Deep Learning
Assignment 2

Classification of MNIST Using Pytorch

Submission Guidelines:

Submission:

Submit all of your codes and results in a single zip file with the name FirstName_RollNumber_02.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 file(Task 1, Task 2), a report, and a folder containing saved models.
- The root directory should be named **FirstName_RollNumber_02**
- Your code notebooks should be named as **'rollNumber_02.py**
- 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.
- The submissions will only be accepted till—-----------.
- DON'T RESUBMIT THE DATASETS PROVIDED IN YOUR SUBMISSION.
- Report without code/experiment will be rewarded zero
- The provided dataset is different from the previous one. It is specifically designed for this assignment and it is imbalanced. **Use the provided dataset for this assignment, do not use the previous one or any other dataset**.
- If you don't follow this dataset and don't prepare Dataloader your 50% marks will be deducted

Due Date: —------------------------

Note: For this assignment (and for others in general) you are not allowed to **search online for any kind of implementation.** Do not share code or look at the other's code. You should not have any implementation related to the assignment, other than your own. In case of any confusion please reach out to the TAs (email them or visit them during their office hours).

Objectives:

- Understand how neural networks can be used for the classification of images
- Understand how you can prepare a data loader for any dataset
- Understand how a model can be created, trained, validated, and saved in PyTorch
- Understand how to read/write and process images in python
- Understand different building blocks of creating ANN and CNN architectures i.e. conv2d, batch normalization, dropout, max polling, etc.

NOTE: It's recommended to use visual studio code/pycharm

you can use Google Colab for running the code but you have to submit a .py code file

Dataset Details:

MNIST dataset contains 10 classes provided dataset have two zip file(train and test) each contain an image folder and 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 apply transformations. You will create a dataset class interhting `torch.utils.data.Dataset` which requires two main function to be implemented:

`__len__` so that len(dataset) returns the size of the dataset.

`__getitem__` to support the indexing such that dataset[i] can be used to get iith sample.

Following is the example of a dataset class for face landmarks dataset (*more detailed tutorial on this link*).

```
class FaceLandmarksDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
           csv_file (string): Path to the csv file with
           annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to
            be applied
                on a sample.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.landmarks_frame)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.root_dir,
                                self.landmarks_frame.iloc[idx, 0])
        image = io.imread(img_name)
        landmarks = self.landmarks_frame.iloc[idx, 1:]
        landmarks = np.array([landmarks])
        landmarks = landmarks.astype('float').reshape(-1, 2)
        sample = {'image': image, 'landmarks': landmarks}

        if self.transform:
            sample = self.transform(sample)

        return sample
```

Link:
https://drive.google.com/drive/folders/1wWCUrSnAZwdzzKsH3vB27bszdqy37es7?usp=sharing

Normlize 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()` subtracts the channel mean and divides by the channel standard deviation.

Task 1:

Classify Images using Fully Connected Neural Networks

In this part, you will create a complete neural network architecture in PyTorch consisting of multiple layers. You are required to report results obtained from different experiments on the MNIST dataset.

Steps

1. **Data loading and normalization**

2. **Initialize Network**

You have to create a function to initialize the network. Parameters should be entered by the user. If your system has a GPU you can turn it on to improve performance. You can use PyTorch's built-in functions to create and initialize the network.

```
net = init_network(no_of_layers, input_dim, neurons_per_layer,
dropout)
```

neuron_per_layer should be a list having elements describing the number of neurons in each hidden layer. The last element of the list should be the dimension of output (In our case 10).

Draw a diagram for the architecture you build and add that to the report using https://alexlenail.me/NN-SVG/AlexNet.html

For more information about dropouts:

https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Note: you have to choose carefully all the parameters for better accuracy

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.

```
net = train(net, train_set_x, train_set_y, valid_set_x,
valid_set_y, learning_rate, training_epochs, loss_func,
optimizer)
```

Optimizer: Use **SGD** and **Adams** and compare the results and add to the report

Loss: Try **categorical cross-entropy** and **focal loss** and compare the results and add to the report

- This function returns a trained Neural Network `net, loss_array, and accuracy_array`

  The standard to choose a learning rate is it should be in the range of less than 1 and greater than 10^-6

  You should not train more than 20 epochs.

- **Early Stopping and Learning rate decay**

In the train 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.

4. **Save Network**

Write a function that saves your network so that it can be used later without training.

5. **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.

6. **Testing Step**

Now create a function that uses a trained network to make predictions on a test set.

```
pred = test(net, model, test_set_x, test_set_y)
Function should return predictions of model.
```

7. **Visualize Step**

Write a function that plots loss and accuracy curves and sample images and predictions made by the model on them. 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 4 accurate and wrong predictions

8. **Main Function**

Now create a main function that calls all the above functions in the required order. The main function should accept “Path of dataset”, “size of training data”, “size of validation data”, “size of testing data”, “number of hidden layers”, “list having a number of neurons in each layer”, “Loss function”, “optimizer”, “batch size”, “learning rate”, “Is GPU enable(By default false if you do not have GPU)”, “drop out”, “Is training (default is False)”, “Visualize Results (Default is False)” and “training epochs” as input. If “Is training” is true, the function should start training from scratch, otherwise, the function should load the saved model and make predictions using it. While performing experiments you can use Google Colab or Jupyter Notebook. Your code must print your name, roll no, your all best/default parameters, training, validation and testing losses, accuracies, f1 scores, and confusion matrix on test data and accuracy and loss curves of training and validation data.

Bonus: Get a bonus on class-based implementation

Task 2:

Classify Images using Convolutional Neural Network using Pytorch

In this part, you will create a complete convolutional neural network architecture in PyTorch consisting of multiple layers. You are required to report results obtained from different experiments on the MNIST dataset.

1. **Data loading and normalization**
   use the same data loader you built-in task 1

2. **Initialize Network**

You have to create a function to initialize the network. Parameters should be entered by the user. If your system has a GPU you can turn it on to improve performance. You can use PyTorch's built-in functions to create and initialize the network.

```
net = init_network(output)
```

neuron_per_layer should be a list having elements describing the number of neurons in each hidden layer. The last element of the list should be the dimension of output (In our case 10).

Draw the diagram for the architecture you build and add that to the report using https://alexlenail.me/NN-SVG/AlexNet.html

Use All other steps same as in Task 1 including training, testing, model save, model load, visualize results, early stoping, learning rate decay

3. **Visualize features**

Plot learned filters of your last convolution layer

4. **Activation Function**

Use different kinds of activation functions and report your analysis based on the results which one is performing better and why? You do not have to report results on all of them. Just use two of them.

- Tanh
- ReLU
- Leaky ReLU
- Parametric ReLU
- Exponential Linear Units (ELUs)
- Swish
- Gaussian Error Linear Unit GeLU

5. **Batch Normalization**

Use the batch normalization layer in your CNN architecture and compare the results with batch normalization. Do share in the report which layers you applied the Batch Normalization, to and why. Do share and cite any reference you read for making such a decision.

6. **Dropout**

Use the dropout layer and compare the results. Do share in the report which layers you applied the dropout, on and why. Do share and cite any reference you read for making such a decision.

Summary:

1. Prepare data loader
2. Design Architecture

3. Train Network
4. Save and load model
5. Report results on testset
6. Visualize accuracy curve, loss curve, confusion matrix, wrong and accurate prediction

The design decision of the neural network and convolution layer depends on your own choice. But you are required to perform different experiments and provide the analysis of all your experiments in the report. Your report must include a visualization of the output of the last convolution layer before and after the pooling.
Please show your loss and accuracy curves on training and validation sets. For a testing set, you should report only accuracy.

Marks Distribution

- **Code** **50%**
  - o Dataloader 10
  - o Architecture (Task1, task 2) 10
  - o Training with early stopping and learning rate decay 15
  - o Activation function 5
  - o Visualization 10
- **Report** **40%**
  - o Loss and Accuracy Curves for validation and training dataset. 2.5
  - o Confusion matrices, Recall, and Accuracy for the testing set. 2.5
  - o Figures along with labels for correct predictions and wrong ones. 2.5
  - o Plot learned filters of your last convolution layer using matplotlib. 2.5
  - o Analysis of your different experiments. 2.5
  - o Show what happens when we do not use convolution layers and when we use convolution layers with neural networks. Show ROC curves, accuracy/loss curve, confusion matrix, and tsne plot. 5.0
  - o Show the **tsne plot** of the last fully connected layer after the first epoch and when training is done. Show the difference between both. 2.5
  - o Report what you learned from this assignment, your analysis, and if you find something innovative or interesting in the conclusion section. 2.5
  - o Task 1 → Report the accuracy by changing the number of neurons in the hidden layers, or a number of hidden layers or changing loss functions, batch size, learning rate, epochs, and the ratio of training and testing data, etc. 5.0
  - o Task 2 → Report the accuracy by adding/subtracting batch norm or dropout layer or changing loss functions, batch size, learning rate, epochs, and the ratio of training and testing data, etc. 5.0
  - o Describe the reasoning for choosing architecture for both tasks. 2.5
  - o For task 1 Describe the reasoning for choosing the number of hiding layers and nodes. 2.5
  - o For task 2 describe the reason for choosing the number of filters for each Conv layer and why you choose a number of FC layers. 2.5
- **Viva** **10%**

Note: Deliverables will be updated in case there are additional tasks in the assignment.

🤓 Good Luck 🤓