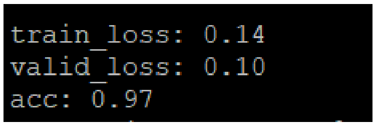
ASSIGNMENT 4
SHIVAM AGNIHOTRI (SA22BB)

1. **All your code (all working) that you wrote to train, validate and test the model.**

(complete code in folder)

```
# training

    while epoch < max_epochs:
        epoch += 1
        print(f"epoch no: {epoch}")
        train_loss = 0.0
        valid_loss = 0.0
        accuracy_val = 0.0

        model.train()

        # training

        for index, batch in enumerate(tqdm(train_dl)):
            optimizer.zero_grad()
            # get the input from batch
            input_data, target = batch
            # pass the input to model and get prediction
            output = model(input_data)
            # calculate the loss and store it in variable loss using the criterion
            loss = F.cross_entropy(output, target)
            # backpropagate the loss
            loss.backward()
            train_loss = train_loss + ((1 / (index + 1)) * (loss - train_loss))

            optimizer.step()
        scheduler.step()

        model.eval()

        # validation

        with torch.no_grad():
            for index, batch in enumerate(tqdm(test_dl)):

                # get the input from batch
                input_data, target = batch
                # pass the input to model and get prediction
                output = model(input_data)
                # calculate the loss and store it in variable loss using the criterion
                loss = F.cross_entropy(output, target)
                # calculate the accuracy and store it in variable acc using the function called accuracy
                acc = accuracy(output, target)
                valid_loss = valid_loss + ((1 / (index + 1)) * (loss - valid_loss))
                accuracy_val = accuracy_val + ((1 / (index + 1)) * (acc - accuracy_val))

        print(f"\ntrain_loss: {train_loss:.2f} \n"
              f"valid_loss: {valid_loss:.2f} \n"
              f"acc: {accuracy_val:.2f}")

        writer.add_scalar("train_loss", train_loss, global_step=epoch)
        writer.add_scalar("valid_loss", valid_loss, global_step=epoch)
        writer.add_scalar("accuracy", accuracy_val, global_step=epoch)

        if epoch % config["save_epoch"] == 0:
            model_save(Path(config["model_save_path"]), model, optimizer, scheduler, epoch,
"current")

            if accuracy_val > max_acc:
                max_acc = accuracy_val
                model_save(Path(config["model_save_path"]), model, optimizer, scheduler, epoch,
"best")

    writer.flush()

    writer.close()
```

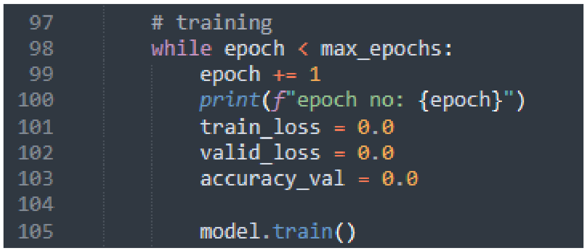
2. Screenshot and steps of training and validating the model.



```
train_loss: 0.14
valid_loss: 0.10
acc: 0.97
```

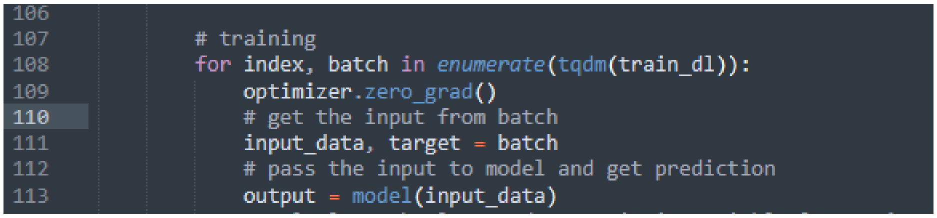
The model is trained and validated using the following steps:

- The code initializes the necessary variables, such as the epoch number, maximum accuracy, and maximum number of epochs specified in the configuration file.
- The model is put into training mode using model.train().



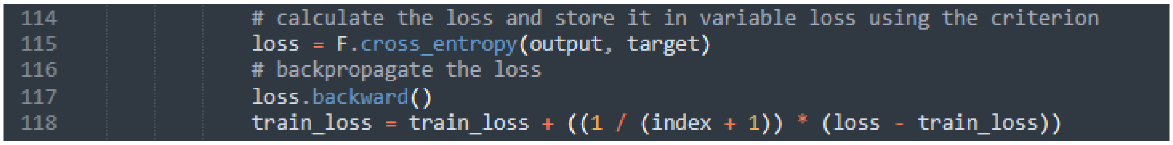
```
    # training
    while epoch < max_epochs:
        epoch += 1
        print(f"epoch no: {epoch}")
        train_loss = 0.0
        valid_loss = 0.0
        accuracy_val = 0.0

        model.train()
```

- The training dataset is loaded into a DataLoader, which returns a batch of images and their corresponding labels for each iteration of the loop.
- For each batch, the optimizer's gradients are zeroed using optimizer.zero_grad().
- The model is fed the input data (images) using output = model(input_data). The output is the predicted class labels for each image in the batch.



```

        # training
        for index, batch in enumerate(tqdm(train_dl)):
            optimizer.zero_grad()
            # get the input from batch
            input_data, target = batch
            # pass the input to model and get prediction
            output = model(input_data)
```

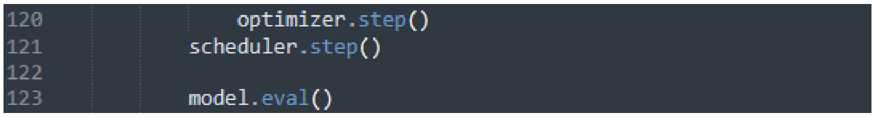
- The model's output is compared to the actual labels for each image in the batch, and the loss is calculated using cross-entropy loss (F.cross_entropy(output, target)).
- The loss is then backpropagated through the network using loss.backward().
- The training loss is updated using the formula train_loss = train_loss + ((1 / (index + 1)) * (loss - train_loss)). This formula is used to calculate a running average of the training loss.



```
            # calculate the loss and store it in variable loss using the criterion
            loss = F.cross_entropy(output, target)
            # backpropagate the loss
            loss.backward()
            train_loss = train_loss + ((1 / (index + 1)) * (loss - train_loss))
```

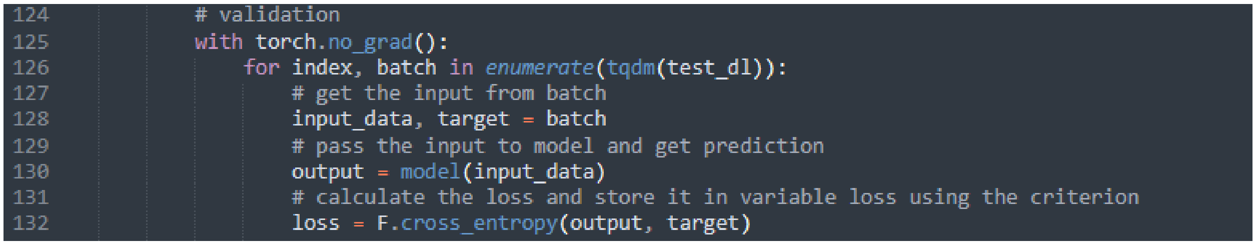
- The optimizer's weights are updated using optimizer.step().

- After all batches in the training dataset have been iterated over, the learning rate scheduler's step function is called using scheduler.step().
- The model is put into evaluation mode using model.eval().



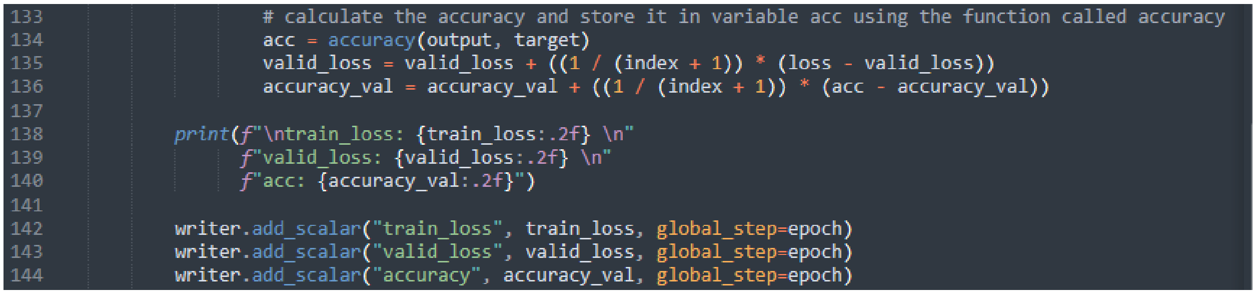
```
                optimizer.step()
            scheduler.step()

            model.eval()
```

- The testing dataset is loaded into a DataLoader, which returns a batch of images and their corresponding labels for each iteration of the loop.
- For each batch, the model's output is computed using output = model(input_data).
- The loss is calculated using cross-entropy loss (F.cross_entropy(output, target)), and the accuracy is calculated using a custom function accuracy(output, target).



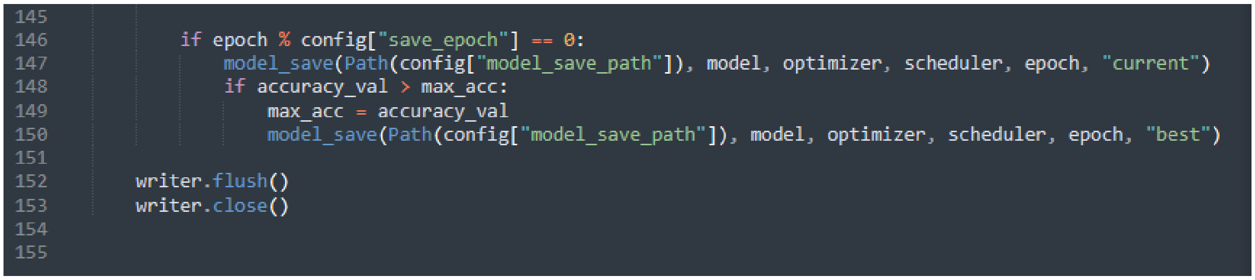
```
            # validation
            with torch.no_grad():
                for index, batch in enumerate(tqdm(test_dl)):
                    # get the input from batch
                    input_data, target = batch
                    # pass the input to model and get prediction
                    output = model(input_data)
                    # calculate the loss and store it in variable loss using the criterion
                    loss = F.cross_entropy(output, target)
```

- The validation loss and accuracy are updated using the same running average formula used for the training loss.
- After all batches in the testing dataset have been iterated over, the training and validation losses and accuracy are printed.
- The current epoch's losses and accuracy are written to TensorBoard using the SummaryWriter object.



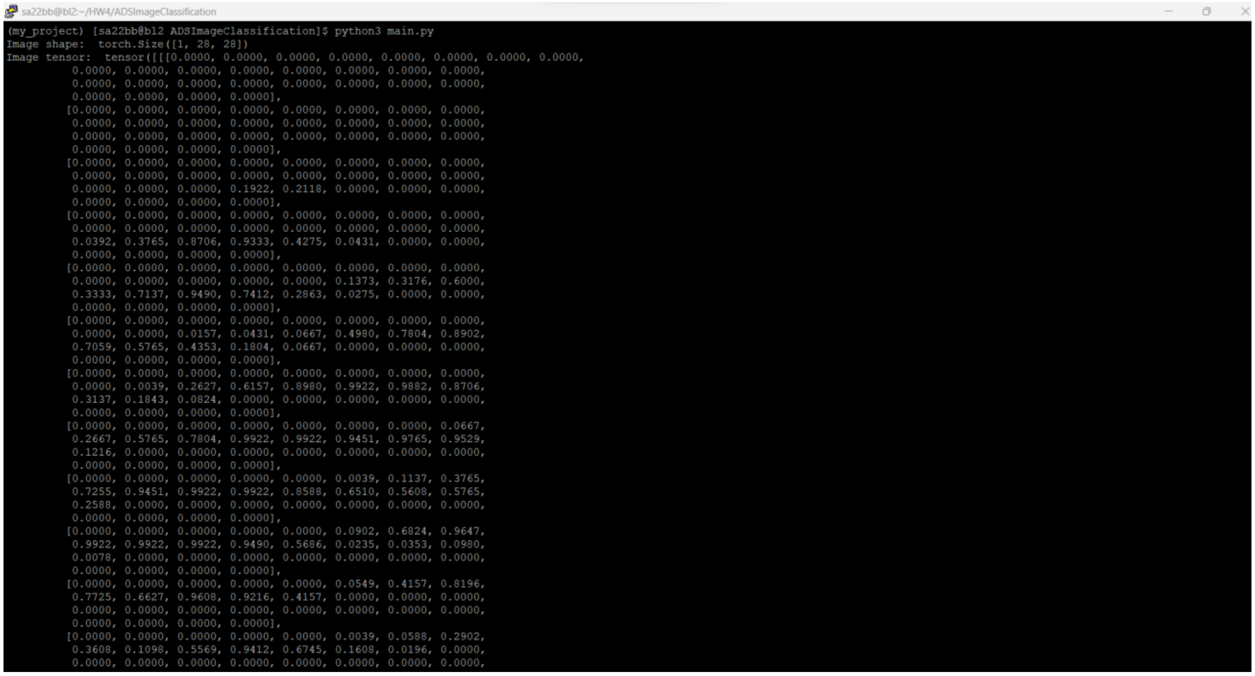
```
                    # calculate the accuracy and store it in variable acc using the function called accuracy
                    acc = accuracy(output, target)
                    valid_loss = valid_loss + ((1 / (index + 1)) * (loss - valid_loss))
                    accuracy_val = accuracy_val + ((1 / (index + 1)) * (acc - accuracy_val))

            print(f"\ntrain_loss: {train_loss:.2f} \n"
                  f"valid_loss: {valid_loss:.2f} \n"
                  f"acc: {accuracy_val:.2f}")

            writer.add_scalar("train_loss", train_loss, global_step=epoch)
            writer.add_scalar("valid_loss", valid_loss, global_step=epoch)
            writer.add_scalar("accuracy", accuracy_val, global_step=epoch)
```

- If the current epoch is a multiple of the save_epoch parameter specified in the configuration file, the model is saved to disk using the model_save() function. If the current accuracy is greater than the current maximum accuracy, the model is saved as the best checkpoint using the same function.
- The loop continues until the maximum number of epochs is reached, at which point the SummaryWriter object is flushed and closed.

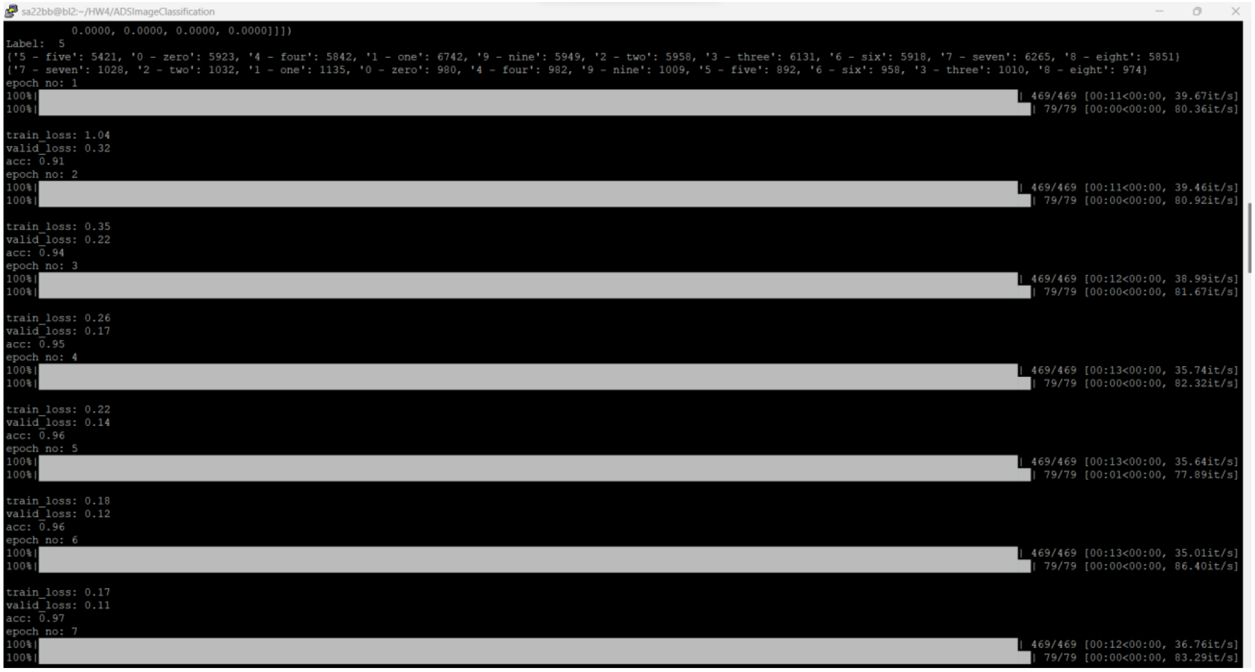


```python

            if epoch % config["save_epoch"] == 0:
                model_save(Path(config["model_save_path"]), model, optimizer, scheduler, epoch, "current")
                if accuracy_val > max_acc:
                    max_acc = accuracy_val
                    model_save(Path(config["model_save_path"]), model, optimizer, scheduler, epoch, "best")

        writer.flush()
        writer.close()


```

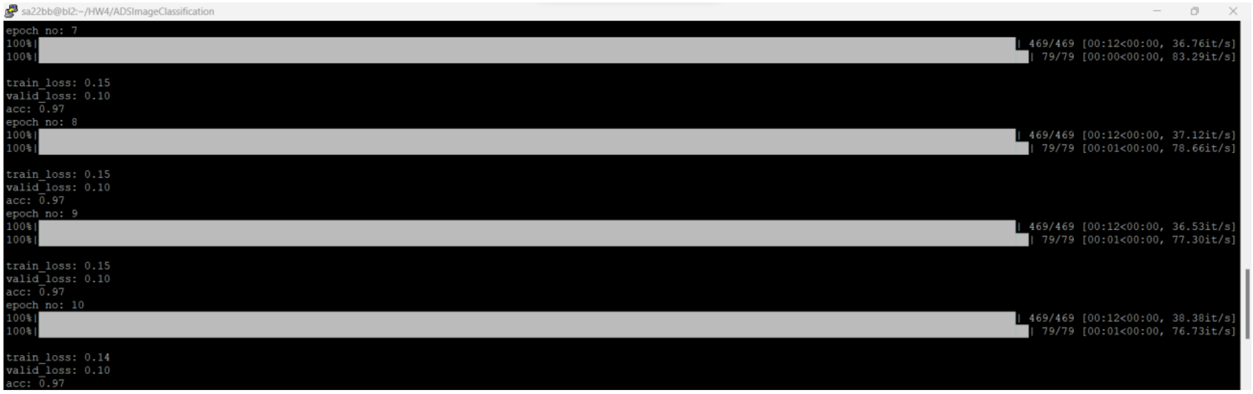
Run screenshots:



```
(my_project) [sa22bb@bl2 ADSImageClassification]$ python3 main.py
Image shape:  torch.Size([1, 28, 28])
Image tensor:  tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.1922, 0.2118, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0392, 0.3765, 0.8706, 0.9333, 0.4275, 0.0431, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1373, 0.3176, 0.6000,
         0.3333, 0.7137, 0.9490, 0.7412, 0.2863, 0.0275, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0157, 0.0431, 0.0667, 0.4980, 0.7804, 0.8902,
         0.7059, 0.5765, 0.4353, 0.1804, 0.0667, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0039, 0.2627, 0.6157, 0.8980, 0.9922, 0.9882, 0.8706,
         0.3137, 0.1843, 0.0824, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0667,
         0.2667, 0.5765, 0.7804, 0.9922, 0.9922, 0.9451, 0.9765, 0.9529,
         0.1216, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.1137, 0.3765,
         0.7255, 0.9451, 0.9922, 0.9922, 0.8588, 0.6510, 0.5608, 0.5765,
         0.2588, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0902, 0.6824, 0.9647,
         0.9922, 0.9922, 0.9922, 0.9490, 0.5686, 0.0235, 0.0353, 0.0980,
         0.0078, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0549, 0.4157, 0.8196,
         0.7725, 0.6627, 0.9608, 0.9216, 0.4157, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.0588, 0.2902,
         0.3608, 0.1098, 0.5569, 0.9412, 0.6745, 0.1608, 0.0196, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
```



```
          0.0000, 0.0000, 0.0000, 0.0000]]])
Label:  5
{'5 - five': 5421, '0 - zero': 5923, '4 - four': 5842, '1 - one': 6742, '9 - nine': 5949, '2 - two': 5958, '3 - three': 6131, '6 - six': 5918, '7 - seven': 6265, '8 - eight': 5851}
{'7 - seven': 1028, '2 - two': 1032, '1 - one': 1135, '0 - zero': 980, '4 - four': 982, '9 - nine': 1009, '5 - five': 892, '6 - six': 958, '3 - three': 1010, '8 - eight': 974}
epoch no: 1
100%|██████████| 469/469 [00:11<00:00, 39.67it/s]
100%|██████████| 79/79 [00:00<00:00, 80.36it/s]

train_loss: 1.04
valid_loss: 0.32
acc: 0.91
epoch no: 2
100%|██████████| 469/469 [00:11<00:00, 39.46it/s]
100%|██████████| 79/79 [00:00<00:00, 80.92it/s]

train_loss: 0.35
valid_loss: 0.22
acc: 0.94
epoch no: 3
100%|██████████| 469/469 [00:12<00:00, 38.99it/s]
100%|██████████| 79/79 [00:00<00:00, 81.67it/s]

train_loss: 0.26
valid_loss: 0.17
acc: 0.95
epoch no: 4
100%|██████████| 469/469 [00:13<00:00, 35.74it/s]
100%|██████████| 79/79 [00:00<00:00, 82.32it/s]

train_loss: 0.22
valid_loss: 0.14
acc: 0.96
epoch no: 5
100%|██████████| 469/469 [00:13<00:00, 35.64it/s]
100%|██████████| 79/79 [00:01<00:00, 77.89it/s]

train_loss: 0.18
valid_loss: 0.12
acc: 0.96
epoch no: 6
100%|██████████| 469/469 [00:13<00:00, 35.01it/s]
100%|██████████| 79/79 [00:00<00:00, 86.40it/s]

train_loss: 0.17
valid_loss: 0.11
acc: 0.97
epoch no: 7
100%|██████████| 469/469 [00:12<00:00, 36.76it/s]
100%|██████████| 79/79 [00:00<00:00, 83.29it/s]
```



```
epoch no: 7
100%|██████████| 469/469 [00:12<00:00, 36.76it/s]
100%|██████████| 79/79 [00:00<00:00, 83.29it/s]

train_loss: 0.15
valid_loss: 0.10
acc: 0.97
epoch no: 8
100%|██████████| 469/469 [00:12<00:00, 37.12it/s]
100%|██████████| 79/79 [00:01<00:00, 78.66it/s]

train_loss: 0.15
valid_loss: 0.10
acc: 0.97
epoch no: 9
100%|██████████| 469/469 [00:12<00:00, 36.53it/s]
100%|██████████| 79/79 [00:01<00:00, 77.30it/s]

train_loss: 0.15
valid_loss: 0.10
acc: 0.97
epoch no: 10
100%|██████████| 469/469 [00:12<00:00, 38.38it/s]
100%|██████████| 79/79 [00:01<00:00, 76.73it/s]

train_loss: 0.14
valid_loss: 0.10
acc: 0.97
```