I selected second part of question 5 to implement. In this part of the problem, after training every two epochs, the trained model was validated over the validation dataset.

The code structure:

1-\_select\_device:

This function will determine on which device the code must be executed, CPU or GPU. If torch.cuda is available, then the global variable of the device will be “cuda:0:, otherwise, it will be “cpu”. No input and no output are required. Later to use this device I used the input.to(device) command to push the data and model to the device. In my case the cuda is available on my laptop and all code were run on the GPU.

2- Sampler

In this method, I coded a sample creator function. This function will receive the dataset on which going to create the training, validation, and test dataset. And will return three arrays including the data items’ indices of the training, validation and test dataset.

In this function, there are 4 variables that define the function behavior. Divide\_data, which value is between 0 to 1 and is used to shrink the size of the dataset. So, if it is one the whole data set will use and if it is zero no data will be used. Two other variables named as validation\_data\_size and test\_data\_size is used as to specify a percentage based on the training dataset size

3- \_make\_dataloaders:

In this function, the transform is defined. Then based on the return values of the sampler function the training, test and validation loader will be initialized and return.

4- class ResNet50\_CIFAR: same as defined in the original code.

5- train:

The main training process is the same as what defined in the original code file. This process includes, forward, loss calculation, backward, and optimizer step. In this function, for the fixed number of epochs, the training will execute over the training data. As the question asked for, on every odd epoch the model will be validated over the validation dataset (valid\_loader) and the result of the loss function, model status and optimizer will be saved in three global variables. Later these values will be used to find the best validation and plotting the loss value changes over the epochs. This function receives the model, optimizer, criterion, train loader, and validation loader as input parameters.

6- validation:

In this function, the model will be validated over the validation dataset. This process includes, forward, loss calculation. The validation result, model state, and optimizer state will be saved in three global arrays for further use.

7- Test:

After finishing the training, this code runs to taste the model accuracy. The accuracy of my model was 65%.

8- Main function:

First, the device is selected. Then, a new model is instantiated from ResNet50\_CIFAR. Then the model is pushed to the device. The after setting the loss function and optimizer the model and the training dataset are pushed to the training function, in which the validation process also occurs. Then the result of the validations is compared to find the best (minimum) loss value. The plot of the loss value changes over the epochs is plotted. And the best result is saved in the best\_model.pth.

I used following command to save the result:

torch.save({'tested\_accuracy':accuracy, 'best\_model': validation\_model\_history [best\_loss\_indx] ,'best\_optimizer': validation\_optimizer\_history[best\_loss\_indx] ,'best\_epoch': best\_loss\_indx, 'best\_loss': validation\_loss\_history[best\_loss\_indx]}, 'bestmodel.pth')

the best\_loss\_index is calculated using numpy.argmin function over the loss calculated in all the validations (np.argmin(validation\_loss\_history).