Deep Learning
Assignment 3

Classification of MNIST Using Pytorch

[Half the grade of normal assignment.]

Deadline : 30th May 2022, 11:59 pm

Submission Guidelines:

Submission:

Submit all of your codes and results in a single zip file with the name FirstName_RollNumber_02_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 **1** python file, a report, and a folder containing saved models.
- The root directory should be named **FirstName_RollNumber_02**_02
- Your code notebooks should be named as **'rollNumber_02.**_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**.

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:

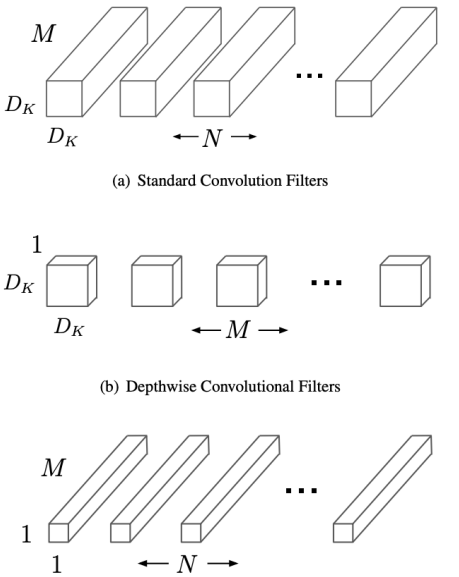
- Understand how to design different CNN architectures
- Understand what is Depthwise separable convolution
- Understand 1x1 convolution
- Understand how you can understand new architectur's implementation

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

Introduction (from paper)

The MobileNet v1 model is based on depthwise separable convolutions which is a form of factorized convolutions which factorizes a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution. For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs the depthwise convolution. A standard convolution both filters and combines inputs into a new set of outputs in one step. The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size.



(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 1

Figure shows how a standard convolution (a) is factorized into a depthwise convolution (b) and a 1 × 1 pointwise convolution (c). A standard convolutional layer takes as input a DF × DF × M feature map

F and produces a DF × DF × N feature map G where DF is the spatial width and height of a square input feature map1 , M is the number of input channels (input depth), DG is the spatial width and height of a square output feature map and N is the number of output channel (output depth). The standard convolutional layer is parameterized by convolution kernel K of size DK ×DK ×M×N where DK is the spatial dimension of the kernel assumed to be square and M is number of input channels and N is the number of output channels as defined previously. The output feature map for standard convolution assuming stride one and padding is computed as:

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \qquad (1)$$

Standard convolutions have the computational cost of:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \qquad (2)$$

where the computational cost depends multiplicatively on the number of input channels M, the number of output channels N the kernel size Dk × Dk and the feature map size DF × DF . MobileNet models address each of these terms and their interactions. First it uses depth wise separable convolutions to break the interaction between the number of output channels and the size of the kernel.

The standard convolution operation has the effect of filtering features based on the convolutional kernels and combining features in order to produce a new representation.The filtering and combination steps can be split into two steps via the use of factorized convolutions called depthwise

separable convolutions for substantial reduction in computational cost. Depthwise separable convolution are made up of two layers: depthwise convolutions and pointwise convolutions. We use depthwise convolutions to apply a single filter per each input channel (input depth). Pointwise convolution, a simple 1×1 convolution, is then used to create a linear combination of the output of the depthwise layer. MobileNets use both batchnorm and ReLU nonlinearities for both layers. Depthwise convolution with one filter per input channel (input depth) can be written as:

$$\hat{\mathbf{G}}_{k,l,m} = \sum_{i,j} \hat{\mathbf{K}}_{i,j,m} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \qquad (3)$$

where Kˆ is the depthwise convolutional kernel of size DK × DK × M where the mth filter in Kˆ is applied to the mth channel in F to produce the mth channel of the filtered output feature map Gˆ . Depthwise convolution has a computational cost of:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \qquad (4)$$

Depthwise convolution is extremely efficient relative to standard convolution. However it only filters input channels, it does not combine them to create new features. So an additional layer that computes a linear combination of the output of depthwise convolution via 1 × 1 convolution is needed in order to generate these new features. The combination of depthwise convolution and 1 × 1 (pointwise) convolution is called depthwise separable convolution which was originally introduced in [26]. Depthwise separable convolutions cost:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \qquad (5)$$

which is the sum of the depthwise and 1 × 1 pointwise convolutions. By expressing convolution as a two step process of filtering and combining we get a reduction in computation of:

$$\begin{aligned} & \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} \\ = \quad & \frac{1}{N} + \frac{1}{D_K^2} \end{aligned}$$

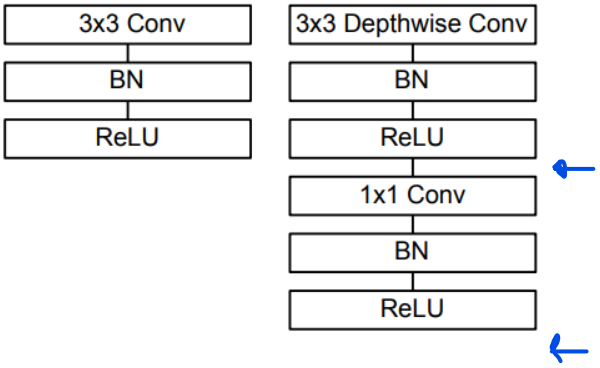
MobileNet uses 3 × 3 depthwise separable convolutions which uses between 8 to 9 times less computation than standard convolutions at only a small reduction in accuracy. Additional factorization in spatial dimensions such as in[16, 31] does not save much additional computation as very little computation is spent in depthwise convolutions.

Architecture

The first layer of the MobileNet is a full convolution, while all following layers are Depthwise Separable Convolutional layers. All the layers are followed by batch normalization and ReLU activations. The final classification layer has a softmax activation. The full architecture is shown below

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

(a) MobileNet architecture (Source: table from the original paper)



(b) Left: Standard Convolutional layer, Right: Depthwise Separable Convolutional layers in MobileNet

Figure 2 MobileNet architecture

Figure shows the difference in architecture flow between normal CNN models vs MobileNets. On the left of the image, we see a 3x3 Convolutional layer followed by batch normalization and ReLU, while, on the right, we see the Depthwise (dw) Separable Convolutional layer — consisting of a 3x3 Depthwise Convolution with a batch norm and ReLU followed by a 1x1 pointwise convolution followed by a batch norm and ReLU.

For more detail read the paper: Link

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.

Same dataset as part 1

Link:

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

Task 1:

Image Classification Using MobileNet

Build the MobileNet network specified in Figure 2(b) and achieve at least 60% test accuracy. In the assignment, you should define your “ConvDw Block” inheriting as shown in Figure 2(b).

Steps

1. **ConvDw Block**

Write a ConvDw class in which you will define a standard MobileNet block as shown in figure 2(b) right. The init function should accept the number of input channels, number of output channels and stride as input. Define your Depthwise and pointwise convolution.

2. Initialize Network

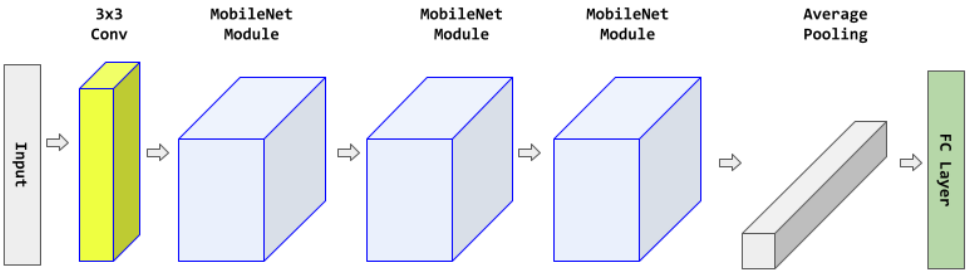


Figure 3: sample network

You have to create a function to initialize the network that inputs five parameters. 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(conv_block_count, mobilenet_block_count, input_dim, num_classes)
conv_block_count: No of convolutional blocks
mobilenet_block_count: No of mobileNet Blocks
input_dim: Image shape
num_classes: No of classes
```

Pattern: input → conv layers → mobilenet layers → average pooling → fc layer → output

Conv block and mobileNet block is mentioned in Figure 2(b) left right

You have to design two architectures

1. Design the architecture same as shown in Figure 3
2. Design an architecture with 4 convolution layer followed by GAP and FC layer

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)
```

The choice of loss function and optimizer and hyper-parameters is up to you.

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 the 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", conv_block_count, mobilenet_block_count, input_dim, num_classes "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.

Summary:

1. Prepare data loader(Assignment 2 code)
2. Design Conv and MobileNet block
3. Design Architecture
4. Train Network
5. Save and load model
6. Report results on testset
7. Visualize accuracy curve, loss curve, confusion matrix, wrong and accurate prediction

Marks Distribution

- **Code** **50%**
  - Dataloader 10
  - Design MobileNet block 10
  - Design architectures mentioned 15
  - Global Average Pooling 5
  - Visualization 10
- **Report** **40%**
  - Compare results of simple convolution block base architecture vs mobilenet block based architecture and report results with reasoning 5.0

- Use no of different conv block and mobileNet block and compare its accuracy 5.0
- Confusion matrices, Recall, and Accuracy for the testing set. 2.5
- Figures along with labels for correct predictions and wrong ones. 2.5
- Plot learned filters of your last convolution layer using matplotlib. 5.0
- Show what happens when we do not use MobileNet block and when we use MobileNet block in your architecture. Show ROC curves, accuracy/loss curve, confusion matrix, and tsne plot. 10.0
- Report what you learned from this assignment, your analysis, and if you find something innovative or interesting in the conclusion section. 5.0
- Report the accuracy by changing loss functions, batch size, learning rate, epochs, and the ratio of training and testing data, etc. 5.0

- **Viva** **10%**

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

🤓 Good Luck 🤓