National University of Sciences and Technology

School of Electrical Engineering and Computer Science

Department of Computing

CS-405 Deep Learning

Fall 2023

# Lab 6

**CNN Architectures for Vegetable Classification**

# Date: 20th Oct 2023

# Name: Alina Nasir

# CMS ID: 342350

# Instructor: Dr Daud Abdullah

**Question 1: What does the torchvision.transforms.Compose() function do in the cell below?**

The torchvision.transforms.Compose() function is used to create a composed sequence of image transformations. It allows us to apply a series of image processing operations to a given image in a specific order. Each operation in the list is applied sequentially to the input image. In the code provided in the lab, there are two transformations applied in sequence, the first one toTensor() converts the input image to PyTorch tensor form and the second one Resize(), resizes the input image to the width and height defined as parameters.

**Question 2: What is the torchvision.datasets.ImageFolder() method used for in the cell below? What are its arguments?**

This method is used to create the dataset from the directory where each subdirectory represents a different class or category of images. The first argument **root** (required) specifies the root directory of the dataset. It is the top-level directory that contains all the subdirectories of the subcategories present. In the code, **data\_path** is provided as the **root** directory.

The second argument is **transforms**. This argument allows us to specify a transformation to be applied to the images as they are loaded from the dataset. It will be applied to each image loaded from the dataset, converting it into a PyTorch tensor and resizing it to the specified dimensions.

**Question 3: What is the if-elif condition used for in the cell below?**

The if-elif condition in the code is used to determine the classes or labels associated with the dataset, specifically, the train\_set dataset. It checks two potential attributes to find the classes the train\_set.classes and train\_set.dataset. This checks if the train\_set object has a classes attribute. If it does, it means that the classes or labels for the dataset are stored as an attribute of the train\_set object itself. If train\_set.classes is not present, the code checks whether the train\_set.dataset object exists and if it has a classes attribute. This is a fallback option, where it looks for the classes in a nested dataset attribute. The purpose of this code is to find and retrieve the class labels associated with the dataset.

**Question 4: Why are we calculating the mean and std of only the train\_set in the cell below? Why not calculate for the whole dataset?**

We are calculating only the mean and standard deviation of the train set to avoid overfitting of the model on the dataset, if the entire dataset is used for calculating the mean and standard deviation then it will learn all the patterns in data.

**Question 5: How are we updating the transforms of our subsets in the cell below?**

The function is designed to update the transforms applied to a PyTorch dataset, whether it's the original dataset or a nested dataset. It first checks whether the dataset object has a transform attribute, and if so, it updates the transform. If the dataset is nested within another object, like a Subset, it also checks for a nested transform attribute.

**Question 6: What transforms are we using in the cell below?**

The first transform is the original transform that we defined that included converting images to tensor and resizing them. The following are the rest of the transforms:

1. Random Horizontal Flip:

This transformation randomly flips images horizontally with a 50% probability. It creates a mirror image of the original image.

1. Random Vertical Flip:

This transformation randomly flips images vertically with a 50% probability. It creates an upside-down image compared to the original.

1. Random Rotation with 45 degrees:

This transformation randomly rotates images by 45 degrees. It introduces random rotations to the images.

1. Gaussian Blur:

This transformation applies Gaussian blur to the images with a kernel size of 5. Gaussian blur smoothens the images, which can be a form of data augmentation.

**Question 7: Why are we only applying them to the train\_set?**

The primary goal of data augmentation is to improve the model's ability to generalize to unseen data. By applying random transformations to the training data, the model learns to be invariant to these transformations and can better handle real-world variations in the data. This helps prevent overfitting. On the other hand, the validation and test data should be presented raw to the model so that is as close as possible to original image in order to evaluate the true performance of the model. Applying data augmentation to these sets would alter the data, potentially leading to incorrect evaluations of model performance.

**Question 8: Instead of fine-tuning the pre-trained model for a few epochs what would happen if we train the pre-trained model for a large number of epochs on the train\_set?**

If we train the pre-trained model on a large number of epochs the validation accuracy does not increase any further significantly

**Question 9: What do you observe in the cells given below related to Tensor board? Please discuss in the report. How do you enable the tensor board in the Colab environment like this one?**

The %load\_ext tensorboard command is used to load the tensor board into the Colab environment otherwise if it is installed on the PC, it is run through the browser. Once it is run, you use the provided address to access it through a browser on your PC.

**Question 10: How is modularity introduced in the over training of our model in this notebook? What does run\_1\_epoch() actually do in this case?**

This function introduced modularity into the code by encapsulating the logic for running one training or validation epoch. It abstracts the process of running an epoch by taking care of the following operations:

1. Setting the model in training or evaluation mode depending on the train flag.
2. Iterating through the data loader (loader) to process each batch.
3. Handling device placement (moving data and model to GPU, if available).
4. Zeroing gradients if it's a training epoch.
5. Forward pass through the model to obtain predictions.
6. Calculating the loss based on the model's predictions and ground truth labels.
7. Performing backpropagation (if it's a training epoch) and updating model parameters.
8. Tracking the number of correct predictions and total loss for the epoch.
9. Calculating accuracy and loss metrics for the entire epoch.

**Question 11: How are we logging Tensorboard values in the cell below? Are you observing the training plots during the training of your the implemented VGG16 model?**

The add\_scalar function is used to add metrics to the Tensorboard. It has 3 arguments the tag that defines the name of the metric. The scalar\_value which is the value of the metric to record and global\_step that is on which milestone to record the scalar value which we have set to epoch. The tags are defined for train loss, train accuracy, validation loss and validation accuracy with scalar\_value being the value obtained at each epoch.

**Task 1: Compare the performance and training plots of the pre-trained model with those of the model trained from scratch.**

The model trained from the scratch took more time to train because it required a larger number of epochs to fully converge. On the other hand, pre-trained model required only a few number of epochs to train on the data further increasing the number of epochs was not producing any significant change. The pre-trained model gave a test accuracy of 96.44% on the other hand the manually trained model gave a very low accuracy of 45.78%.

**Task 2: Comparison of RES-NET50 with VGG16 pre-trained:**

On the same number of epochs, the RES-NET50 gave a test accuracy of 86.44% whereas VGG16 pre-trained model gave an accuracy of 96.44%. Hence performance od VGG16 was better for this dataset