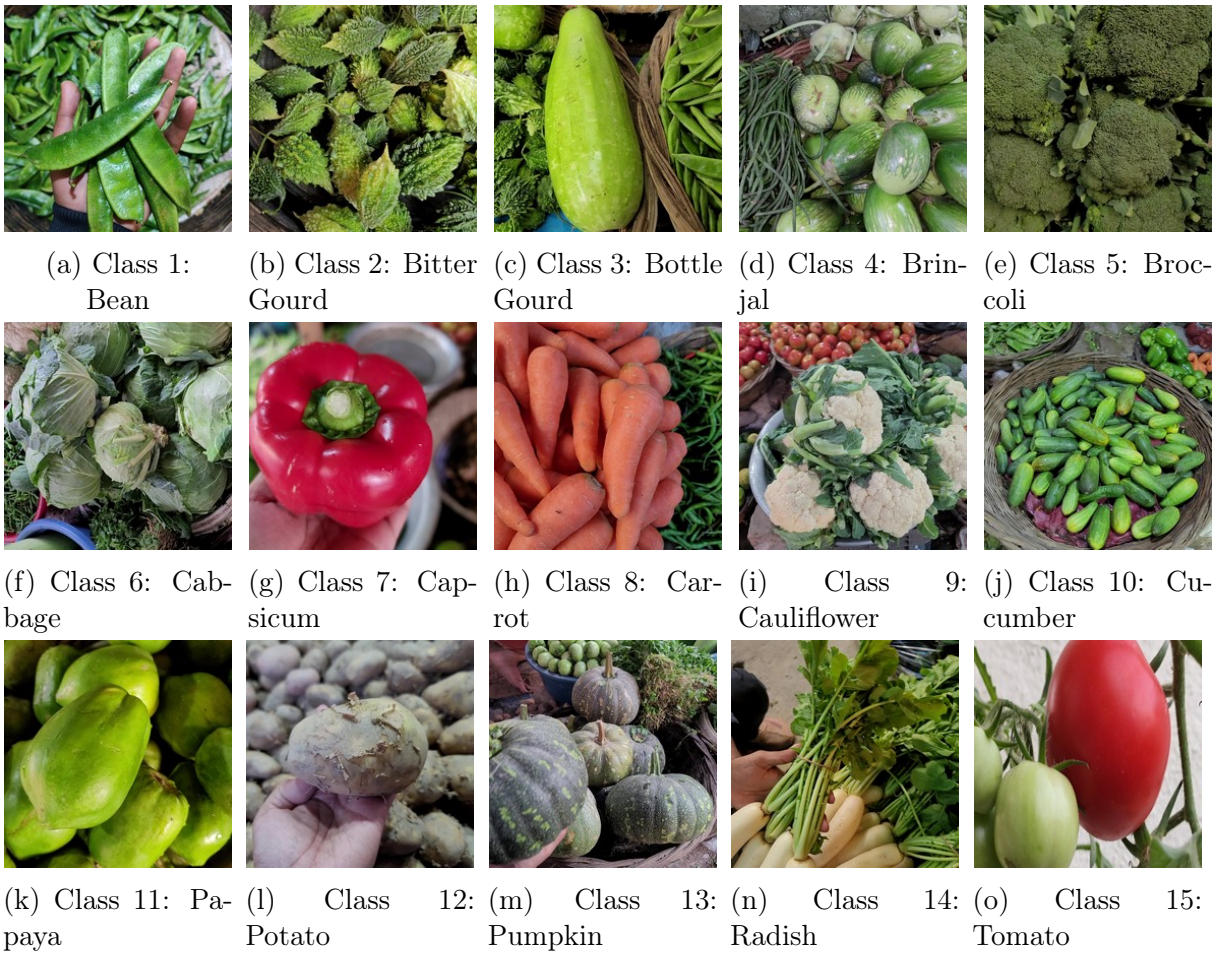


Figure 1: A simple CNN architecture

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision by enabling automated feature extraction and hierarchical learning from image data. Unlike traditional neural networks, CNNs preserve the spatial structure of images through convolutional layers, which apply filters to extract features such as edges, textures, and shapes. Given an input image represented as a matrix of pixel values, CNNs perform a series of operations, including convolution, pooling, and non-linear activation, to progressively learn hierarchical representations of the input data. These learned representations enable CNNs to effectively classify images into different categories.

1 Part 1: Implementation of a Simple CNN Model

In this section of the assignment, you will implement a basic CNN model from scratch for the task of image classification using the deep learning framework **PyTorch**.
Your objective is to construct a simple CNN architecture using only the fundamental low-level components provided by PyTorch such as convolutional layers, activation functions, and pooling layers. It is not allowed to use any high-level libraries, pretrained models, or pre-built architectures in this section; instead, you must **manually implement both the model and the entire training pipeline yourselves**, including the training and validation loops. Your model should have **5 convolutional layers and a fully connected layer**. You will then train this model on the provided dataset containing images of various vegetables.



(a) Class 1: Bean (b) Class 2: Bitter Gourd (c) Class 3: Bottle Gourd (d) Class 4: Brinjal (e) Class 5: Broccoli

(f) Class 6: Cabbage (g) Class 7: Capsicum (h) Class 8: Carrot (i) Class 9: Cauliflower (j) Class 10: Cucumber

(k) Class 11: Papaya (l) Class 12: Potato (m) Class 13: Pumpkin (n) Class 14: Radish (o) Class 15: Tomato

Figure 2: Different vegetable species sample images from different classes.

To facilitate this task, **you will use a subset of Vegetable Image Dataset** [1] containing 4500 images of different vegetable species in different folders, corresponding 15 labels. The species type for each class is stated in Figure 2. To ensure balanced and fair training, the dataset is split per class into 3000 training images (200 per class), 750 validation images (50 per class), and 750 testing images (50 per class). The same unlabeled test set is provided within the assignment files and will be shared in the Kaggle environment which will be used for submission. You should **use the validation set during development** to select your best-performing model, and then **evaluate this final model on Kaggle**, ensuring a proper and unbiased model selection process.

Steps to Follow

1. Load the vegetable image dataset using PyTorch's data loading utilities.
2. Preprocess the images by resizing them to a uniform size (256x256), normalizing pixel values, and applying any other necessary transformations using *torchvision.transforms*.
3. Design a **CNN architecture with 5 convolutional layers** followed by activation functions (e.g., ReLU).

4. Intersperse pooling layers (e.g., max pooling) to reduce spatial dimensions and extract key features.
5. Flatten the output from convolutional layers to feed into a fully connected layer.
6. Add a fully connected layer at the end to map features to the output classes.
7. Feel free to experiment with architectural choices such as the type of activation function, number of filters, filter sizes, pooling strategies, or the structure of the fully connected layers to improve performance.
8. Define appropriate loss function for multi-class classification.
9. Choose an optimizer (SGD or Adam) and set its parameters (e.g., learning rate).
10. Explain your choice of activation functions, loss functions and optimization algorithms.
11. Iterate over the training dataset in mini-batches. Implement forward pass, feed the inputs through the network to compute the predictions.
12. Compute the loss between predictions and actual labels. Implement backward pass, compute gradients of the loss with respect to model parameters.
13. Update model parameters using the optimizer based on computed gradients.
14. Validate the model on the validation set periodically to monitor performance and plot the validation loss.
15. Repeat the training process for a suitable number of epochs (at least 30 epochs).
16. You can conduct experiments with different learning rates (lr) and batch sizes to find best performing model. Keep track of performance metrics during these experiments to identify the optimal configuration. Select your best model with respect to validation accuracy. It is enough for you to visualize the accuracy and loss change of the best model across training and validation datasets. Mention about your lr and batch size selection process in your notebook/report.
17. Compute metrics such as accuracy, precision, recall, and F1-score to assess classification performance.
18. Visualize confusion matrix for your best model to understand the model's behavior across different classes.
19. Test the best model on the test set to evaluate its performance in the Kaggle environment. Report your Kaggle score in the notebook/report.
20. Explain and analyze your findings and results.
21. Summarize the performance of the model on the test set. Comment on the results.
22. Discuss any challenges encountered during training and potential areas for improvement.

2 Part 2: Transfer Learning and Comparative Analysis

In this section of the assignment, you will explore the concept of transfer learning by utilizing a pre-trained CNN model for image classification. Specifically, you will employ transfer learning techniques to fine-tune pre-trained ResNet-18 and MobileNet model on the Vegetable Image Dataset.
The objective is to compare a fine-tuned pre-trained models with the CNN model implemented from scratch. Through this comparative analysis, you will provide insights and comments on the advantages and disadvantages of both approaches, shedding light on the practical considerations when choosing between implementing a CNN from scratch versus leveraging pre-trained models.

Transfer learning is a powerful technique in deep learning, especially when working with limited labeled data. Instead of training a CNN from scratch, transfer learning leverages pre-trained models that have been trained on large-scale datasets like ImageNet, which contain millions of labeled images across thousands of categories. These pre-trained models have already learned generic features that are useful for a wide range of visual recognition tasks. In the context of this assignment, you will employ transfer learning using pre-trained architectures to classify images of different vegetable species in the Vegetable Dataset. You'll work with ResNet-18, a lightweight residual network pre-trained on ImageNet, and explore a second architecture, MobileNet, known for its efficiency on resource-constrained environments. By leveraging these models, you will develop different models using transfer learning and conduct a comparative analysis among them, alongside the CNN model you constructed from scratch. You will perform the following:

ResNet-18 Models:

- Finetune the weights of the final fully connected (FC) layer in the ResNet-18 network. (base model for ResNet-18)
- Finetune the weights of convolutional layer block 3, convolutional layer block 4, and the final FC layer in the ResNet-18 network. (second model for ResNet-18)
- Finetune all weights of the ResNet-18 network. (third model for ResNet-18)

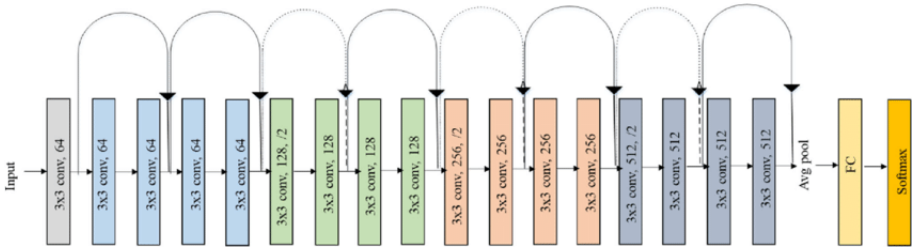


Figure 3: ResNet-18 architecture diagram [2]

MobileNet Models:

- Finetune the weights of the final FC layer in the MobileNet network. (base model for MobileNet)
- Finetune all weights of the MobileNet network. (second model for MobileNet)

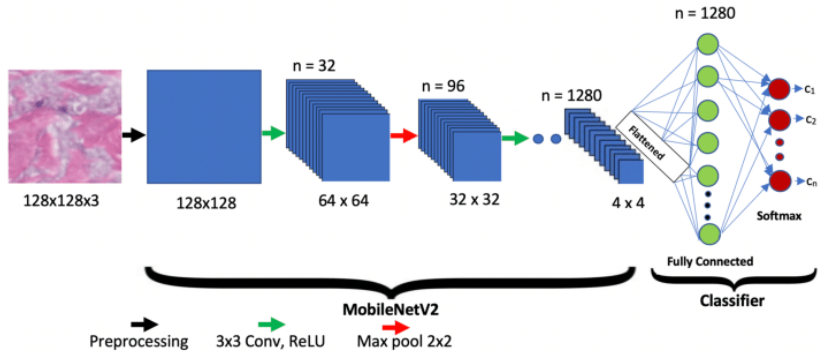


Figure 4: MobileNetV2 architecture diagram [3]

Steps to Follow

ResNet-18 Models

1. Use PyTorch's `torchvision` library to load the pre-trained ResNet-18 model that has been trained on ImageNet.
2. Ensure that the model's architecture matches the ResNet-18 architecture.
3. Replace the final fully connected (FC) layer of the ResNet-18 model with a new FC layer that matches the number of classes in the Vegetable dataset.
4. Define an appropriate loss function and optimizer for training. Set hyperparameters (e.g., learning rate) based on your previous experiments.
5. Train only the final FC layer using the Vegetable dataset while keeping the ResNet-18 weights frozen. *(base model for ResNet-18)*
6. For the second model, unfreeze convolutional layer blocks 3 and 4, and train these layers along with the final FC layer. *(second model for ResNet-18)*
7. For the third model, unfreeze all layers and train the entire ResNet-18 model. *(third model for ResNet-18)*

MobileNet Models

8. Load the pre-trained MobileNetV2 model from `torchvision` and ensure the architecture matches the standard MobileNetV2.
9. Replace the final FC layer with a new one matching the number of classes in the Vegetable dataset.
10. Train only the final FC layer while keeping MobileNet weights frozen. *(base model for MobileNet)*
11. Train the entire MobileNet network by unfreezing all layers. *(second model for MobileNet)*

Comparative Analysis and Evaluation

12. Monitor training for all models and evaluate their performance on the validation set periodically. Visualize the accuracy and loss changes for each model across training and validation datasets. Conduct a comparative analysis among the models, commenting on their performance. Select the best model based on validation accuracy.

13. Compute and show metrics such as accuracy, precision, recall, and F1-score to assess classification performance of the best model.

14. Test the best model (from ResNet-18 and MobileNet) on the test set and evaluate their performance in the Kaggle environment. Report your Kaggle score in the notebook/report.

15. Compare the performance of the these best fine-tuned models (from ResNet-18 and MobileNet) with your CNN model implemented from scratch in terms of accuracy, training efficiency, and resource usage.

16. Provide insights and comments on the advantages and disadvantages of transfer learning with pre-trained models versus implementing a CNN from scratch.

17. Discuss practical considerations when choosing between these approaches, considering factors such as dataset size, computational resources, and task complexity.

18. Summarize the findings of the comparative analysis between the fine-tuned models and the CNN model.

Note: When increasing the number of layers to be fine-tuned, it's essential to decrease the learning rate accordingly. This helps maintain stability and facilitates effective training of the deeper layers.

3 What to Hand In

Your submitted solution should include the following:

- The filled-in Jupyter Notebook as both your source code and report. The notebook should include (1) markdown cells reporting your written answers alongside any relevant figures and images and (2) well-commented code cells with reproducible results.

- Additionally, your report should be self-contained, encompassing a concise summary of the problem statement and a detailed exposition of your implemented solution.

- Begin by providing a concise overview of the problem statement, detailing the objectives and scope of the assignment. Define the tasks involved in Part 1 (CNN from scratch) and Part 2 (transfer learning).

- Describe the dataset. Include information about the features, targets, and any preprocessing steps applied to the data.

- Explain the approach adopted for each part of the assignment. Describe the algorithms implemented.

- Explain the evaluation metrics utilized to assess the performance of your models and justify their relevance to the problem at hand.
- Reflect on the outcomes of your experiments.
- Feel free to include tables or figures to emphasize specific aspects of your solution and enhance clarity for the reader.
- **You are also expected to participate in the upcoming Kaggle challenges.** You will use Kaggle's submission system to evaluate the performance of your best-performing models (ResNet-18, MobileNet and CNN from scratch). This step ensures that the models are tested on the hidden test set to reveal their true performance in an unbiased manner. The Kaggle submission system will compute and display the final scores based on your uploaded predictions. **Each model will contribute 5% of the grade, based on Kaggle test performance.** Challenge details will be announced shortly. **Stay tuned!**

You should prepare a ZIP file named <student id>.zip containing the Jupyter notebook in ipynb format as well as its .py (Python file) version containing all your codes. Submit it to submit.cs.hacettepe.edu.tr, where you will be assigned a submission. The file hierarchy is up to you as long as your Jupyter notebook and Python file works fine.

Attention! Please note that training the models could potentially require a considerable amount of time, particularly depending on the performance capabilities of your computer. To ensure that you obtain the required results in a timely manner, it is advisable to start the assignment early. If you find yourself starting the assignment later than anticipated and your computer's GPU capacity is limited, it might be beneficial to implement the assignment using Google Colab, leveraging its GPU resources for faster computation.

4 Grading

- Part1: Implementation of Simple CNN Model (%35 + %5 for Kaggle Test Results)
- Part2: Transfer Learning and Analysis (%50 + %10 for Kaggle Test Results)

Note: Preparing a good report is important as well as the correctness of your solutions! You should explain your choices and their effects to the results.

Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for online/AI sources. Make use of them responsibly, refrain from generating the code you are asked to implement. Remember that we also have have access to such tools, making it easier to detect such cases.

References

[1] Vegetable Image Dataset
[2] Deep Residual Learning for Image Recognition
[3] MobileNetV2: Inverted Residuals and Linear Bottlenecks