Deep Learning – Spring 2024
Assignment 3
Classification & Transfer Learning with CNNs

Due Date: 11:59 PM on Thursday, 22 March 2024 Total Marks: 200

Submission:

Submit all of your codes and result in a single zip file with the name FirstName_RollNumber_01.zip.

- Submit a single zip file containing:
  (a) codes (b) report (c) Saved Models (d) Readme.txt
- There should be **Report.pdf** detailing your experience and highlighting any interesting results. Kindly don't explain your code in the report, just explain the results. Your report should include your comments on the results of all the steps, with images, for example, what happened when you changed the learning rate, etc.
- Readme.txt should explain how to run your code, preferably it should accept the command line arguments e.g dataset path used for training the model.
- The assignment is only acceptable in .py files. No Jupyter notebooks.
- In the root directory, there should be 2 python files, a report, and a folder containing saved models.
- Root directory should be named as **FirstName_RollNumber_03**
- Your code script files should be named as **'rollNumber_03_task1.py'**
- You have to submit a .py code file**.** Follow all the naming conventions.
- For each convention, there is a 3% penalty if you don't follow it.
- Email the instructor or TA if there are any questions. You cannot look at others' code or use others' code, however, you can discuss it with each other. **Plagiarism will lead to a straight zero with additional consequences as well.**
- 10% (of obtained marks) deduction per day for a late submission.
- Late submissions will only be accepted till **26th March at midnight.**
- **DON'T RESUBMIT THE DATASETS PROVIDED IN YOUR SUBMISSION.**
- Use the provided dataset for this assignment, do not use any other dataset.
- **Marks will be allocated according to the viva assessment**. In case your code functions correctly but lacks explanation, marks will not be awarded for that segment.
- In case of any ambiguity email the TA and keep Dr. Mohsen in cc.

Note: For this assignment (and for others in general), **it is strictly prohibited to search online for any kind of implementation (except the link provided and do not copy code from provided links)**. You are not allowed to share code or view others' code. You should only possess your own implementation related to the assignment.

Objectives: In this assignment, you will write the code for customized CNN and Transfer learning models. The goals of this assignment are as follows.

- Understand how to use efficient convolutional neural networks for classification tasks.
  - Understand how you can prepare a data loader for any dataset.
  - Understand how a model can be created, trained, validated, and saved in PyTorch
- How to design your own customized feature extractor using convolutional neural networks
- How to leverage the pre-trained model for transfer learning

⚠ **Important:** With your submission, you also need to submit one .py file named **'rollNumber_03_allCode.py'** in your zip folder. This file should contain the code of all 'code files' in your project. NO SUBMISSION WILL BE ACCEPTED WITHOUT IT.

Task 01: (100 marks)

MNIST Classification with Convolutional Neural Networks (CNNs) in a supervised setting.

Instructions:

- Load data using Data Loader class.
- You can use nn.Module layers to implement the model.
- Split the training data into training and validation sets, with 10% or 15% of the training set aside for validation. This function must be implemented from scratch, without using library functions.

Dataset Details: MNIST is a widely used benchmark dataset in the field of computer vision and machine learning. It contains 60,000 training images and 10,000 testing images of handwritten digits, with 10 classes ranging from 0 to 9. Each sample is 28x28 in size which is a 2D grayscale image. You are provided with a dataset having two zip files (train and test) each containing an image folder and a corresponding CSV file having two columns i.e. image name and labels. Your goal is to design/extend the data loader class according to this data and make it in the form of a data loader object with batch size divide train set to train and validation set in this data loader class and use during training the model. Pytorch dataset class is responsible for loading images and ground truths and applying transformations. You will create a dataset class inheriting torch.utils.data. For more details about DataLoader visit Pytorch official documentation.

Normalize Image in Transform:
When you read an image into memory, the pixels usually have 8-bit integers between 0 and 255 for all three channels. But regression models (including neural networks) prefer floating point values within a smaller range. Often, you want values to have a mean of 0 and a standard deviation of 1 like the standard normal distribution. Doing this transformation is called normalizing your images. In PyTorch, you can normalize your images with torchvision, a utility that provides convenient preprocessing transformations. For each value in an image, torchvision.transforms.Normalize() subtract the channel mean and divides it by the channel standard deviation.

One-hot Encoding:
You need to convert the labels to one hot encoding because we now have 10 classes and their labels are 0,1,2,3...and 9. For each training sample, you need to generate a vector of length 10, whose all indices will be zeros except the index of its original label which will be 1. For example,

Training Sample	True Label	One-hot-Encoding										
X1	5	0	0	0	0	0	1	0	0	0	0	0
X2	7	0	0	0	0	0	0	0	1	0	0	0
X3	0	1	0	0	0	0	0	0	0	0	0	0

To do: The objective is to develop a Convolutional Neural Network (CNN) model to classify the MNIST images into their respective digit labels. The task will involve the following steps:

1. **Data Preprocessing**: You are required to follow the above-given guidance to prepare a dataset for training. This will involve loading and normalizing the data, as well as splitting the data into training and validation sets.
2. **Network Building**: Construct a CNN model with multiple convolutional and batch normalization layers, along with at least one pooling layer. Experiment with different layers

and explain their performance in the report. After feature extraction add one or more fully connected (FC) layers along with output layer (see Figure 1). The model should be trained using the training set and validated using the validation set (separate from training set).

a. **Validation set**: You have to write code for separate split the dataset into training and validation. Your input is training set and output is training and validation set. You are not allowed to use any function from the library to split the data.
b. **Metric**: The model's performance should be evaluated using accuracy as the metric. Train, validation, and test result will be added to the report. Missing results will be awarded as 0 in this part.

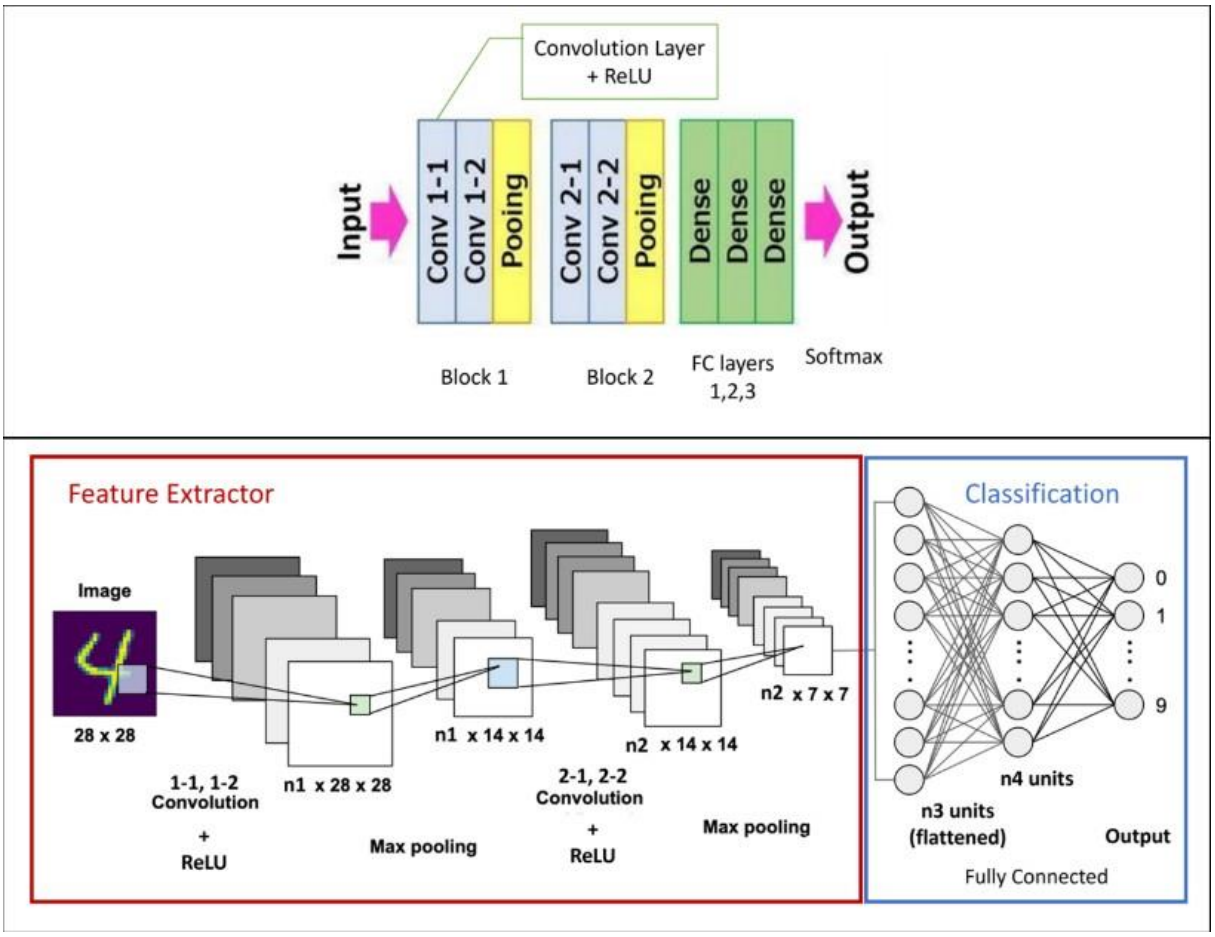

Figure 1: Building blocks of Convolutional Neural Network

3. **Training**: Now create a function to train the initialized network. You also have to keep track of loss and accuracy on training and validation data for each epoch to display loss and accuracy curves.

The train method of your model should be like as below.

model, loss_array, accuracy_array = train(net, train_set_x, train_set_y, valid_set_x, valid_set_y, learning_rate, training_epochs, loss_func, optimizer, batch_size)

---

4. **Back-Propagation**: For Back-Propagation you can use auto-grad (PyTorch) to update gradients. For this, you will need to initialize all weight matrices in network initialization using tensors (of PyTorch). At the time of training use the 'requires_grad=True' parameter in tensors. You can use loss.backward() and optimizer.step() functions to compute gradients and

update weights. You can also use the built-in optimizer and loss functions of PyTorch.

5. **Early Stopping and Learning rate decay**: In the training function, you have to keep track of training and validation loss. Early stopping is done when validation loss starts increasing while training loss is decreasing or constant. If training loss is not decreasing up to several epochs then we can decrease the learning rate (learning rate decay). If training loss does not decrease for many epochs then we can also apply early stopping.
6. **Loss function**: Use the CrossEntropy. Optimizer and hyper-parameters are up to you.
7. **Save Network**: Write a function that saves your network so that it can be used later without training.
8. **Load Network**: Write a function that loads your saved network so that it can be used without training. The function should return the loaded model.
9. **Testing Step**: Now create a function that uses a trained network to make predictions on a test set. The function should return predictions of the model.

   pred = test(net, model, test_set_x, test_set_y)
10. **Runtime Testing**: Write a function that takes input an image from the user at run time, preprocesses it, and predicts its class label is 0 to 9.
11. **Visualize Step:**
    a. Write a function that plots loss and accuracy curves and sample images and predictions made by the model on them. Show your loss and accuracy curves on training and validation sets. For a testing set, you should report only accuracy.
    b. The function should also plot the confusion matrix, f1_score, and accuracy of test data. Review sklearn.metrics for getting different metrics of predictions. It will also visualize 5 accurate and wrong predictions.

Summary:

1. Prepare data loader.
2. Design Architecture
3. Train Network
4. Save and load model.
5. Report results on test_set
6. Visualize the accuracy curve, loss curve, confusion matrix, and wrong and accurate prediction.
7. Visualize weights.

Note: The design decisions regarding the neural network and convolution layers are at your discretion. Conduct different experiments and provide analysis for each in the report.

Task 02: (100 marks)

Transfer learning for Image Dataset 2 Classification with Convolutional Neural Networks (CNNs).

Instructions:

- Students with even roll numbers should utilize VGG16, while those with odd roll numbers must employ ResNet34 as their base architecture.
- For example, "MSCS22013" ends with 13, therefore ResNet34 will be utilized in this case.
- If your model isn't matched with your roll number, it will not be accepted for evaluation.

Dataset Description (Image_classification.zip): This dataset comprises 7 classes, each directory. corresponding to the label name of the image. It is an image classification dataset. Initially, you need to assign a label for each class. Let's assume "bike" is labeled as 0, and each image from the bike directory will be assigned as 0, following the same pattern for the remaining classes.

Data Preprocessing: Download the **Image_classification.zip**. Follow the guidance same from task 1 to prepare the dataset using DataLoader class. This will involve Loading, Resizing the image to the input size of Vgg16/ResNet34 architecture, and normalizing the data, as well as splitting the data into training and validation sets. Standard input size for image is (224, 224, 3).

Split Data: You are required to divide the data into train, test, and validation sets. You must write your own method for splitting; library functions are not permitted. The distribution of each set should be as follows:

1. Train: 70%
2. Test: 15%
3. Validation: 15%

Part-1

Model Building:

- Begin by studying the model architecture from published articles before initiating your implementation on VGG16 or ResNet34.
- **Freeze all CNN blocks except the last one**. Design a model architecture (ResNet34 or VGG16) from scratch. Then, load the weights of ImageNet and fine-tune your model according to the dataset, adjusting the output layer of the model to 7 units to accommodate the 7 classes. Hyper-tune the model accordingly.

Weights:

- ResNet34: Download weights from the following link: ResNet34 Weights
- VGG16: Download weights from the following link: VGG16 Weights

Part2:

Now, follow the steps to load the model using the PyTorch library with weights. Fine-tune the model and modify the last layer from 1000 to 7. Utilize appropriate loss function and optimizer and analyze the results and add it in the report.

Implement all the methods for training, testing and everything else as mentioned in the task-1, missing of any function will lead to penalty.

Report

- In the report, you must include the training and validation graphs, as well as the testing accuracy and F1-score for task 1 and task2 (part-a, and part-b).
- Every observation about the dataset/model training testing will be mentioned in detail, your model is overfit/underfit, which used and what is the effect of learning rate, the rationale behind the chosen loss function, and any other relevant aspects.

Evaluation

- It's important to note that any concept covered in the assignment may be queried during evaluation, hence thorough preparation is essential. Failure to perform well during the oral examination (viva) may result in deductions from this segment of your marks.
- You may be questioned on various aspects, such as the performance of your model, its application, the reasoning behind selecting a specific loss function, among others. Therefore, it's crucial to be well-prepared for a comprehensive understanding of the assignment.
