Introduction to PyTorch

This tutorial shows the basics of PyTorch library.

We design simple Neural Networks for classification task on MNIST dataset.

You're probably going to need PyTorch documentation:

https://pytorch.org/docs/stable/index.html

and tutorials:

https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

In [ ]:
```
!mkdir data
```

1. PyTorch is based on tensor operations. First, let's try to use them:

- create simple python list with four values and convert it to PyTorch tensor
- create numpy array with random values and shape (1,3,7,7) and convert it to Pytorch tensor
- create PyTorch tensor with random values and shape (1,3,7,7) with preset seed
- create PyTorch tensor with linear space in range from -5 to 15 and reshape it to tensor with shape (1,3,7,7)
- create PyTorch tensor of zeros with shape (1,3,7,7)

For display use `print` function.

In [17]:
```
import torch
import numpy as np
from typing import Tuple
import tqdm
from tabulate import tabulate
python_list = [1, 2, 3, 4]
tensor_from_list = torch.tensor(python_list)
print(f"Tensor from list: {tensor_from_list}")

numpy_arr = np.random.rand(1, 3, 7, 7)
tensor_from_numpy = torch.tensor(numpy_arr)
print(f"Tensor from numpy arr: {tensor_from_numpy}")

# dzięki temu przy kazdym uruchomieniu otrzymujemy te same losowe wartośc
random_tensor = torch.rand(1, 3, 7, 7)
torch.manual_seed(0)
print(f"Random tensor with preset seed: {random_tensor}")
```

```python
# linspace tworzy nam liniowe rozłoenie danych (czyli chyba ciąg? taka sa
linear_tensor = torch.linspace(-5, 15, steps=1*3*7*7).reshape(1, 3, 7, 7)
print(linear_tensor)

zeros_tensor = torch.zeros(1, 3, 7, 7)
print(f"Zeros tensor: {zeros_tensor}")
```

```
Tensor from list: tensor([1, 2, 3, 4])
Tensor from numpy arr: tensor([[[[0.1979, 0.2089, 0.8812, 0.3688, 0.5101,
0.0525, 0.4303],
          [0.9911, 0.0207, 0.5338, 0.7258, 0.9917, 0.8546, 0.1344],
          [0.4611, 0.9622, 0.4394, 0.2708, 0.8185, 0.9220, 0.1635],
          [0.6040, 0.1079, 0.8381, 0.2752, 0.3549, 0.1486, 0.9634],
          [0.0252, 0.1644, 0.1698, 0.5441, 0.2444, 0.1428, 0.9955],
          [0.5414, 0.9531, 0.8055, 0.7594, 0.8380, 0.9566, 0.1384],
          [0.4404, 0.6970, 0.7797, 0.6291, 0.5225, 0.2466, 0.7973]],

         [[0.1731, 0.0968, 0.3959, 0.6689, 0.0241, 0.9399, 0.8844],
          [0.4737, 0.0414, 0.4566, 0.8066, 0.3761, 0.5983, 0.3666],
          [0.5807, 0.8480, 0.1096, 0.3851, 0.8597, 0.7061, 0.1032],
          [0.8467, 0.8615, 0.8638, 0.6549, 0.5167, 0.6238, 0.3705],
          [0.5182, 0.1182, 0.4048, 0.0111, 0.2754, 0.1509, 0.0785],
          [0.5646, 0.5618, 0.0799, 0.8472, 0.8155, 0.0692, 0.0155],
          [0.1041, 0.2831, 0.9905, 0.5570, 0.4696, 0.5949, 0.6720]],

         [[0.2689, 0.1559, 0.3610, 0.9099, 0.0967, 0.6367, 0.9011],
          [0.8516, 0.7169, 0.7008, 0.7046, 0.2938, 0.7228, 0.4940],
          [0.0229, 0.3864, 0.5015, 0.7872, 0.9237, 0.8596, 0.1773],
          [0.4732, 0.1491, 0.9773, 0.0663, 0.6229, 0.2485, 0.6849],
          [0.9634, 0.5058, 0.4425, 0.5908, 0.0383, 0.6257, 0.2074],
          [0.2559, 0.3446, 0.8127, 0.1353, 0.5073, 0.6480, 0.5414],
          [0.4851, 0.0487, 0.6537, 0.9839, 0.3720, 0.2927, 0.2682]]]],
       dtype=torch.float64)
Random tensor with preset seed: tensor([[[[0.4963, 0.7682, 0.0885, 0.1320,
0.3074, 0.6341, 0.4901],
          [0.8964, 0.4556, 0.6323, 0.3489, 0.4017, 0.0223, 0.1689],
          [0.2939, 0.5185, 0.6977, 0.8000, 0.1610, 0.2823, 0.6816],
          [0.9152, 0.3971, 0.8742, 0.4194, 0.5529, 0.9527, 0.0362],
          [0.1852, 0.3734, 0.3051, 0.9320, 0.1759, 0.2698, 0.1507],
          [0.0317, 0.2081, 0.9298, 0.7231, 0.7423, 0.5263, 0.2437],
          [0.5846, 0.0332, 0.1387, 0.2422, 0.8155, 0.7932, 0.2783]],

         [[0.4820, 0.8198, 0.9971, 0.6984, 0.5675, 0.8352, 0.2056],
          [0.5932, 0.1123, 0.1535, 0.2417, 0.7262, 0.7011, 0.2038],
          [0.6511, 0.7745, 0.4369, 0.5191, 0.6159, 0.8102, 0.9801],
          [0.1147, 0.3168, 0.6965, 0.9143, 0.9351, 0.9412, 0.5995],
          [0.0652, 0.5460, 0.1872, 0.0340, 0.9442, 0.8802, 0.0012],
          [0.5936, 0.4158, 0.4177, 0.2711, 0.6923, 0.2038, 0.6833],
          [0.7529, 0.8579, 0.6870, 0.0051, 0.1757, 0.7497, 0.6047]],

         [[0.1100, 0.2121, 0.9704, 0.8369, 0.2820, 0.3742, 0.0237],
          [0.4910, 0.1235, 0.1143, 0.4725, 0.5751, 0.2952, 0.7967],
```

```
         [0.1957, 0.9537, 0.8426, 0.0784, 0.3756, 0.5226, 0.5730],
         [0.6186, 0.6962, 0.5300, 0.2560, 0.7366, 0.0204, 0.2036],
         [0.3748, 0.2564, 0.3251, 0.0902, 0.3936, 0.6069, 0.1743],
         [0.4743, 0.8579, 0.4486, 0.5139, 0.4569, 0.6012, 0.8179],
         [0.9736, 0.8175, 0.9747, 0.4638, 0.0508, 0.2630, 0.8405]]]])
tensor([[[[-5.0000, -4.8630, -4.7260, -4.5890, -4.4521, -4.3151, -4.1781],
         [-4.0411, -3.9041, -3.7671, -3.6301, -3.4932, -3.3562, -3.2192],
         [-3.0822, -2.9452, -2.8082, -2.6712, -2.5342, -2.3973, -2.2603],
         [-2.1233, -1.9863, -1.8493, -1.7123, -1.5753, -1.4384, -1.3014],
         [-1.1644, -1.0274, -0.8904, -0.7534, -0.6164, -0.4795, -0.3425],
         [-0.2055, -0.0685,  0.0685,  0.2055,  0.3425,  0.4795,  0.6164],
         [ 0.7534,  0.8904,  1.0274,  1.1644,  1.3014,  1.4384,  1.575
3]],

        [[ 1.7123,  1.8493,  1.9863,  2.1233,  2.2603,  2.3973,  2.5342],
         [ 2.6712,  2.8082,  2.9452,  3.0822,  3.2192,  3.3562,  3.4932],
         [ 3.6301,  3.7671,  3.9041,  4.0411,  4.1781,  4.3151,  4.4521],
         [ 4.5890,  4.7260,  4.8630,  5.0000,  5.1370,  5.2740,  5.4110],
         [ 5.5479,  5.6849,  5.8219,  5.9589,  6.0959,  6.2329,  6.3699],
         [ 6.5068,  6.6438,  6.7808,  6.9178,  7.0548,  7.1918,  7.3288],
         [ 7.4658,  7.6027,  7.7397,  7.8767,  8.0137,  8.1507,  8.287
7]],

        [[ 8.4247,  8.5616,  8.6986,  8.8356,  8.9726,  9.1096,  9.2466],
         [ 9.3836,  9.5205,  9.6575,  9.7945,  9.9315, 10.0685, 10.2055],
         [10.3425, 10.4795, 10.6164, 10.7534, 10.8904, 11.0274, 11.1644],
         [11.3014, 11.4384, 11.5753, 11.7123, 11.8493, 11.9863, 12.1233],
         [12.2603, 12.3973, 12.5342, 12.6712, 12.8082, 12.9452, 13.0822],
         [13.2192, 13.3562, 13.4932, 13.6301, 13.7671, 13.9041, 14.0411],
         [14.1781, 14.3151, 14.4521, 14.5890, 14.7260, 14.8630, 15.000
0]]]])
Zeros tensor: tensor([[[[0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0.],
```

```
[0., 0., 0., 0., 0., 0., 0.]]]])
```

2. PyTorch allows applying GPU for computations. Check if gpu (CUDA) is available and then use it as `device`, else use `'cpu'`. Then, move one of your tensors to selected device.

In [2]:
```
# gpu jest lepsze do wykonywania skomplikowanych obliczen dlatego chcieli
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# przenosimy nasz tensor na urządzenie które będzie korzystac z wczesniej
random_tensor = random_tensor.to(device)
print(f"Tensor on {device}: {random_tensor}")
```

```
Using device: cpu
Tensor on cpu: tensor([[[[7.0080e-01, 8.6437e-01, 7.5487e-01, 7.1401e-01,
4.8101e-01,
            4.3614e-01, 6.4462e-01],
           [9.5264e-01, 1.2725e-01, 2.3432e-01, 7.9959e-01, 5.4008e-01,
            8.4314e-02, 2.6639e-01],
           [7.7565e-02, 1.8192e-01, 6.4973e-01, 5.9840e-01, 5.7920e-01,
            5.0073e-02, 3.5495e-01],
           [4.1902e-05, 5.5996e-01, 9.1004e-01, 2.8843e-01, 2.7448e-01,
            1.8270e-01, 5.3634e-01],
           [2.0827e-01, 9.1346e-01, 3.7689e-01, 7.4214e-01, 3.2981e-01,
            2.3370e-01, 7.7565e-01],
           [7.7355e-01, 9.7310e-01, 6.6742e-01, 5.7325e-01, 6.4083e-01,
            7.8802e-01, 2.8502e-01],
           [4.1989e-01, 8.2637e-01, 3.2067e-01, 1.5849e-01, 6.3668e-01,
            8.8936e-01, 7.9497e-01]],

          [[6.8548e-01, 7.9268e-01, 5.1928e-01, 7.5358e-01, 9.9550e-01,
            8.8905e-01, 4.2463e-01],
           [1.7213e-01, 8.9372e-01, 1.1398e-01, 1.0622e-01, 8.0143e-01,
            8.8916e-01, 9.6725e-02],
           [4.1713e-01, 4.8904e-01, 1.7239e-02, 2.6127e-01, 7.6730e-01,
            4.9176e-01, 3.8155e-01],
           [2.3672e-02, 2.2816e-01, 2.9710e-01, 7.7397e-01, 2.7208e-01,
            8.5644e-01, 5.5000e-01],
           [3.6513e-01, 3.6393e-01, 1.2503e-01, 9.5675e-01, 1.7554e-01,
            6.4664e-01, 5.2367e-01],
           [4.8684e-01, 5.2972e-01, 5.4134e-01, 4.2196e-01, 7.8296e-01,
            2.1832e-01, 8.7881e-01],
           [2.8872e-01, 1.6445e-01, 4.0563e-01, 4.6850e-01, 2.4356e-01,
            4.7162e-01, 4.3645e-01]],

          [[9.6860e-01, 3.9752e-01, 2.4091e-01, 3.7410e-01, 2.2564e-01,
            5.2210e-01, 8.2195e-01],
           [8.2360e-01, 6.2337e-01, 1.4158e-01, 3.2934e-01, 1.1538e-01,
            1.1906e-01, 7.0142e-01],
           [3.6576e-01, 4.5279e-01, 9.2385e-02, 5.0944e-01, 5.9279e-01,
            9.1178e-01, 9.3597e-01],
           [2.7988e-01, 2.5921e-02, 7.8537e-01, 9.3055e-01, 8.2178e-01,
            9.0187e-01, 3.3890e-01],
           [1.6154e-01, 2.4392e-01, 4.9567e-01, 6.8660e-01, 8.9719e-02,
            6.1184e-01, 8.4940e-01],
           [9.7104e-01, 8.9192e-01, 1.7125e-01, 4.7201e-02, 7.9129e-01,
            3.6652e-01, 6.1345e-01],
           [1.6851e-01, 6.0670e-02, 3.3917e-01, 6.6320e-01, 5.6449e-01,
            2.7340e-01, 3.8840e-01]]]])
```

3. To train a network, we need a dataset.

Please download `MNIST` dataset with `torchvision.dataset`.

For any kind of ML task, validation or testing is required.

So, create train and test datasets.

For train dataset apply also augmentation transforms, crop, translation and rotation.

For both apply ToTensor.

Next, pack datasets into `DataLoader` s with batch size of 64. Use variables with names: `train_loader` and `test_loader` .

Next display sizes of datasets, shapes of elements and display a few images and their labels.

Finally, compare the number of objects in each class in both datasets.

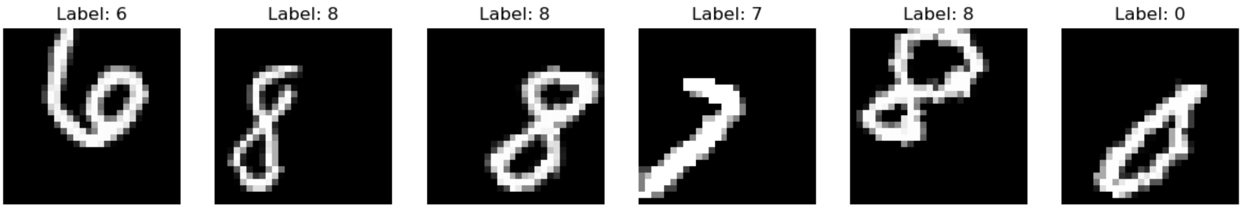
In [3]:
```
import torch
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor, Compose, RandomCrop, RandomA
import matplotlib.pyplot as plt

# Compose łączy kilka transformacji w jedną sekwencję
# RandomCrop przycina obraz do rozmiaru 28x28 pikseli z losowym paddingie
# RandomAffine losowa rotacja obrazu o maks 15 stopni i translacja czyli
# ToTensor konwertuje na tensor bo tak chce torch
train_transform = Compose([
    RandomCrop(28, padding=4),
    RandomAffine(degrees=15, translate=(0.1, 0.1)),
    ToTensor()
])

# Tu nie sotsujemy augmentacji zeby wyniki byly porównywalne
test_transform = Compose([
    ToTensor()
])

try:
    train_dataset = datasets.MNIST(
        root="./data", train=True, download=True, transform=train_transfo
    )
    test_dataset = datasets.MNIST(
        root="./data", train=False, download=True, transform=test_transfo
    )
except Exception as e:
    print(f"Failed to download MNIST dataset: {e}")

# DataLoader klasa ułatwia iteracji po danych w partiach
# batch_size liczba probek w jednej partii
# shuffle losowe mieszanie danych w kazdej epoce
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

print(f"Train dataset size: {len(train_dataset)}")
```

```
print(f"Test dataset size: {len(test_dataset)}")
```

```
Train dataset size: 60000
Test dataset size: 10000
```

In [4]:
```
dataiter = iter(train_loader)
images, labels = next(dataiter)
print(f"Batch shape: {images.size()}") # [64, 1, 28, 28] - 64 liczba obra
print(f"Labels shape: {labels.size()}") # 64 etykiety

fig, axes = plt.subplots(1, 6, figsize=(15, 4))
for i in range(6):
    ax = axes[i]
    ax.imshow(images[i].numpy().squeeze(), cmap="gray")
    ax.set_title(f"Label: {labels[i].item()}")
    ax.axis("off")
plt.show()

train_labels = [label for _, label in train_loader.dataset]
test_labels = [label for _, label in test_loader.dataset]

# zlicza liczbę wystapien kazdej klasy
train_class_counts = torch.tensor(train_labels).bincount()
test_class_counts = torch.tensor(test_labels).bincount()

print(f"Train class counts: {train_class_counts}")
print(f"Test class counts: {test_class_counts}")
```

```
Batch shape: torch.Size([64, 1, 28, 28])
Labels shape: torch.Size([64])
```




```
Train class counts: tensor([5923, 6742, 5958, 6131, 5842, 5421, 5918, 626
5, 5851, 5949])
Test class counts: tensor([ 980, 1135, 1032, 1010,  982,  892,  958, 1028,
974, 1009])
```

In [20]:
```
import matplotlib.pyplot as plt


def draw_loss_test(epochs, history):
    loss_train = history["loss_train"]
    loss_test = history["loss_test"]
    acc_train = history["acc_train"]
    acc_test = history["acc_test"]

    loss_train_shape = len(loss_train)
    loss_test_shape = len(loss_test)
    acc_train_shape = len(acc_train)
    acc_test_shape = len(acc_test)
```

```
    if (
        loss_train_shape != loss_test_shape
        or acc_train_shape != acc_test_shape
    ):
        raise ValueError(
            f"Different number of epochs for train and test loss: {loss_t
        )

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history["loss_train"], label="Train loss")
    plt.plot(epochs, history["loss_test"], label="Test loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history["acc_train"], label="Train accuracy")
    plt.plot(epochs, history["acc_test"], label="Test accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()

    plt.show()
```

5. We have our dataset ready, let's create model for classification task.

Please define class `MLP` as Multi Layer Perceptron with two hidden fully connected layers with bias.

Class must inherit from `torch.nn.Module`.

Apply following configuration:

- first layer with 512 neurons,
- second layer with 512 neurons,
- output layer adjusted to the size of classification problem.

For `__init__` method add parameters: input_shape and output_size.

Don't forget about nonlinearities!

For hidden layers you can use `ReLU` module from `torch.nn`.

For output apply softmax function.

Define layer-by-layer network processing in `forward` method with argument as a network.

Help: input tensor - batch of images with shape (batch_size, channels, height, width) - (channels = 1 for gray scale images).

Instantiate model as `net` object.

Layers:

- `torch.nn.Sequential`
  - layer allows for forward pass through component layers:
    ```
    t_in: Tensor
    t_out: Tensor = t_in
    for L in layers:
        t_out = L(t_out)
    return t_out
    ```
- `torch.nn.Flatten`
  - layer makes input tensor flattened:
  - (bs, CH, H, W) -> (bs, CH * H * W)
- `torch.nn.Linear(ch_in, ch_out, bias)`
  - 'classical' neural network layer - fully connected
  - ch_in is a number of input channels
  - ch_out is a number of output channels / number of neurons in layer
  - bias - whether to use bias parameter
  - for Linear layers it is recommended to use flatten layer before, when input has more than 2 dimensions
  - operation implemented by this layer is a vector / matrix multiplication
    - `y = W x v` or `y = W x v + b`
    - W has a shape [ch_out, ch_in]
    - v has a shape [ch_in]
    - b has a shape [ch_in]
    - y has a shape [ch_out]
- `torch.nn.ReLU`
  - layer applies ReLU function on input tensor
- `torch.nn.Softmax(dim)`
  - layer applies softmax function on input tensor
  - `dim` - dimension over which function is calculated

For the formulas of activation function go to torch documentation.

In [6]:
```
import torch.nn as nn
```

```
# klasa definiująca wielowarstwowy perceptron
class MLP(nn.Module):

    def __init__(self, input_shape, output_size) -> None:
        super(MLP, self).__init__()
        # Sequential to kontener ktory pozwala na zdefiniowanie listy war
        self.model = nn.Sequential(
            # Flatten warstwa splaszczajaca tensor z (batch_size, channel
            # np. obrazki 28x28 maja (batch_size, 1, 28, 28) i splaszczy
            # 784 to liczba cech
            nn.Flatten(),
            # Linear y = Wx + b (W macierz wag, x dane wejsciowe, b wekto
            # in_features - liczba wejsciowych cech, out_features - liczb
            nn.Linear(input_shape[0] * input_shape[1] * input_shape[2], 5
            # ReLU f(x) = max(0, x)
            # warstwa aktywacji - wprowadza nieliniowosc modelu (bez tego
            # pozwala sieci na uczenie sie bardziej zlozonych funkcji
            nn.ReLU(),
            nn.Linear(512, 512, bias=True),
            nn.ReLU(),
            # ostatnia warstwa przeksztalca dane wyjsciowe na dane o rozm
            nn.Linear(512, output_size, bias=True),
            # Softmax softmax(x_i) = exp(x_i) / sum(exp(x_j))
            # warstwa aktywacji - przeksztalca wyjscia w pradopodobienstw
            # dim=1 oznacza ze softmax jest stosowany wzdluz osi klas  (d
            # czesto nie uzywamy tego gdy korzystamy z CrossEntroyLoss bo
            nn.Softmax(dim=1)
        )

    # dfiniuje przeplyw danych przez siec, x: tensor wejsciowy (batch_siz
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.model(x)

# obrazek w skali szarosci o rozmairze 28x28 pikseli
input_shape = (1, 28, 28)
# liczb klas do klasyfikacji np. 10 do liczb od 0 do 9
output_size = 10

net = MLP(input_shape, output_size)
net = net.to(device)

net2 = MLP(input_shape, output_size)
net2 = net2.to(device)
```

6. To train network we need to know 'how good or bad' results it gives. Please, instantiate `torch.nn.CrossEntropyLoss` as `loss_fcn`.

In [7]:
```
# informuje jak dobrze model przewiduje prawdziwe klasy
loss_fn = nn.CrossEntropyLoss()
```

7. To score network define accuracy metric. For network output you need to decide

what is the final network answer. For clasification we can assume, that the final answer is the class with highest probability (`argmax`).

`torch.no_grad()` prevents gradient requirement for computations inside method.

In [8]:
```
from abc import ABC, abstractmethod
from typing import Any


class BaseMetric(ABC):

    @abstractmethod
    def __call__(self, y_pred, y_ref) -> Any:
        raise NotImplementedError()


class AccuracyMetric(BaseMetric):

    def __init__(self) -> None:
        pass

    @torch.no_grad()
    def __call__(self, y_pred: torch.Tensor, y_ref: torch.Tensor) -> torc
        """
        :param y_pred: tensor of shape (batch_size, num_of_classes) type
        :param y_ref: tensor with shape (batch_size,) and type Long
        :return: scalar tensor with accuracy metric for batch
        """
        predicted_classes = torch.argmax(y_pred, dim=1) # zwroci indeks k
        correct_predictions = (predicted_classes == y_ref).float() # poró
        score: torch.Tensor = correct_predictions.mean() # srednia z popr

        return score


metric = AccuracyMetric()
```

8. To change network parameters, we need an optimizer object. Instantiate `torch.optim.SGD` (with `net` work parameters) as `optimizer`. Use learning rate = 0.001.

In [9]:
```
# Stochastic Gradient Descent algorytm do optymalizacji (minimzlizacji fu
# parameters to wagi i biasy, lr kork o jaki będą aktualizaowane parametr
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)
optimizer2 = torch.optim.SGD(net2.parameters(), lr=0.01)
```

9. Now define training / testing function:

In [24]:
```
def train_or_test(
```

```
    model,  # sieć neuronowa która trenujemy lub testujemy
    data_generator,  # generator danych, ktory dostarcza dane wejsciowe i
    criterion,  # funkcja straty, ktora oblicza roznice miedzy predykcjam
    metric: BaseMetric,  # funkcja metryki, ktora oblicza dokladnosc pred
    mode: str = "test",
    optimizer: torch.optim.Optimizer = None,  # optymalizator - aktualiza
    update_period: int = None,  # Okres aktualizacji, określa co ile batc
    device=torch.device("cpu"),
) -> Tuple[torch.nn.Module, float, float]:

    # change model mode to train or test
    if mode == 'train':
        model.train()
    elif mode == 'test':
        model.eval()
    else:
        raise RuntimeError(f"Unsupported mode: {mode}. Use 'train' or 'te

    # move model to device
    model.to(device)

    # reset model parameters' gradients with optimizer
    if mode == 'train':
        optimizer.zero_grad()

    total_loss = 0.0 # suma strat dla wszystkich batchy
    total_accuracy = 0.0 # suma dokladnosci dla wszystkich batchy
    samples_num = 0 # liczba probek przetworzonych przez model

    for i, (X, y) in enumerate(tqdm.tqdm(data_generator)):
        # convert tensors to device
        X, y = X.to(device), y.to(device)

        # process by network
        y_pred = model(X)

        # calculate loss
        loss = criterion(y_pred, y)

        # designate gradient based on loss
        if mode == 'train':
            loss.backward()

        if mode == 'train' and (i+1) % update_period == 0:
            # update parameters with optimizer
            optimizer.step()
            optimizer.zero_grad()

        # calculate accuracy
        accuracy = metric(y_pred, y)

        total_loss += loss.item() * y_pred.shape[0] # y_pred.shape[0] to
        total_accuracy += accuracy.item() * y_pred.shape[0]
```

```
        samples_num += y_pred.shape[0]

    if samples_num == 0:
        return model, 0.0, 0.0

    return model, total_loss / samples_num, total_accuracy / samples_num
```

In [ ]:
```
net_preview, loss_preview, acc_preview = train_or_test(
    net,
    train_loader,
    loss_fn, metric,
    'train',
    update_period=5,
    optimizer=optimizer,
    device=device,
)
```

10. Prepare training loop function (over epochs):

- adjust max number of epochs to achieve satisfactory results,
- **EXTENSION EXERCISE** implement stopping training when accuracy exceeds certain value.

In [25]:
```
def training(model,
             train_loader,
             test_loader,
             loss_fn,
             metric,
             optimizer,
             update_period,
             epoch_max,
             device,
             early_stopping_accuracy=None):
    loss_train = []
    loss_test = []
    acc_train = []
    acc_test = []

    for e in range(epoch_max):
        print(f"\nEpoka: {e + 1}/{epoch_max}")

        model.train()
        total_loss_train = 0.0
        total_acc_train = 0.0
        samples_num_train = 0

        for i, (X, y) in enumerate(tqdm.tqdm(train_loader, colour="red",
            X, y = X.to(device), y.to(device)

            y_pred = model(X)
```

```
            loss = loss_fn(y_pred, y)
            loss.backward()

            if (i + 1) % update_period == 0:
                optimizer.step()
                optimizer.zero_grad()

            accuracy = metric(y_pred, y)
            batch_size = y.size(0)
            total_loss_train += loss.item() * batch_size
            total_acc_train += accuracy.item() * batch_size
            samples_num_train += batch_size

        epoch_loss_train = total_loss_train / samples_num_train
        epoch_acc_train = total_acc_train / samples_num_train
        loss_train.append(epoch_loss_train)
        acc_train.append(epoch_acc_train)

        print(
            f"Train loss: {epoch_loss_train:.4f}\nTrain accuracy: {epoch_
        )

        model.eval()
        total_loss_test = 0.0
        total_acc_test = 0.0
        samples_num_test = 0

        with torch.no_grad():
            for X, y in tqdm.tqdm(test_loader, colour='green', smoothing=
                X, y = X.to(device), y.to(device)

                y_pred = model(X)

                loss = loss_fn(y_pred, y)
                accuracy = metric(y_pred, y)

                batch_size = y.size(0)
                total_loss_test += loss.item() * batch_size
                total_acc_test += accuracy.item() * batch_size
                samples_num_test += batch_size

        epoch_loss_test = total_loss_test / samples_num_test
        epoch_acc_test = total_acc_test / samples_num_test
        loss_test.append(epoch_loss_test)
        acc_test.append(epoch_acc_test)

        print(f"Test loss: {epoch_loss_test:.4f}\nTest accuracy: {epoch_a

        if (early_stopping_accuracy and epoch_acc_train >= early_stopping
            print(f"Training accuracy of {epoch_acc_train:.4f} achieved,
            break

    return model, {'loss_train': loss_train,
```

```
                    'acc_train': acc_train,
                    'loss_test': loss_test,
                    'acc_test': acc_test}
```

In [10]:
```
def train_one_epoch(
    model, train_loader, loss_fn, metric, optimizer, update_period, devic
):
    model.train()
    total_loss_train = 0.0
    total_acc_train = 0.0
    samples_num_train = 0

    with tqdm.tqdm(
        train_loader, colour="red", ncols=100
    ) as t:
        for i, (X, y) in enumerate(t):
            X, y = X.to(device), y.to(device)

            y_pred = model(X)
            loss = loss_fn(y_pred, y)
            loss.backward()

            if (i + 1) % update_period == 0:
                optimizer.step()
                optimizer.zero_grad()

            accuracy = metric(y_pred, y)
            batch_size = y.size(0)
            total_loss_train += loss.item() * batch_size
            total_acc_train += accuracy.item() * batch_size
            samples_num_train += batch_size

            current_loss = total_loss_train / samples_num_train
            current_acc = total_acc_train / samples_num_train

            t.set_postfix(loss=f'{current_loss:.4f}', accuracy=f'{current

    epoch_loss_train = total_loss_train / samples_num_train
    epoch_acc_train = total_acc_train / samples_num_train

    return epoch_loss_train, epoch_acc_train


def test_one_epoch(model, test_loader, loss_fn, metric, device):
    model.eval()
    total_loss_test = 0.0
    total_acc_test = 0.0
    samples_num_test = 0

    with torch.no_grad():
        with tqdm.tqdm(
            test_loader, colour="green", ncols=100
        ) as t:
```

```
            for X, y in t:
                X, y = X.to(device), y.to(device)

                y_pred = model(X)
                loss = loss_fn(y_pred, y)
                accuracy = metric(y_pred, y)

                batch_size = y.size(0)
                total_loss_test += loss.item() * batch_size
                total_acc_test += accuracy.item() * batch_size
                samples_num_test += batch_size

                current_loss = total_loss_test / samples_num_test
                current_acc = total_acc_test / samples_num_test

                t.set_postfix(loss=f'{current_loss:.4f}', accuracy=f'{cur

    epoch_loss_test = total_loss_test / samples_num_test
    epoch_acc_test = total_acc_test / samples_num_test

    return epoch_loss_test, epoch_acc_test


def test_or_train(
    model,
    train_loader,
    test_loader,
    loss_fn,
    metric,
    optimizer,
    update_period,
    epoch_max,
    device,
    mode="train",
    early_stopping_accuracy=None,
):
    loss_train = []
    loss_test = []
    acc_train = []
    acc_test = []

    for e in range(epoch_max):
        print(f"Epoch: {e + 1}/{epoch_max}")

        if mode in ["train", "both"]:
            epoch_loss_train, epoch_acc_train = train_one_epoch(
                model,
                train_loader,
                loss_fn,
                metric,
                optimizer,
                update_period,
                device,
```

```
            )
            loss_train.append(epoch_loss_train)
            acc_train.append(epoch_acc_train)

        if mode in ["test", "both"]:
            epoch_loss_test, epoch_acc_test = test_one_epoch(
                model, test_loader, loss_fn, metric, device
            )
            loss_test.append(epoch_loss_test)
            acc_test.append(epoch_acc_test)

        if (
            early_stopping_accuracy
            and epoch_acc_train >= early_stopping_accuracy
        ):
            print(
                f"Training accuracy of {epoch_acc_train:.4f} achieved, st
            )
            break

    return model, {
        "loss_train": loss_train,
        "acc_train": acc_train,
        "loss_test": loss_test,
        "acc_test": acc_test,
    }
```

11. Display training history.

In [13]:
```
net2, history2 = test_or_train(
    net2,
    train_loader,
    test_loader,
    loss_fn,
    metric,
    optimizer2,
    update_period=5,
    epoch_max=100,
    device=device,
    mode="both",
    early_stopping_accuracy=0.98,
)
```

```
Epoch: 1/100
100%|██████████| 938/938 [00:04<00:00, 198.24it/s, acc
uracy=0.1132, loss=2.3024]
100%|██████████| 157/157 [00:00<00:00, 500.36it/s, acc
uracy=0.1276, loss=2.3016]
Epoch: 2/100
```

```
100%|██████████| 938/938 [00:04<00:00, 202.14it/s, acc
uracy=0.1279, loss=2.3018]
100%|██████████| 157/157 [00:00<00:00, 478.90it/s, acc
uracy=0.1411, loss=2.3004]
Epoch: 3/100
100%|██████████| 938/938 [00:04<00:00, 203.05it/s, acc
uracy=0.1262, loss=2.3011]
100%|██████████| 157/157 [00:00<00:00, 510.15it/s, acc
uracy=0.1157, loss=2.2990]
Epoch: 4/100
100%|██████████| 938/938 [00:04<00:00, 202.04it/s, acc
uracy=0.1097, loss=2.3002]
100%|██████████| 157/157 [00:00<00:00, 501.29it/s, acc
uracy=0.0997, loss=2.2971]
Epoch: 5/100
100%|██████████| 938/938 [00:04<00:00, 200.40it/s, acc
uracy=0.0999, loss=2.2990]
100%|██████████| 157/157 [00:00<00:00, 484.43it/s, acc
uracy=0.0980, loss=2.2940]
Epoch: 6/100
100%|██████████| 938/938 [00:04<00:00, 198.90it/s, acc
uracy=0.0987, loss=2.2967]
100%|██████████| 157/157 [00:00<00:00, 488.73it/s, acc
uracy=0.0980, loss=2.2872]
Epoch: 7/100
100%|██████████| 938/938 [00:04<00:00, 198.49it/s, acc
uracy=0.0987, loss=2.2917]
100%|██████████| 157/157 [00:00<00:00, 495.19it/s, acc
uracy=0.0980, loss=2.2727]
Epoch: 8/100
100%|██████████| 938/938 [00:04<00:00, 198.98it/s, acc
uracy=0.1048, loss=2.2844]
100%|██████████| 157/157 [00:00<00:00, 496.70it/s, acc
uracy=0.1703, loss=2.2576]
Epoch: 9/100
100%|██████████| 938/938 [00:04<00:00, 189.19it/s, acc
uracy=0.1441, loss=2.2771]
100%|██████████| 157/157 [00:00<00:00, 494.45it/s, acc
uracy=0.2138, loss=2.2443]
Epoch: 10/100
100%|██████████| 938/938 [00:04<00:00, 197.21it/s, acc
uracy=0.1703, loss=2.2698]
100%|██████████| 157/157 [00:00<00:00, 495.92it/s, acc
uracy=0.2413, loss=2.2335]
Epoch: 11/100
100%|██████████| 938/938 [00:04<00:00, 197.04it/s, acc
uracy=0.1862, loss=2.2632]
100%|██████████| 157/157 [00:00<00:00, 490.07it/s, acc
uracy=0.2526, loss=2.2246]
Epoch: 12/100
```

```
100%|██████████| 938/938 [00:04<00:00, 191.55it/s, accuracy=0.1989, loss=2.2553]
100%|██████████| 157/157 [00:00<00:00, 376.70it/s, accuracy=0.2598, loss=2.2138]
Epoch: 13/100
100%|██████████| 938/938 [00:04<00:00, 187.97it/s, accuracy=0.2107, loss=2.2469]
100%|██████████| 157/157 [00:00<00:00, 474.79it/s, accuracy=0.2620, loss=2.1975]
Epoch: 14/100
100%|██████████| 938/938 [00:04<00:00, 191.66it/s, accuracy=0.2213, loss=2.2339]
100%|██████████| 157/157 [00:00<00:00, 374.21it/s, accuracy=0.2765, loss=2.1702]
Epoch: 15/100
100%|██████████| 938/938 [00:04<00:00, 197.28it/s, accuracy=0.2364, loss=2.2171]
100%|██████████| 157/157 [00:00<00:00, 485.69it/s, accuracy=0.3091, loss=2.1412]
Epoch: 16/100
100%|██████████| 938/938 [00:05<00:00, 173.45it/s, accuracy=0.2620, loss=2.1958]
100%|██████████| 157/157 [00:00<00:00, 496.47it/s, accuracy=0.3618, loss=2.1098]
Epoch: 17/100
100%|██████████| 938/938 [00:05<00:00, 174.68it/s, accuracy=0.2980, loss=2.1740]
100%|██████████| 157/157 [00:00<00:00, 485.76it/s, accuracy=0.4212, loss=2.0757]
Epoch: 18/100
100%|██████████| 938/938 [00:05<00:00, 180.39it/s, accuracy=0.3351, loss=2.1497]
100%|██████████| 157/157 [00:00<00:00, 430.39it/s, accuracy=0.4899, loss=2.0368]
Epoch: 19/100
100%|██████████| 938/938 [00:04<00:00, 198.03it/s, accuracy=0.3612, loss=2.1275]
100%|██████████| 157/157 [00:00<00:00, 451.10it/s, accuracy=0.5266, loss=1.9997]
Epoch: 20/100
100%|██████████| 938/938 [00:04<00:00, 199.45it/s, accuracy=0.3999, loss=2.1028]
100%|██████████| 157/157 [00:00<00:00, 428.73it/s, accuracy=0.5783, loss=1.9622]
Epoch: 21/100
100%|██████████| 938/938 [00:04<00:00, 194.05it/s, accuracy=0.4158, loss=2.0836]
100%|██████████| 157/157 [00:00<00:00, 443.94it/s, accuracy=0.5879, loss=1.9383]
Epoch: 22/100
```

```
100%|██████████| 938/938 [00:05<00:00, 185.39it/s, accuracy=0.4271, loss=2.0657]
100%|██████████| 157/157 [00:00<00:00, 530.34it/s, accuracy=0.5999, loss=1.9165]
Epoch: 23/100
100%|██████████| 938/938 [00:05<00:00, 180.34it/s, accuracy=0.4388, loss=2.0502]
100%|██████████| 157/157 [00:00<00:00, 475.98it/s, accuracy=0.6068, loss=1.8986]
Epoch: 24/100
100%|██████████| 938/938 [00:04<00:00, 196.44it/s, accuracy=0.4434, loss=2.0401]
100%|██████████| 157/157 [00:00<00:00, 486.52it/s, accuracy=0.6091, loss=1.8889]
Epoch: 25/100
100%|██████████| 938/938 [00:04<00:00, 194.00it/s, accuracy=0.4518, loss=2.0298]
100%|██████████| 157/157 [00:00<00:00, 461.56it/s, accuracy=0.6118, loss=1.8812]
Epoch: 26/100
100%|██████████| 938/938 [00:04<00:00, 195.07it/s, accuracy=0.4587, loss=2.0206]
100%|██████████| 157/157 [00:00<00:00, 477.50it/s, accuracy=0.6147, loss=1.8743]
Epoch: 27/100
100%|██████████| 938/938 [00:04<00:00, 196.63it/s, accuracy=0.4667, loss=2.0118]
100%|██████████| 157/157 [00:00<00:00, 460.50it/s, accuracy=0.6169, loss=1.8663]
Epoch: 28/100
100%|██████████| 938/938 [00:04<00:00, 194.21it/s, accuracy=0.4717, loss=2.0050]
100%|██████████| 157/157 [00:00<00:00, 473.61it/s, accuracy=0.6262, loss=1.8528]
Epoch: 29/100
100%|██████████| 938/938 [00:04<00:00, 195.23it/s, accuracy=0.4817, loss=1.9969]
100%|██████████| 157/157 [00:00<00:00, 462.92it/s, accuracy=0.6718, loss=1.8284]
Epoch: 30/100
100%|██████████| 938/938 [00:05<00:00, 183.27it/s, accuracy=0.4978, loss=1.9846]
100%|██████████| 157/157 [00:00<00:00, 516.37it/s, accuracy=0.6953, loss=1.8108]
Epoch: 31/100
100%|██████████| 938/938 [00:05<00:00, 185.92it/s, accuracy=0.5093, loss=1.9727]
100%|██████████| 157/157 [00:00<00:00, 391.81it/s, accuracy=0.7032, loss=1.7988]
Epoch: 32/100
```

```
100%|██████████| 938/938 [00:05<00:00, 183.22it/s, accuracy=0.5202, loss=1.9617]
100%|██████████| 157/157 [00:00<00:00, 464.33it/s, accuracy=0.7100, loss=1.7882]
Epoch: 33/100
100%|██████████| 938/938 [00:04<00:00, 194.47it/s, accuracy=0.5333, loss=1.9494]
100%|██████████| 157/157 [00:00<00:00, 410.10it/s, accuracy=0.7089, loss=1.7856]
Epoch: 34/100
100%|██████████| 938/938 [00:05<00:00, 187.35it/s, accuracy=0.5436, loss=1.9383]
100%|██████████| 157/157 [00:00<00:00, 475.99it/s, accuracy=0.7160, loss=1.7776]
Epoch: 35/100
100%|██████████| 938/938 [00:05<00:00, 185.57it/s, accuracy=0.5521, loss=1.9302]
100%|██████████| 157/157 [00:00<00:00, 492.62it/s, accuracy=0.7181, loss=1.7746]
Epoch: 36/100
100%|██████████| 938/938 [00:05<00:00, 181.71it/s, accuracy=0.5609, loss=1.9198]
100%|██████████| 157/157 [00:00<00:00, 426.56it/s, accuracy=0.7189, loss=1.7705]
Epoch: 37/100
100%|██████████| 938/938 [00:04<00:00, 191.37it/s, accuracy=0.5689, loss=1.9126]
100%|██████████| 157/157 [00:00<00:00, 442.01it/s, accuracy=0.7217, loss=1.7678]
Epoch: 38/100
100%|██████████| 938/938 [00:04<00:00, 195.17it/s, accuracy=0.5777, loss=1.9032]
100%|██████████| 157/157 [00:00<00:00, 439.42it/s, accuracy=0.7264, loss=1.7619]
Epoch: 39/100
100%|██████████| 938/938 [00:04<00:00, 196.53it/s, accuracy=0.5819, loss=1.8977]
100%|██████████| 157/157 [00:00<00:00, 442.62it/s, accuracy=0.7300, loss=1.7579]
Epoch: 40/100
100%|██████████| 938/938 [00:04<00:00, 194.06it/s, accuracy=0.5947, loss=1.8860]
100%|██████████| 157/157 [00:00<00:00, 444.47it/s, accuracy=0.7351, loss=1.7533]
Epoch: 41/100
100%|██████████| 938/938 [00:04<00:00, 194.97it/s, accuracy=0.6039, loss=1.8757]
100%|██████████| 157/157 [00:00<00:00, 472.68it/s, accuracy=0.7398, loss=1.7473]
Epoch: 42/100
```

```
100%|██████████| 938/938 [00:04<00:00, 191.76it/s, accuracy=0.6109, loss=1.8685]
100%|██████████| 157/157 [00:00<00:00, 463.34it/s, accuracy=0.7371, loss=1.7479]
Epoch: 43/100
100%|██████████| 938/938 [00:04<00:00, 191.56it/s, accuracy=0.6221, loss=1.8589]
100%|██████████| 157/157 [00:00<00:00, 471.25it/s, accuracy=0.7438, loss=1.7419]
Epoch: 44/100
100%|██████████| 938/938 [00:04<00:00, 192.71it/s, accuracy=0.6294, loss=1.8499]
100%|██████████| 157/157 [00:00<00:00, 456.62it/s, accuracy=0.7471, loss=1.7305]
Epoch: 45/100
100%|██████████| 938/938 [00:04<00:00, 193.38it/s, accuracy=0.6612, loss=1.8335]
100%|██████████| 157/157 [00:00<00:00, 461.05it/s, accuracy=0.8135, loss=1.7036]
Epoch: 46/100
100%|██████████| 938/938 [00:04<00:00, 194.82it/s, accuracy=0.6784, loss=1.8189]
100%|██████████| 157/157 [00:00<00:00, 470.18it/s, accuracy=0.8199, loss=1.6927]
Epoch: 47/100
100%|██████████| 938/938 [00:04<00:00, 192.85it/s, accuracy=0.6852, loss=1.8095]
100%|██████████| 157/157 [00:00<00:00, 469.21it/s, accuracy=0.8174, loss=1.6875]
Epoch: 48/100
100%|██████████| 938/938 [00:04<00:00, 192.17it/s, accuracy=0.6968, loss=1.7971]
100%|██████████| 157/157 [00:00<00:00, 459.38it/s, accuracy=0.8216, loss=1.6799]
Epoch: 49/100
100%|██████████| 938/938 [00:05<00:00, 183.41it/s, accuracy=0.7056, loss=1.7881]
100%|██████████| 157/157 [00:00<00:00, 435.21it/s, accuracy=0.8267, loss=1.6721]
Epoch: 50/100
100%|██████████| 938/938 [00:04<00:00, 194.99it/s, accuracy=0.7096, loss=1.7818]
100%|██████████| 157/157 [00:00<00:00, 436.39it/s, accuracy=0.8298, loss=1.6679]
Epoch: 51/100
100%|██████████| 938/938 [00:04<00:00, 195.21it/s, accuracy=0.7181, loss=1.7728]
100%|██████████| 157/157 [00:00<00:00, 428.96it/s, accuracy=0.8304, loss=1.6629]
Epoch: 52/100
```

```
100%|██████████| 938/938 [00:04<00:00, 194.91it/s, accuracy=0.7228, loss=1.7652]
100%|██████████| 157/157 [00:00<00:00, 439.41it/s, accuracy=0.8328, loss=1.6579]
Epoch: 53/100
100%|██████████| 938/938 [00:04<00:00, 194.73it/s, accuracy=0.7285, loss=1.7607]
100%|██████████| 157/157 [00:00<00:00, 427.10it/s, accuracy=0.8369, loss=1.6539]
Epoch: 54/100
100%|██████████| 938/938 [00:04<00:00, 191.41it/s, accuracy=0.7333, loss=1.7529]
100%|██████████| 157/157 [00:00<00:00, 470.92it/s, accuracy=0.8331, loss=1.6543]
Epoch: 55/100
100%|██████████| 938/938 [00:04<00:00, 192.86it/s, accuracy=0.7371, loss=1.7488]
100%|██████████| 157/157 [00:00<00:00, 472.80it/s, accuracy=0.8373, loss=1.6493]
Epoch: 56/100
100%|██████████| 938/938 [00:04<00:00, 193.67it/s, accuracy=0.7403, loss=1.7446]
100%|██████████| 157/157 [00:00<00:00, 476.44it/s, accuracy=0.8382, loss=1.6452]
Epoch: 57/100
100%|██████████| 938/938 [00:04<00:00, 194.03it/s, accuracy=0.7464, loss=1.7382]
100%|██████████| 157/157 [00:00<00:00, 464.09it/s, accuracy=0.8420, loss=1.6401]
Epoch: 58/100
100%|██████████| 938/938 [00:04<00:00, 192.87it/s, accuracy=0.7493, loss=1.7341]
100%|██████████| 157/157 [00:00<00:00, 460.41it/s, accuracy=0.8429, loss=1.6396]
Epoch: 59/100
100%|██████████| 938/938 [00:04<00:00, 192.57it/s, accuracy=0.7519, loss=1.7310]
100%|██████████| 157/157 [00:00<00:00, 469.92it/s, accuracy=0.8433, loss=1.6384]
Epoch: 60/100
100%|██████████| 938/938 [00:04<00:00, 194.03it/s, accuracy=0.7560, loss=1.7259]
100%|██████████| 157/157 [00:00<00:00, 469.44it/s, accuracy=0.8466, loss=1.6348]
Epoch: 61/100
100%|██████████| 938/938 [00:04<00:00, 192.99it/s, accuracy=0.7575, loss=1.7232]
100%|██████████| 157/157 [00:00<00:00, 471.72it/s, accuracy=0.8460, loss=1.6335]
Epoch: 62/100
```

```
100%|██████████| 938/938 [00:04<00:00, 193.37it/s, accuracy=0.7619, loss=1.7184]
100%|██████████| 157/157 [00:00<00:00, 463.57it/s, accuracy=0.8462, loss=1.6314]
Epoch: 63/100
100%|██████████| 938/938 [00:04<00:00, 192.89it/s, accuracy=0.7671, loss=1.7147]
100%|██████████| 157/157 [00:00<00:00, 466.09it/s, accuracy=0.8495, loss=1.6298]
Epoch: 64/100
100%|██████████| 938/938 [00:04<00:00, 192.37it/s, accuracy=0.7688, loss=1.7115]
100%|██████████| 157/157 [00:00<00:00, 462.45it/s, accuracy=0.8511, loss=1.6280]
Epoch: 65/100
100%|██████████| 938/938 [00:04<00:00, 194.37it/s, accuracy=0.7689, loss=1.7091]
100%|██████████| 157/157 [00:00<00:00, 469.05it/s, accuracy=0.8513, loss=1.6257]
Epoch: 66/100
100%|██████████| 938/938 [00:04<00:00, 192.46it/s, accuracy=0.7735, loss=1.7055]
100%|██████████| 157/157 [00:00<00:00, 474.32it/s, accuracy=0.8479, loss=1.6271]
Epoch: 67/100
100%|██████████| 938/938 [00:04<00:00, 193.48it/s, accuracy=0.7754, loss=1.7029]
100%|██████████| 157/157 [00:00<00:00, 467.79it/s, accuracy=0.8539, loss=1.6220]
Epoch: 68/100
100%|██████████| 938/938 [00:04<00:00, 194.06it/s, accuracy=0.7783, loss=1.7000]
100%|██████████| 157/157 [00:00<00:00, 474.30it/s, accuracy=0.8550, loss=1.6210]
Epoch: 69/100
100%|██████████| 938/938 [00:04<00:00, 194.16it/s, accuracy=0.7820, loss=1.6966]
100%|██████████| 157/157 [00:00<00:00, 465.09it/s, accuracy=0.8547, loss=1.6208]
Epoch: 70/100
100%|██████████| 938/938 [00:04<00:00, 192.72it/s, accuracy=0.7813, loss=1.6965]
100%|██████████| 157/157 [00:00<00:00, 469.79it/s, accuracy=0.8556, loss=1.6193]
Epoch: 71/100
100%|██████████| 938/938 [00:04<00:00, 192.85it/s, accuracy=0.7846, loss=1.6926]
100%|██████████| 157/157 [00:00<00:00, 449.96it/s, accuracy=0.8575, loss=1.6168]
Epoch: 72/100
```

```
100%|██████████| 938/938 [00:04<00:00, 193.14it/s, accuracy=0.7873, loss=1.6896]
100%|██████████| 157/157 [00:00<00:00, 467.66it/s, accuracy=0.8570, loss=1.6162]
Epoch: 73/100
100%|██████████| 938/938 [00:05<00:00, 183.05it/s, accuracy=0.7883, loss=1.6881]
100%|██████████| 157/157 [00:00<00:00, 439.08it/s, accuracy=0.8579, loss=1.6146]
Epoch: 74/100
100%|██████████| 938/938 [00:04<00:00, 193.56it/s, accuracy=0.7882, loss=1.6877]
100%|██████████| 157/157 [00:00<00:00, 429.35it/s, accuracy=0.8573, loss=1.6150]
Epoch: 75/100
100%|██████████| 938/938 [00:04<00:00, 193.46it/s, accuracy=0.7906, loss=1.6854]
100%|██████████| 157/157 [00:00<00:00, 432.93it/s, accuracy=0.8596, loss=1.6129]
Epoch: 76/100
100%|██████████| 938/938 [00:04<00:00, 193.17it/s, accuracy=0.7917, loss=1.6833]
100%|██████████| 157/157 [00:00<00:00, 457.26it/s, accuracy=0.8591, loss=1.6129]
Epoch: 77/100
100%|██████████| 938/938 [00:04<00:00, 192.22it/s, accuracy=0.7944, loss=1.6807]
100%|██████████| 157/157 [00:00<00:00, 468.97it/s, accuracy=0.8628, loss=1.6109]
Epoch: 78/100
100%|██████████| 938/938 [00:04<00:00, 192.39it/s, accuracy=0.7990, loss=1.6764]
100%|██████████| 157/157 [00:00<00:00, 420.09it/s, accuracy=0.8612, loss=1.6099]
Epoch: 79/100
100%|██████████| 938/938 [00:04<00:00, 192.49it/s, accuracy=0.7972, loss=1.6779]
100%|██████████| 157/157 [00:00<00:00, 467.70it/s, accuracy=0.8607, loss=1.6113]
Epoch: 80/100
100%|██████████| 938/938 [00:04<00:00, 190.30it/s, accuracy=0.8001, loss=1.6744]
100%|██████████| 157/157 [00:00<00:00, 472.07it/s, accuracy=0.8644, loss=1.6083]
Epoch: 81/100
100%|██████████| 938/938 [00:04<00:00, 193.13it/s, accuracy=0.8000, loss=1.6743]
100%|██████████| 157/157 [00:00<00:00, 456.24it/s, accuracy=0.8623, loss=1.6084]
Epoch: 82/100
```

```
100%|██████████| 938/938 [00:04<00:00, 192.58it/s, accuracy=0.8018, loss=1.6726]
100%|██████████| 157/157 [00:00<00:00, 470.32it/s, accuracy=0.8609, loss=1.6098]
Epoch: 83/100
100%|██████████| 938/938 [00:04<00:00, 193.15it/s, accuracy=0.8038, loss=1.6702]
100%|██████████| 157/157 [00:00<00:00, 468.85it/s, accuracy=0.8620, loss=1.6078]
Epoch: 84/100
100%|██████████| 938/938 [00:04<00:00, 192.42it/s, accuracy=0.8043, loss=1.6697]
100%|██████████| 157/157 [00:00<00:00, 472.73it/s, accuracy=0.8627, loss=1.6068]
Epoch: 85/100
100%|██████████| 938/938 [00:04<00:00, 192.40it/s, accuracy=0.8065, loss=1.6667]
100%|██████████| 157/157 [00:00<00:00, 455.30it/s, accuracy=0.8629, loss=1.6063]
Epoch: 86/100
100%|██████████| 938/938 [00:04<00:00, 192.12it/s, accuracy=0.8069, loss=1.6664]
100%|██████████| 157/157 [00:00<00:00, 469.48it/s, accuracy=0.8643, loss=1.6045]
Epoch: 87/100
100%|██████████| 938/938 [00:04<00:00, 192.94it/s, accuracy=0.8093, loss=1.6638]
100%|██████████| 157/157 [00:00<00:00, 461.51it/s, accuracy=0.8647, loss=1.6048]
Epoch: 88/100
100%|██████████| 938/938 [00:04<00:00, 192.63it/s, accuracy=0.8102, loss=1.6629]
100%|██████████| 157/157 [00:00<00:00, 469.32it/s, accuracy=0.8653, loss=1.6030]
Epoch: 89/100
100%|██████████| 938/938 [00:04<00:00, 192.89it/s, accuracy=0.8110, loss=1.6622]
100%|██████████| 157/157 [00:00<00:00, 466.72it/s, accuracy=0.8653, loss=1.6028]
Epoch: 90/100
100%|██████████| 938/938 [00:05<00:00, 179.55it/s, accuracy=0.8122, loss=1.6604]
100%|██████████| 157/157 [00:00<00:00, 471.74it/s, accuracy=0.8652, loss=1.6029]
Epoch: 91/100
100%|██████████| 938/938 [00:05<00:00, 185.70it/s, accuracy=0.8104, loss=1.6615]
100%|██████████| 157/157 [00:00<00:00, 471.59it/s, accuracy=0.8648, loss=1.6020]
Epoch: 92/100
```

```
100%|██████████| 938/938 [00:04<00:00, 191.22it/s, accuracy=0.8143, loss=1.6580]
100%|██████████| 157/157 [00:00<00:00, 472.59it/s, accuracy=0.8667, loss=1.6009]
Epoch: 93/100
100%|██████████| 938/938 [00:04<00:00, 193.34it/s, accuracy=0.8149, loss=1.6569]
100%|██████████| 157/157 [00:00<00:00, 461.90it/s, accuracy=0.8674, loss=1.5998]
Epoch: 94/100
100%|██████████| 938/938 [00:04<00:00, 190.51it/s, accuracy=0.8173, loss=1.6555]
100%|██████████| 157/157 [00:00<00:00, 458.80it/s, accuracy=0.8663, loss=1.6014]
Epoch: 95/100
100%|██████████| 938/938 [00:05<00:00, 161.19it/s, accuracy=0.8168, loss=1.6553]
100%|██████████| 157/157 [00:00<00:00, 390.59it/s, accuracy=0.8659, loss=1.6007]
Epoch: 96/100
100%|██████████| 938/938 [00:05<00:00, 174.85it/s, accuracy=0.8180, loss=1.6539]
100%|██████████| 157/157 [00:00<00:00, 474.41it/s, accuracy=0.8681, loss=1.5997]
Epoch: 97/100
100%|██████████| 938/938 [00:05<00:00, 168.80it/s, accuracy=0.8178, loss=1.6531]
100%|██████████| 157/157 [00:00<00:00, 472.48it/s, accuracy=0.8677, loss=1.5985]
Epoch: 98/100
100%|██████████| 938/938 [00:04<00:00, 193.93it/s, accuracy=0.8188, loss=1.6526]
100%|██████████| 157/157 [00:00<00:00, 447.67it/s, accuracy=0.8668, loss=1.5986]
Epoch: 99/100
100%|██████████| 938/938 [00:04<00:00, 200.07it/s, accuracy=0.8182, loss=1.6527]
100%|██████████| 157/157 [00:00<00:00, 469.84it/s, accuracy=0.8691, loss=1.5975]
Epoch: 100/100
100%|██████████| 938/938 [00:04<00:00, 189.96it/s, accuracy=0.8217, loss=1.6499]
100%|██████████| 157/157 [00:00<00:00, 475.05it/s, accuracy=0.8660, loss=1.6026]
```

In [21]:
```
headers = history2.keys()
rows = zip(*history2.values())
print(tabulate(rows, headers=headers, tablefmt="pretty"))
```

```
+--------------------+---------------------+--------------------+----------
-+
```

loss_train	acc_train	loss_test	acc_test
2.3023917994181313	0.11323333333333334	2.3015937885284425	0.1276
2.301779421488444	0.12786666666666666	2.3004300525665284	0.1411
2.301105199686686	0.12621666666666667	2.299012487792969	0.1157
2.300234745279948	0.10966666666666666	2.297085223007202	0.0997
2.2989735811869303	0.0999	2.2939566635131836	0.098
2.2967103847503663	0.09873333333333334	2.2872416290283204	0.098
2.291719249725342	0.09871666666666666	2.2726911727905272	0.098
2.2843660423278807	0.10476666666666666	2.2576027240753174	0.1703
2.277094981002808	0.14411666666666667	2.2443132095336913	0.2138
2.269759626261393	0.17028333333333334	2.233527021026611	0.2413
2.2631625747680664	0.18615	2.224640707397461	0.2526
2.2553498407999673	0.19888333333333333	2.2138193153381347	0.2598
2.246927773920695	0.21073333333333333	2.197453458404541	0.262
2.2338521931966144	0.22133333333333333	2.1702111965179443	0.2765
2.2170833276112876	0.23645	2.141159670257568	0.3091
2.195816873041789	0.2620166666666667	2.10979799156189	0.3618
2.1739661933898926	0.29801666666666665	2.0757017070770263	0.4212
2.149686218770345	0.33513333333333334	2.036761261367798	0.4899
2.1274956385294597	0.3612166666666667	1.9996891317367553	0.5266
2.1027778367360432	0.39986666666666665	1.9622000730514526	0.5783
2.08357397702535	0.41575	1.9382757385253906	0.5879
2.0657051109313964	0.4271	1.9164779043197633	0.5999
2.0502357849121093	0.4388	1.8985953496932984	0.6068
2.0400845144907636	0.4433666666666667	1.8888752071380615	0.6091
2.029832869974772	0.4518333333333333	1.881192015838623	0.6118

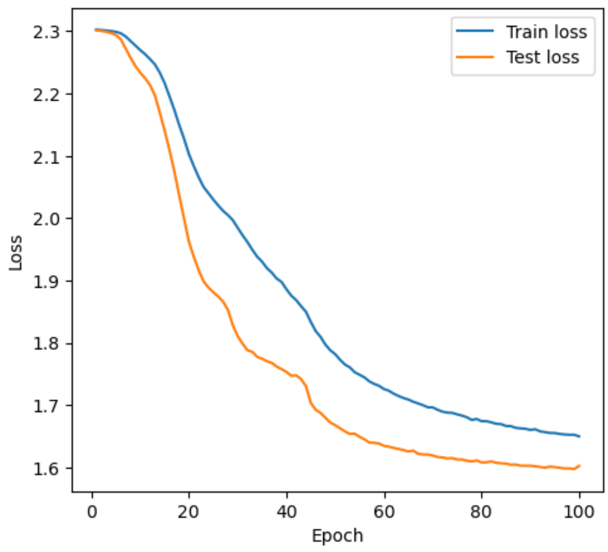
2.020633637491862	0.45868333333333333	1.8742718015670776	0.6147
2.0117791975657147	0.46675	1.8662612926483155	0.6169
2.004992022387187	0.4717166666666667	1.852794076538086	0.6262
1.9968504940032958	0.4816666666666667	1.8283619041442871	0.6718
1.9846370325724283	0.4977666666666667	1.810757007408142	0.6953
1.9726618200937907	0.5093166666666666	1.7988089021682738	0.7032
1.9616771308898926	0.5202333333333333	1.788227031135559	0.71
1.9494058584213256	0.5333166666666667	1.785608842086792	0.7089
1.938273152669271	0.54365	1.7775830883026122	0.716
1.9302074719111124	0.5521166666666667	1.7745666666030884	0.7181
1.9197937030792236	0.5609	1.770523743247986	0.7189
1.912576283899943	0.56895	1.767826984024048	0.7217
1.9031891501108806	0.5777333333333333	1.7619387874603272	0.7264
1.8976596705118816	0.5819333333333333	1.7578776252746582	0.73
1.8859897172927858	0.5946666666666667	1.7533020643234254	0.7351
1.8756669408798219	0.6038833333333333	1.7473428983688355	0.7398
1.8685280645370483	0.6108666666666667	1.7478581304550171	0.7371
1.8588582169850667	0.6220833333333333	1.741906698036194	0.7438
1.8498946297963461	0.62945	1.7304668962478638	0.7471
1.8334701229731243	0.6612166666666667	1.7035838775634766	0.8135
1.8189258522669474	0.6784	1.6926774543762206	0.8199
1.8095341126124065	0.6851666666666667	1.6874780223846435	0.8174
1.797077129236857	0.6968166666666666	1.6799101720809937	0.8216
1.78806405321757	0.7055833333333333	1.6721400651931764	0.8267
1.7817921817143758	0.7095666666666667	1.6679342073440553	0.8298
1.7728199191411336	0.7180666666666666	1.6629396572113038	0.8304

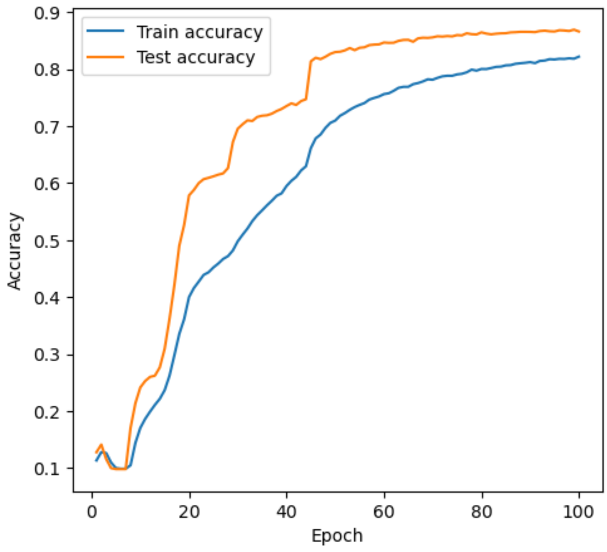
1.7651949092229207	0.7228166666666667	1.6579278593063353	0.8328
1.760677334022522	0.7285166666666667	1.6539058145523071	0.8369
1.7529308547973632	0.7333	1.6542914155960082	0.8331
1.7487792910893758	0.7370833333333333	1.6492848066329957	0.8373
1.7446007921854656	0.7403166666666666	1.6452130805969238	0.8382
1.7381847756067912	0.7464333333333333	1.6401126943588258	0.842
1.7340990405400594	0.7493166666666666	1.6395992765426635	0.8429
1.731024103609721	0.7519166666666667	1.6384345928192139	0.8433
1.7259252298355103	0.75605	1.6348146808624267	0.8466
1.7231981597264607	0.7575	1.6335084999084473	0.846
1.7184376949946085	0.76185	1.6313756980895997	0.8462
1.714709228960673	0.7671	1.629794972229004	0.8495
1.7114928155263265	0.7688333333333334	1.628007110595703	0.8511
1.7090858845392862	0.7688666666666667	1.6257491724014281	0.8513
1.7055176521937052	0.7734833333333333	1.6271148765563965	0.8479
1.7028684744517009	0.7754166666666666	1.6220443759918213	0.8539
1.7000056770324707	0.7783	1.620961150741577	0.855
1.6965951473236085	0.7820333333333334	1.6208162227630616	0.8547
1.6964880917867025	0.78135	1.6193018367767333	0.8556
1.6926296309789022	0.7845666666666666	1.6168323093414307	0.8575
1.6896281602223715	0.78725	1.6162059545516967	0.857
1.6881227117538453	0.78825	1.6146186946868897	0.8579
1.6876972617467245	0.7881666666666667	1.6150481632232665	0.8573
1.6853521058400471	0.7906166666666666	1.612942784500122	0.8596
1.6833227584203085	0.7917	1.6128609712600708	0.8591
1.6806547307332356	0.79435	1.610928844833374	0.8628
1.6763857836405436	0.7990333333333334	1.6098757152557372	0.8612

```
|
| 1.6779367630004882 | 0.7972333333333333  | 1.611296816444397  |  0.8607
|
| 1.6743898506800334 | 0.8001333333333334  | 1.6082801803588866 |  0.8644
|
| 1.6742977463404338 |        0.8          | 1.608447886276245  |  0.8623
|
| 1.672567369969686  | 0.8017666666666666  | 1.6097805156707763 |  0.8609
|
| 1.6701994068145751 |       0.8038        | 1.607767364692688  |  0.862
|
| 1.6696795998891194 | 0.8042833333333334  | 1.6067954328536986 |  0.8627
|
| 1.6666988358179728 | 0.8064833333333333  | 1.6063465614318848 |  0.8629
|
| 1.6664206731796265 | 0.8069166666666666  | 1.6045262624740602 |  0.8643
|
| 1.6637693227767945 |       0.80925       |  1.60480203666687  |  0.8647
|
| 1.662858319536845  | 0.8101666666666667  | 1.6029749931335449 |  0.8653
|
| 1.6622229460398357 |        0.811        | 1.602805283164978  |  0.8653
|
| 1.660368582979838  | 0.8122166666666667  | 1.6029315828323365 |  0.8652
|
| 1.661468242963155  | 0.8104166666666667  | 1.6020055009841918 |  0.8648
|
| 1.6579865900675457 | 0.8142666666666667  | 1.6009479763031005 |  0.8667
|
| 1.6568754144032796 | 0.8149333333333333  | 1.5998276706695556 |  0.8674
|
| 1.655471849822998  | 0.8173166666666667  | 1.6014277215957642 |  0.8663
|
| 1.6553486333847045 |       0.81685       | 1.6007486192703246 |  0.8659
|
| 1.6538977573394775 | 0.8179833333333333  | 1.599662559890747  |  0.8681
|
| 1.6531105925242107 | 0.8177666666666666  | 1.598543883895874  |  0.8677
|
| 1.6526153662363687 |       0.81885       | 1.5986144170761107 |  0.8668
|
| 1.652674685160319  | 0.8181666666666667  | 1.5974585599899291 |  0.8691
|
| 1.6499494305928548 | 0.8217333333333333  | 1.602588733291626  |  0.866
|
+--------------------+---------------------+--------------------+---------
-+
```

In [22]:

```
epochs = range(1, len(history2["loss_train"]) + 1)
draw_loss_test(epochs, history2)
```





In [26]:
```python
net, history = training(
    net,
    train_loader,
    test_loader,
    loss_fn,
    metric,
    optimizer,
    update_period=5,
    epoch_max=100,
    device=device,
    early_stopping_accuracy=0.98
)
```

```
Epoka: 1/100
100%|██████████| 938/938
[00:03<00:00, 246.67it/s]
Train loss: 2.3023
Train accuracy: 0.1151
100%|██████████| 157/157
[00:00<00:00, 713.15it/s]
Test loss: 2.3015
Test accuracy: 0.1905

Epoka: 2/100
100%|██████████| 938/938
[00:03<00:00, 250.51it/s]
Train loss: 2.3016
Train accuracy: 0.1659
100%|██████████| 157/157
[00:00<00:00, 732.43it/s]
Test loss: 2.3003
Test accuracy: 0.2611

Epoka: 3/100
100%|██████████| 938/938
[00:03<00:00, 249.77it/s]
```

```
Train loss: 2.3008
Train accuracy: 0.1692
100%|██████████████████████████████| 157/157
[00:00<00:00, 637.85it/s]
Test loss: 2.2987
Test accuracy: 0.2170

Epoka: 4/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 249.93it/s]
Train loss: 2.2998
Train accuracy: 0.1371
100%|██████████████████████████████| 157/157
[00:00<00:00, 736.85it/s]
Test loss: 2.2966
Test accuracy: 0.1370

Epoka: 5/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 249.55it/s]
Train loss: 2.2984
Train accuracy: 0.1083
100%|██████████████████████████████| 157/157
[00:00<00:00, 719.84it/s]
Test loss: 2.2933
Test accuracy: 0.1096

Epoka: 6/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 247.41it/s]
Train loss: 2.2958
Train accuracy: 0.1007
100%|██████████████████████████████| 157/157
[00:00<00:00, 715.40it/s]
Test loss: 2.2862
Test accuracy: 0.0988

Epoka: 7/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 247.97it/s]
Train loss: 2.2903
Train accuracy: 0.0990
100%|██████████████████████████████| 157/157
[00:00<00:00, 724.32it/s]
Test loss: 2.2714
Test accuracy: 0.0989

Epoka: 8/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 247.94it/s]
```

```
Train loss: 2.2824
Train accuracy: 0.1140
100%|██████████████████████████████| 157/157
[00:00<00:00, 721.41it/s]
Test loss: 2.2551
Test accuracy: 0.1885

Epoka: 9/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 246.45it/s]
Train loss: 2.2745
Train accuracy: 0.1557
100%|██████████████████████████████| 157/157
[00:00<00:00, 713.42it/s]
Test loss: 2.2403
Test accuracy: 0.2304

Epoka: 10/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 248.26it/s]
Train loss: 2.2661
Train accuracy: 0.1824
100%|██████████████████████████████| 157/157
[00:00<00:00, 690.45it/s]
Test loss: 2.2268
Test accuracy: 0.2549

Epoka: 11/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 247.61it/s]
Train loss: 2.2579
Train accuracy: 0.2041
100%|██████████████████████████████| 157/157
[00:00<00:00, 685.75it/s]
Test loss: 2.2118
Test accuracy: 0.2700

Epoka: 12/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 244.63it/s]
Train loss: 2.2462
Train accuracy: 0.2210
100%|██████████████████████████████| 157/157
[00:00<00:00, 685.65it/s]
Test loss: 2.1881
Test accuracy: 0.2797

Epoka: 13/100
100%|██████████████████████████████| 938/938
[00:03<00:00, 246.40it/s]
```

```
Train loss: 2.2321
Train accuracy: 0.2276
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 686.89it/s]
Test loss: 2.1604
Test accuracy: 0.2811

Epoka: 14/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 246.54it/s]
Train loss: 2.2143
Train accuracy: 0.2364
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 684.99it/s]
Test loss: 2.1322
Test accuracy: 0.3105

Epoka: 15/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 245.21it/s]
Train loss: 2.1950
Train accuracy: 0.2673
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 687.70it/s]
Test loss: 2.1030
Test accuracy: 0.3761

Epoka: 16/100
100%|████████████████████████████████████████| 938/938
[00:04<00:00, 230.16it/s]
Train loss: 2.1716
Train accuracy: 0.3055
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 699.42it/s]
Test loss: 2.0693
Test accuracy: 0.4537

Epoka: 17/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 244.60it/s]
Train loss: 2.1495
Train accuracy: 0.3418
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 702.59it/s]
Test loss: 2.0328
Test accuracy: 0.5003

Epoka: 18/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 241.76it/s]
```

```
Train loss: 2.1257
Train accuracy: 0.3679
100%|██████████| 157/157
[00:00<00:00, 704.33it/s]
Test loss: 1.9981
Test accuracy: 0.5231

Epoka: 19/100
100%|██████████| 938/938
[00:03<00:00, 245.17it/s]
Train loss: 2.1079
Train accuracy: 0.3816
100%|██████████| 157/157
[00:00<00:00, 698.16it/s]
Test loss: 1.9725
Test accuracy: 0.5324

Epoka: 20/100
100%|██████████| 938/938
[00:03<00:00, 244.77it/s]
Train loss: 2.0895
Train accuracy: 0.3948
100%|██████████| 157/157
[00:00<00:00, 630.84it/s]
Test loss: 1.9506
Test accuracy: 0.5425

Epoka: 21/100
100%|██████████| 938/938
[00:03<00:00, 244.56it/s]
Train loss: 2.0776
Train accuracy: 0.4012
100%|██████████| 157/157
[00:00<00:00, 698.37it/s]
Test loss: 1.9338
Test accuracy: 0.5478

Epoka: 22/100
100%|██████████| 938/938
[00:03<00:00, 242.25it/s]
Train loss: 2.0644
Train accuracy: 0.4132
100%|██████████| 157/157
[00:00<00:00, 679.73it/s]
Test loss: 1.9105
Test accuracy: 0.5792

Epoka: 23/100
100%|██████████| 938/938
[00:04<00:00, 208.82it/s]
```

```
Train loss: 2.0518
Train accuracy: 0.4268
100%|██████████| 157/157
[00:00<00:00, 689.74it/s]
Test loss: 1.8863
Test accuracy: 0.6171

Epoka: 24/100
100%|██████████| 938/938
[00:04<00:00, 233.79it/s]
Train loss: 2.0395
Train accuracy: 0.4378
100%|██████████| 157/157
[00:00<00:00, 614.96it/s]
Test loss: 1.8687
Test accuracy: 0.6237

Epoka: 25/100
100%|██████████| 938/938
[00:04<00:00, 231.02it/s]
Train loss: 2.0272
Train accuracy: 0.4524
100%|██████████| 157/157
[00:00<00:00, 691.65it/s]
Test loss: 1.8454
Test accuracy: 0.6784

Epoka: 26/100
100%|██████████| 938/938
[00:04<00:00, 230.70it/s]
Train loss: 2.0128
Train accuracy: 0.4710
100%|██████████| 157/157
[00:00<00:00, 682.00it/s]
Test loss: 1.8311
Test accuracy: 0.6865

Epoka: 27/100
100%|██████████| 938/938
[00:03<00:00, 238.10it/s]
Train loss: 2.0013
Train accuracy: 0.4828
100%|██████████| 157/157
[00:00<00:00, 621.07it/s]
Test loss: 1.8200
Test accuracy: 0.6917

Epoka: 28/100
100%|██████████| 938/938
[00:03<00:00, 242.90it/s]
```

```
Train loss: 1.9901
Train accuracy: 0.4942
100%|██████████| 157/157 [00:00<00:00, 705.30it/s]
Test loss: 1.8102
Test accuracy: 0.6980

Epoka: 29/100
100%|██████████| 938/938 [00:03<00:00, 243.15it/s]
Train loss: 1.9806
Train accuracy: 0.5027
100%|██████████| 157/157 [00:00<00:00, 633.52it/s]
Test loss: 1.8058
Test accuracy: 0.6989

Epoka: 30/100
100%|██████████| 938/938 [00:04<00:00, 220.69it/s]
Train loss: 1.9697
Train accuracy: 0.5134
100%|██████████| 157/157 [00:00<00:00, 665.14it/s]
Test loss: 1.7973
Test accuracy: 0.7030

Epoka: 31/100
100%|██████████| 938/938 [00:03<00:00, 240.05it/s]
Train loss: 1.9601
Train accuracy: 0.5215
100%|██████████| 157/157 [00:00<00:00, 619.20it/s]
Test loss: 1.7918
Test accuracy: 0.7092

Epoka: 32/100
100%|██████████| 938/938 [00:03<00:00, 241.88it/s]
Train loss: 1.9525
Train accuracy: 0.5281
100%|██████████| 157/157 [00:00<00:00, 683.34it/s]
Test loss: 1.7848
Test accuracy: 0.7122

Epoka: 33/100
100%|██████████| 938/938 [00:03<00:00, 241.79it/s]
```

```
Train loss: 1.9426
Train accuracy: 0.5383
100%|██████████| 157/157
[00:00<00:00, 688.53it/s]
Test loss: 1.7837
Test accuracy: 0.7116

Epoka: 34/100
100%|██████████| 938/938
[00:03<00:00, 240.03it/s]
Train loss: 1.9323
Train accuracy: 0.5486
100%|██████████| 157/157
[00:00<00:00, 696.59it/s]
Test loss: 1.7764
Test accuracy: 0.7173

Epoka: 35/100
100%|██████████| 938/938
[00:03<00:00, 242.13it/s]
Train loss: 1.9245
Train accuracy: 0.5554
100%|██████████| 157/157
[00:00<00:00, 684.99it/s]
Test loss: 1.7727
Test accuracy: 0.7222

Epoka: 36/100
100%|██████████| 938/938
[00:03<00:00, 242.05it/s]
Train loss: 1.9151
Train accuracy: 0.5648
100%|██████████| 157/157
[00:00<00:00, 694.42it/s]
Test loss: 1.7692
Test accuracy: 0.7247

Epoka: 37/100
100%|██████████| 938/938
[00:03<00:00, 240.56it/s]
Train loss: 1.9070
Train accuracy: 0.5749
100%|██████████| 157/157
[00:00<00:00, 683.91it/s]
Test loss: 1.7651
Test accuracy: 0.7288

Epoka: 38/100
100%|██████████| 938/938
[00:03<00:00, 242.10it/s]
```

```
Train loss: 1.8974
Train accuracy: 0.5829
100%|██████████| 157/157
[00:00<00:00, 687.42it/s]
Test loss: 1.7598
Test accuracy: 0.7316

Epoka: 39/100
100%|██████████| 938/938
[00:03<00:00, 241.95it/s]
Train loss: 1.8892
Train accuracy: 0.5931
100%|██████████| 157/157
[00:00<00:00, 664.83it/s]
Test loss: 1.7547
Test accuracy: 0.7368

Epoka: 40/100
100%|██████████| 938/938
[00:03<00:00, 240.57it/s]
Train loss: 1.8795
Train accuracy: 0.6015
100%|██████████| 157/157
[00:00<00:00, 678.88it/s]
Test loss: 1.7531
Test accuracy: 0.7368

Epoka: 41/100
100%|██████████| 938/938
[00:03<00:00, 241.56it/s]
Train loss: 1.8692
Train accuracy: 0.6126
100%|██████████| 157/157
[00:00<00:00, 676.98it/s]
Test loss: 1.7488
Test accuracy: 0.7379

Epoka: 42/100
100%|██████████| 938/938
[00:03<00:00, 240.14it/s]
Train loss: 1.8629
Train accuracy: 0.6185
100%|██████████| 157/157
[00:00<00:00, 610.79it/s]
Test loss: 1.7470
Test accuracy: 0.7393

Epoka: 43/100
100%|██████████| 938/938
[00:03<00:00, 241.46it/s]
```

```
Train loss: 1.8533
Train accuracy: 0.6288
100%|██████████| 157/157
[00:00<00:00, 685.25it/s]
Test loss: 1.7453
Test accuracy: 0.7386

Epoka: 44/100
100%|██████████| 938/938
[00:03<00:00, 241.30it/s]
Train loss: 1.8466
Train accuracy: 0.6353
100%|██████████| 157/157
[00:00<00:00, 686.41it/s]
Test loss: 1.7416
Test accuracy: 0.7417

Epoka: 45/100
100%|██████████| 938/938
[00:03<00:00, 238.09it/s]
Train loss: 1.8416
Train accuracy: 0.6397
100%|██████████| 157/157
[00:00<00:00, 673.71it/s]
Test loss: 1.7383
Test accuracy: 0.7448

Epoka: 46/100
100%|██████████| 938/938
[00:04<00:00, 224.69it/s]
Train loss: 1.8341
Train accuracy: 0.6462
100%|██████████| 157/157
[00:00<00:00, 667.89it/s]
Test loss: 1.7360
Test accuracy: 0.7467

Epoka: 47/100
100%|██████████| 938/938
[00:03<00:00, 240.58it/s]
Train loss: 1.8300
Train accuracy: 0.6487
100%|██████████| 157/157
[00:00<00:00, 675.04it/s]
Test loss: 1.7332
Test accuracy: 0.7476

Epoka: 48/100
100%|██████████| 938/938
[00:03<00:00, 239.23it/s]
```

```
Train loss: 1.8217
Train accuracy: 0.6573
100%|████████████████████| 157/157
[00:00<00:00, 665.17it/s]
Test loss: 1.7297
Test accuracy: 0.7495

Epoka: 49/100
100%|████████████████████| 938/938
[00:03<00:00, 241.38it/s]
Train loss: 1.8162
Train accuracy: 0.6636
100%|████████████████████| 157/157
[00:00<00:00, 691.73it/s]
Test loss: 1.7268
Test accuracy: 0.7510

Epoka: 50/100
100%|████████████████████| 938/938
[00:03<00:00, 241.39it/s]
Train loss: 1.8118
Train accuracy: 0.6671
100%|████████████████████| 157/157
[00:00<00:00, 679.91it/s]
Test loss: 1.7245
Test accuracy: 0.7527

Epoka: 51/100
100%|████████████████████| 938/938
[00:03<00:00, 236.94it/s]
Train loss: 1.8076
Train accuracy: 0.6697
100%|████████████████████| 157/157
[00:00<00:00, 679.60it/s]
Test loss: 1.7231
Test accuracy: 0.7540

Epoka: 52/100
100%|████████████████████| 938/938
[00:03<00:00, 240.62it/s]
Train loss: 1.8018
Train accuracy: 0.6756
100%|████████████████████| 157/157
[00:00<00:00, 673.83it/s]
Test loss: 1.7211
Test accuracy: 0.7543

Epoka: 53/100
100%|████████████████████| 938/938
[00:03<00:00, 239.81it/s]
```

```
Train loss: 1.7996
Train accuracy: 0.6777
100%|██████████| 157/157
[00:00<00:00, 600.01it/s]
Test loss: 1.7186
Test accuracy: 0.7573

Epoka: 54/100
100%|██████████| 938/938
[00:03<00:00, 239.60it/s]
Train loss: 1.7941
Train accuracy: 0.6825
100%|██████████| 157/157
[00:00<00:00, 677.72it/s]
Test loss: 1.7187
Test accuracy: 0.7547

Epoka: 55/100
100%|██████████| 938/938
[00:03<00:00, 239.97it/s]
Train loss: 1.7922
Train accuracy: 0.6846
100%|██████████| 157/157
[00:00<00:00, 661.82it/s]
Test loss: 1.7142
Test accuracy: 0.7600

Epoka: 56/100
100%|██████████| 938/938
[00:03<00:00, 237.85it/s]
Train loss: 1.7889
Train accuracy: 0.6861
100%|██████████| 157/157
[00:00<00:00, 665.58it/s]
Test loss: 1.7143
Test accuracy: 0.7592

Epoka: 57/100
100%|██████████| 938/938
[00:03<00:00, 239.50it/s]
Train loss: 1.7846
Train accuracy: 0.6907
100%|██████████| 157/157
[00:00<00:00, 658.31it/s]
Test loss: 1.7111
Test accuracy: 0.7620

Epoka: 58/100
100%|██████████| 938/938
[00:03<00:00, 240.21it/s]
```

```
Train loss: 1.7822
Train accuracy: 0.6927
100%|██████████| 157/157 [00:00<00:00, 663.23it/s]
Test loss: 1.7108
Test accuracy: 0.7616

Epoka: 59/100
100%|██████████| 938/938 [00:03<00:00, 238.73it/s]
Train loss: 1.7801
Train accuracy: 0.6948
100%|██████████| 157/157 [00:00<00:00, 674.06it/s]
Test loss: 1.7094
Test accuracy: 0.7616

Epoka: 60/100
100%|██████████| 938/938 [00:03<00:00, 239.13it/s]
Train loss: 1.7777
Train accuracy: 0.6959
100%|██████████| 157/157 [00:00<00:00, 675.39it/s]
Test loss: 1.7076
Test accuracy: 0.7636

Epoka: 61/100
100%|██████████| 938/938 [00:03<00:00, 239.61it/s]
Train loss: 1.7759
Train accuracy: 0.6976
100%|██████████| 157/157 [00:00<00:00, 678.15it/s]
Test loss: 1.7060
Test accuracy: 0.7648

Epoka: 62/100
100%|██████████| 938/938 [00:03<00:00, 235.24it/s]
Train loss: 1.7717
Train accuracy: 0.7012
100%|██████████| 157/157 [00:00<00:00, 593.69it/s]
Test loss: 1.7055
Test accuracy: 0.7652

Epoka: 63/100
100%|██████████| 938/938 [00:03<00:00, 239.62it/s]
```

```
Train loss: 1.7699
Train accuracy: 0.7037
100%|██████████| 157/157
[00:00<00:00, 673.99it/s]
Test loss: 1.7042
Test accuracy: 0.7660

Epoka: 64/100
100%|██████████| 938/938
[00:03<00:00, 239.55it/s]
Train loss: 1.7660
Train accuracy: 0.7076
100%|██████████| 157/157
[00:00<00:00, 676.79it/s]
Test loss: 1.7036
Test accuracy: 0.7658

Epoka: 65/100
100%|██████████| 938/938
[00:03<00:00, 236.60it/s]
Train loss: 1.7656
Train accuracy: 0.7071
100%|██████████| 157/157
[00:00<00:00, 663.19it/s]
Test loss: 1.7011
Test accuracy: 0.7684

Epoka: 66/100
100%|██████████| 938/938
[00:03<00:00, 240.33it/s]
Train loss: 1.7634
Train accuracy: 0.7099
100%|██████████| 157/157
[00:00<00:00, 644.84it/s]
Test loss: 1.7019
Test accuracy: 0.7674

Epoka: 67/100
100%|██████████| 938/938
[00:03<00:00, 237.69it/s]
Train loss: 1.7615
Train accuracy: 0.7101
100%|██████████| 157/157
[00:00<00:00, 675.59it/s]
Test loss: 1.6993
Test accuracy: 0.7701

Epoka: 68/100
100%|██████████| 938/938
[00:03<00:00, 237.88it/s]
```

```
Train loss: 1.7595
Train accuracy: 0.7129
100%|██████████| 157/157
[00:00<00:00, 673.30it/s]
Test loss: 1.6986
Test accuracy: 0.7705

Epoka: 69/100
100%|██████████| 938/938
[00:03<00:00, 240.28it/s]
Train loss: 1.7563
Train accuracy: 0.7152
100%|██████████| 157/157
[00:00<00:00, 666.92it/s]
Test loss: 1.6978
Test accuracy: 0.7714

Epoka: 70/100
100%|██████████| 938/938
[00:03<00:00, 238.14it/s]
Train loss: 1.7568
Train accuracy: 0.7140
100%|██████████| 157/157
[00:00<00:00, 678.51it/s]
Test loss: 1.6965
Test accuracy: 0.7731

Epoka: 71/100
100%|██████████| 938/938
[00:03<00:00, 239.67it/s]
Train loss: 1.7542
Train accuracy: 0.7171
100%|██████████| 157/157
[00:00<00:00, 675.45it/s]
Test loss: 1.6955
Test accuracy: 0.7723

Epoka: 72/100
100%|██████████| 938/938
[00:03<00:00, 239.72it/s]
Train loss: 1.7524
Train accuracy: 0.7183
100%|██████████| 157/157
[00:00<00:00, 679.26it/s]
Test loss: 1.6945
Test accuracy: 0.7740

Epoka: 73/100
100%|██████████| 938/938
[00:03<00:00, 238.92it/s]
```

```
Train loss: 1.7503
Train accuracy: 0.7217
100%|██████████| 157/157
[00:00<00:00, 689.95it/s]
Test loss: 1.6936
Test accuracy: 0.7738

Epoka: 74/100
100%|██████████| 938/938
[00:03<00:00, 238.95it/s]
Train loss: 1.7499
Train accuracy: 0.7205
100%|██████████| 157/157
[00:00<00:00, 682.48it/s]
Test loss: 1.6936
Test accuracy: 0.7744

Epoka: 75/100
100%|██████████| 938/938
[00:03<00:00, 235.66it/s]
Train loss: 1.7496
Train accuracy: 0.7210
100%|██████████| 157/157
[00:00<00:00, 602.87it/s]
Test loss: 1.6933
Test accuracy: 0.7756

Epoka: 76/100
100%|██████████| 938/938
[00:03<00:00, 239.24it/s]
Train loss: 1.7469
Train accuracy: 0.7240
100%|██████████| 157/157
[00:00<00:00, 675.33it/s]
Test loss: 1.6929
Test accuracy: 0.7742

Epoka: 77/100
100%|██████████| 938/938
[00:03<00:00, 239.56it/s]
Train loss: 1.7459
Train accuracy: 0.7252
100%|██████████| 157/157
[00:00<00:00, 617.45it/s]
Test loss: 1.6906
Test accuracy: 0.7773

Epoka: 78/100
100%|██████████| 938/938
[00:03<00:00, 237.46it/s]
```

```
Train loss: 1.7430
Train accuracy: 0.7267
100%|██████████| 157/157
[00:00<00:00, 663.36it/s]
Test loss: 1.6900
Test accuracy: 0.7764

Epoka: 79/100
100%|██████████| 938/938
[00:03<00:00, 239.28it/s]
Train loss: 1.7439
Train accuracy: 0.7259
100%|██████████| 157/157
[00:00<00:00, 669.20it/s]
Test loss: 1.6897
Test accuracy: 0.7773

Epoka: 80/100
100%|██████████| 938/938
[00:03<00:00, 240.12it/s]
Train loss: 1.7415
Train accuracy: 0.7280
100%|██████████| 157/157
[00:00<00:00, 681.43it/s]
Test loss: 1.6891
Test accuracy: 0.7781

Epoka: 81/100
100%|██████████| 938/938
[00:03<00:00, 237.58it/s]
Train loss: 1.7412
Train accuracy: 0.7287
100%|██████████| 157/157
[00:00<00:00, 674.95it/s]
Test loss: 1.6889
Test accuracy: 0.7778

Epoka: 82/100
100%|██████████| 938/938
[00:03<00:00, 239.24it/s]
Train loss: 1.7393
Train accuracy: 0.7301
100%|██████████| 157/157
[00:00<00:00, 678.86it/s]
Test loss: 1.6886
Test accuracy: 0.7785

Epoka: 83/100
100%|██████████| 938/938
[00:03<00:00, 239.88it/s]
```

```
Train loss: 1.7385
Train accuracy: 0.7310
100%|██████████| 157/157 [00:00<00:00, 602.86it/s]
Test loss: 1.6877
Test accuracy: 0.7787

Epoka: 84/100
100%|██████████| 938/938 [00:03<00:00, 238.88it/s]
Train loss: 1.7376
Train accuracy: 0.7315
100%|██████████| 157/157 [00:00<00:00, 677.97it/s]
Test loss: 1.6876
Test accuracy: 0.7792

Epoka: 85/100
100%|██████████| 938/938 [00:03<00:00, 239.57it/s]
Train loss: 1.7356
Train accuracy: 0.7337
100%|██████████| 157/157 [00:00<00:00, 680.47it/s]
Test loss: 1.6866
Test accuracy: 0.7796

Epoka: 86/100
100%|██████████| 938/938 [00:03<00:00, 237.68it/s]
Train loss: 1.7353
Train accuracy: 0.7339
100%|██████████| 157/157 [00:00<00:00, 664.36it/s]
Test loss: 1.6857
Test accuracy: 0.7802

Epoka: 87/100
100%|██████████| 938/938 [00:03<00:00, 240.11it/s]
Train loss: 1.7336
Train accuracy: 0.7353
100%|██████████| 157/157 [00:00<00:00, 674.27it/s]
Test loss: 1.6854
Test accuracy: 0.7809

Epoka: 88/100
100%|██████████| 938/938 [00:03<00:00, 240.54it/s]
```

```
Train loss: 1.7338
Train accuracy: 0.7346
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 649.29it/s]
Test loss: 1.6842
Test accuracy: 0.7814

Epoka: 89/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 238.05it/s]
Train loss: 1.7324
Train accuracy: 0.7358
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 669.34it/s]
Test loss: 1.6842
Test accuracy: 0.7819

Epoka: 90/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 237.74it/s]
Train loss: 1.7309
Train accuracy: 0.7379
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 659.02it/s]
Test loss: 1.6843
Test accuracy: 0.7813

Epoka: 91/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 237.96it/s]
Train loss: 1.7311
Train accuracy: 0.7370
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 680.71it/s]
Test loss: 1.6836
Test accuracy: 0.7820

Epoka: 92/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 238.76it/s]
Train loss: 1.7287
Train accuracy: 0.7398
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 661.63it/s]
Test loss: 1.6824
Test accuracy: 0.7822

Epoka: 93/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 238.81it/s]
```

```
Train loss: 1.7278
Train accuracy: 0.7407
100%|██████████| 157/157
[00:00<00:00, 673.09it/s]
Test loss: 1.6827
Test accuracy: 0.7825

Epoka: 94/100
100%|██████████| 938/938
[00:03<00:00, 238.77it/s]
Train loss: 1.7273
Train accuracy: 0.7410
100%|██████████| 157/157
[00:00<00:00, 676.83it/s]
Test loss: 1.6820
Test accuracy: 0.7825

Epoka: 95/100
100%|██████████| 938/938
[00:03<00:00, 239.07it/s]
Train loss: 1.7275
Train accuracy: 0.7405
100%|██████████| 157/157
[00:00<00:00, 671.91it/s]
Test loss: 1.6824
Test accuracy: 0.7823

Epoka: 96/100
100%|██████████| 938/938
[00:03<00:00, 238.67it/s]
Train loss: 1.7259
Train accuracy: 0.7422
100%|██████████| 157/157
[00:00<00:00, 606.60it/s]
Test loss: 1.6822
Test accuracy: 0.7819

Epoka: 97/100
100%|██████████| 938/938
[00:04<00:00, 223.24it/s]
Train loss: 1.7256
Train accuracy: 0.7417
100%|██████████| 157/157
[00:00<00:00, 581.40it/s]
Test loss: 1.6805
Test accuracy: 0.7837

Epoka: 98/100
100%|██████████| 938/938
[00:03<00:00, 238.74it/s]
```

```
Train loss: 1.7251
Train accuracy: 0.7432
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 669.25it/s]
Test loss: 1.6809
Test accuracy: 0.7830

Epoka: 99/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 238.04it/s]
Train loss: 1.7244
Train accuracy: 0.7435
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 670.58it/s]
Test loss: 1.6811
Test accuracy: 0.7826

Epoka: 100/100
100%|████████████████████████████████████████| 938/938
[00:03<00:00, 238.64it/s]
Train loss: 1.7234
Train accuracy: 0.7443
100%|████████████████████████████████████████| 157/157
[00:00<00:00, 675.67it/s]
Test loss: 1.6821
Test accuracy: 0.7815
```

In [27]:
```
headers = history.keys()
rows = zip(*history.values())
print(tabulate(rows, headers=headers, tablefmt='pretty'))
```

```
+--------------------+---------------------+--------------------+---------
-+
|    loss_train      |      acc_train      |     loss_test      | acc_test
|
+--------------------+---------------------+--------------------+---------
-+
| 2.302253891372681  | 0.11513333333333334 | 2.3015244480133057 |  0.1905
|
| 2.3015840198516844 |       0.1659        | 2.300276218032837  |  0.2611
|
| 2.300844240442912  | 0.16923333333333335 | 2.2987437141418456 |  0.217
|
|  2.29984623743693  | 0.13713333333333333 | 2.2966318099975584 |  0.137
|
| 2.2984486667633055 | 0.10833333333333334 | 2.2932555164337156 |  0.1096
|
| 2.2958373302459716 |       0.10065       | 2.2861803852081297 |  0.0988
|
| 2.290309753545125  | 0.09903333333333333 | 2.2713675563812257 |  0.0989
|
| 2.282408146540324  | 0.11401666666666667 | 2.2551241802215576 |  0.1885
```

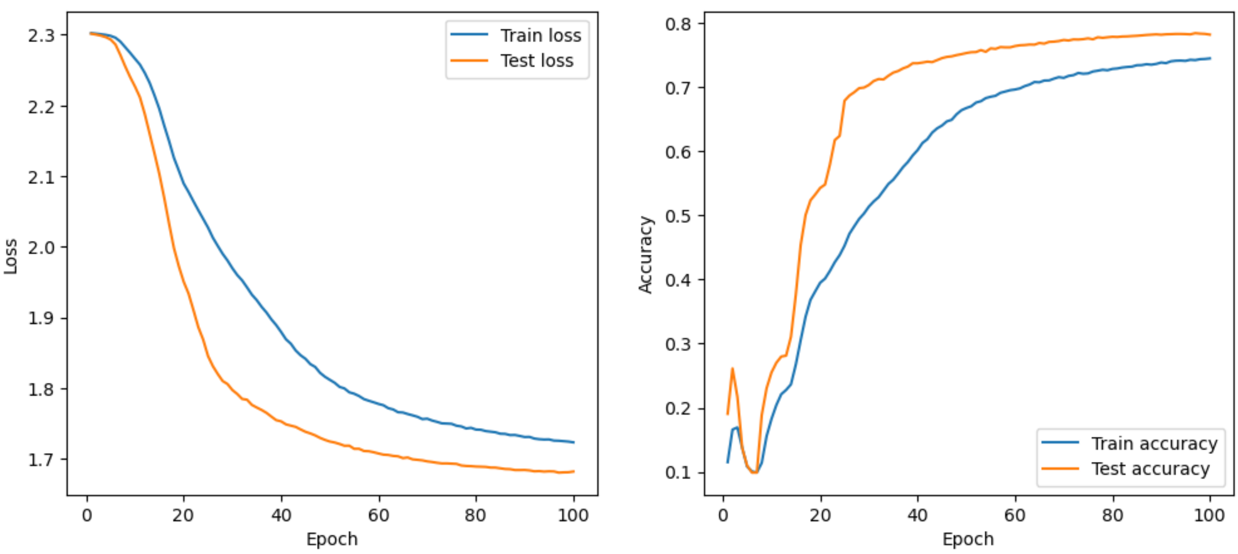
2.274461056772868	0.1557	2.2402670627593992	0.2304
2.2661020184834797	0.18236666666666668	2.226756703186035	0.2549
2.2579274979909263	0.2041	2.2117916984558104	0.27
2.246191771952311	0.22096666666666667	2.188144364929199	0.2797
2.2321220471700034	0.22765	2.1603829486846924	0.2811
2.214310261408488	0.23635	2.1322396579742433	0.3105
2.195043679936727	0.2673333333333333	2.1030152755737306	0.3761
2.171557197189331	0.30548333333333333	2.0693416980743407	0.4537
2.1495066078186036	0.34176666666666666	2.032821955490112	0.5003
2.125733141454061	0.36795	1.9980935123443604	0.5231
2.107855320549011	0.38155	1.972504916191101	0.5324
2.089513953145345	0.39476666666666665	1.9505500785827636	0.5425
2.077560060755412	0.40123333333333333	1.9338432062149047	0.5478
2.0644258569081626	0.41318333333333335	1.9105321641921997	0.5792
2.051775353240967	0.42678333333333335	1.886344721031189	0.6171
2.0395387331644694	0.43783333333333335	1.8686718469619752	0.6237
2.02718345896403	0.45243333333333335	1.845398878288269	0.6784
2.012774441019694	0.47103333333333336	1.831066000556946	0.6865
2.0012552225748697	0.48278333333333334	1.820034048652649	0.6917
1.9900975998560588	0.4942166666666667	1.8101621887207031	0.698
1.9806062050501505	0.5026833333333334	1.8058312572479247	0.6989
1.9696693993886312	0.5134166666666666	1.7973448667526246	0.703
1.9600650800704955	0.5215166666666666	1.7917962175369262	0.7092
1.9525055934270223	0.5281333333333333	1.784802610397339	0.7122
1.942646544011434	0.53835	1.7837140434265137	0.7116
1.9323185910542806	0.5486333333333333	1.7763620609283448	0.7173

1.924548531850179	0.55535	1.7726523147583009	0.7222
1.9151218688964844	0.5648333333333333	1.7691554153442384	0.7247
1.9070255125045776	0.5748833333333333	1.7650784734725953	0.7288
1.8973816565831503	0.5829166666666666	1.7597935619354248	0.7316
1.8892160735448202	0.59315	1.7547349527359009	0.7368
1.8795310118993123	0.6014833333333334	1.7530566568374635	0.7368
1.8691610275268555	0.6125666666666667	1.7487932538986206	0.7379
1.862936517270406	0.6184666666666667	1.7470100233078003	0.7393
1.8532951913833617	0.6287666666666667	1.74525532207489	0.7386
1.8465696384429933	0.6353166666666666	1.741640721130371	0.7417
1.8415825243631998	0.6397166666666667	1.7383205612182617	0.7448
1.8340559964497885	0.6462166666666667	1.73600588722229	0.7467
1.8300407899220785	0.6486833333333333	1.7332159881591798	0.7476
1.8217039003372193	0.6572833333333333	1.7296825006484986	0.7495
1.8161588084538778	0.6636	1.726840637397766	0.751
1.8118267523447673	0.66705	1.7245335115432738	0.7527
1.807637751897176	0.6697166666666666	1.7231371269226075	0.754
1.8018449128468832	0.6755833333333333	1.721068071937561	0.7543
1.7996356215159097	0.6776666666666666	1.7185642642974854	0.7573
1.7941316501617433	0.6825333333333333	1.718650922203064	0.7547
1.7921761681874593	0.6845833333333333	1.7142424886703491	0.76
1.7888977373758952	0.6861333333333334	1.7142597557067871	0.7592
1.7845726915359497	0.69075	1.7111425048828126	0.762
1.782200001525879	0.6927166666666666	1.7108277946472168	0.7616
1.780063365618388	0.6948333333333333	1.7093623094558716	0.7616
1.7777117013295491	0.6958833333333333	1.7076313745498657	0.7636
1.775894584274292	0.6976333333333333	1.7059596210479737	0.7648

1.7717196739196777	0.7012	1.70550514087677	0.7652
1.7699199912389119	0.7036666666666667	1.7042118900299073	0.766
1.7660190184911093	0.7076166666666667	1.70364856300354	0.7658
1.7655882168451944	0.7071166666666666	1.7010777143478393	0.7684
1.7634079926172892	0.7098833333333333	1.7019110918045044	0.7674
1.7615486763636272	0.7100666666666666	1.6993078481674195	0.7701
1.7595406878789266	0.7128666666666666	1.6986120492935182	0.7705
1.7562510000864664	0.7151833333333333	1.6978486322402955	0.7714
1.7568376184463501	0.7140166666666666	1.6964753234863281	0.7731
1.7542193012873333	0.7171	1.6955408672332763	0.7723
1.752369516436259	0.7182666666666667	1.694457053565979	0.774
1.7502541810353598	0.7217	1.6935770433425903	0.7738
1.7499465909322103	0.72045	1.693608193206787	0.7744
1.7495777379989623	0.721	1.693255319404602	0.7756
1.7469037206013998	0.7239666666666666	1.6928973207473754	0.7742
1.7458859028498332	0.7252333333333333	1.6906259256362914	0.7773
1.7430447369893391	0.7266833333333333	1.6899588235855103	0.7764
1.7438965025583903	0.7259166666666667	1.6897081899642945	0.7773
1.741455255762736	0.728	1.6891486711502075	0.7781
1.7411678303400675	0.7287	1.6889302242279052	0.7778
1.739330744934082	0.7301333333333333	1.6886378028869629	0.7785
1.7384634753545125	0.7310166666666666	1.6876980792999268	0.7787
1.7376301279703776	0.7315333333333334	1.6876182680130005	0.7792
1.7355824155171713	0.7337	1.6866435876846313	0.7796
1.7353393600463867	0.7339333333333333	1.6856848493576049	0.7802
1.7336210536956786	0.7353166666666666	1.6854048727035522	0.7809

```
| 1.7337747392018636 |       0.7346        | 1.684184511756897  |  0.7814
|
| 1.7324473445892334 |       0.7358        | 1.6841529350280762 |  0.7819
|
| 1.7309279483159383 |       0.73795       | 1.6842665603637694 |  0.7813
|
| 1.731064684486389  |        0.737        | 1.6836421688079835 |  0.782
|
| 1.728739034016927  | 0.7398333333333333  | 1.6824288076400757 |  0.7822
|
| 1.7278187502543132 | 0.7407333333333334  | 1.6826981336593627 |  0.7825
|
| 1.7273465980529785 | 0.7409666666666667  | 1.6820294580459594 |  0.7825
|
| 1.7274952431360882 | 0.7405333333333334  | 1.6824315338134765 |  0.7823
|
| 1.7258596537907918 | 0.7422333333333333  | 1.682209283065796  |  0.7819
|
| 1.7256150826772054 | 0.7417333333333334  | 1.6805241174697876 |  0.7837
|
| 1.725051641146342  | 0.7431833333333333  | 1.6809105926513672 |  0.783
|
| 1.7244251010258993 | 0.7435333333333334  | 1.6810608158111573 |  0.7826
|
| 1.7233768075942992 | 0.7443333333333333  | 1.6820848791122436 |  0.7815
|
+--------------------+---------------------+--------------------+---------
-+
```

In [28]:
```python
epochs = range(1, len(history["loss_train"]) + 1)
draw_loss_test(epochs, history)
```



12. Save model and optimizer states to a file.

Use method `state_dict` and function `torch.save`.

In [29]:
```
model_path = './models/net.pth'
optimizer_path = "./models/optimizer.pth"

torch.save(net.state_dict(), model_path)
torch.save(optimizer.state_dict(), optimizer_path)
```

In [30]:
```
model_path2 = "./models/net2.pth"
optimizer_path2 = "./models/optimizer2.pth"

torch.save(net2.state_dict(), model_path2)
torch.save(optimizer2.state_dict(), optimizer_path2)
```

In [31]:
```
checkpoint_path = "./checkpoints/checkpoint.pth"

torch.save(
    {
        "model_state_dict": net.state_dict(),
        "optimizer_state_dict": optimizer.state_dict(),
    },
    checkpoint_path,
)
```

In [32]:
```
checkpoint_path2 = "./checkpoints/checkpoint2.pth"

torch.save(
    {
        "model_state_dict": net2.state_dict(),
        "optimizer_state_dict": optimizer2.state_dict(),
    },
    checkpoint_path2,
)
```

13. Create new network with the same architecture and initialize it with saved weights. Compare evaluations for both networks.

Use `torch.load` and `load_state_dict`.

In [33]:
```
net3 = MLP(input_shape, output_size)
optimizer3 = torch.optim.SGD(net2.parameters(), lr=0.01)

net3.load_state_dict(torch.load(model_path))
optimizer3.load_state_dict(torch.load(optimizer_path))
```

```
/var/folders/0b/brzkvl1j0tn9xzynh5pr39sw0000gn/T/ipykernel_17512/140974620
3.py:4: FutureWarning: You are using `torch.load` with `weights_only=False
` (the current default value), which uses the default pickle module implic
itly. It is possible to construct malicious pickle data which will execute
arbitrary code during unpickling (See https://github.com/pytorch/pytorch/b
lob/main/SECURITY.md#untrusted-models for more details). In a future relea
se, the default value for `weights_only` will be flipped to `True`. This l
imits the functions that could be executed during unpickling. Arbitrary ob
jects will no longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via `torch.serialization.add_safe_globa
ls`. We recommend you start setting `weights_only=True` for any use case w
here you don't have full control of the loaded file. Please open an issue
on GitHub for any issues related to this experimental feature.
  net3.load_state_dict(torch.load(model_path))
/var/folders/0b/brzkvl1j0tn9xzynh5pr39sw0000gn/T/ipykernel_17512/140974620
3.py:5: FutureWarning: You are using `torch.load` with `weights_only=False
` (the current default value), which uses the default pickle module implic
itly. It is possible to construct malicious pickle data which will execute
arbitrary code during unpickling (See https://github.com/pytorch/pytorch/b
lob/main/SECURITY.md#untrusted-models for more details). In a future relea
se, the default value for `weights_only` will be flipped to `True`. This l
imits the functions that could be executed during unpickling. Arbitrary ob
jects will no longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via `torch.serialization.add_safe_globa
ls`. We recommend you start setting `weights_only=True` for any use case w
here you don't have full control of the loaded file. Please open an issue
on GitHub for any issues related to this experimental feature.
  optimizer3.load_state_dict(torch.load(optimizer_path))
```

In [34]:

```
checkpoint = torch.load("./checkpoints/checkpoint.pth")
net4 = MLP(input_shape, output_size)
optimizer4 = torch.optim.SGD(net3.parameters(), lr=0.01)

net4.load_state_dict(checkpoint["model_state_dict"])
optimizer4.load_state_dict(checkpoint["optimizer_state_dict"])

net4.to(device)
```

```
/var/folders/0b/brzkvl1j0tn9xzynh5pr39sw0000gn/T/ipykernel_17512/47768843.
py:1: FutureWarning: You are using `torch.load` with `weights_only=False`
(the current default value), which uses the default pickle module implicit
ly. It is possible to construct malicious pickle data which will execute a
rbitrary code during unpickling (See https://github.com/pytorch/pytorch/bl
ob/main/SECURITY.md#untrusted-models for more details). In a future releas
e, the default value for `weights_only` will be flipped to `True`. This li
mits the functions that could be executed during unpickling. Arbitrary obj
ects will no longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via `torch.serialization.add_safe_globa
ls`. We recommend you start setting `weights_only=True` for any use case w
here you don't have full control of the loaded file. Please open an issue
on GitHub for any issues related to this experimental feature.
  checkpoint = torch.load("./checkpoints/checkpoint.pth")
```

Out[34]:
```
MLP(
  (model): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=784, out_features=512, bias=True)
    (2): ReLU()
    (3): Linear(in_features=512, out_features=512, bias=True)
    (4): ReLU()
    (5): Linear(in_features=512, out_features=10, bias=True)
    (6): Softmax(dim=1)
  )
)
```

14. **EXTENSION EXERCISE**

Define your own model and train it.

Try to achieve better results.

You can use different parameters, layers e.g.:

- conv2d
- maxpooling2d
- batch norm 2d
- and more...

Save weights to a file.

In [35]:
```
import torch
import torch.nn as nn
import torch.optim as optim


class FastCNNModel(nn.Module):

    def __init__(self, input_shape, output_size) -> None:
        super(FastCNNModel, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(
                in_channels=input_shape[0],
                out_channels=16,
                kernel_size=3,
                padding=1,
            ),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(
                in_channels=16, out_channels=32, kernel_size=3, padding=1
            ),
            nn.BatchNorm2d(32),
            nn.ReLU(),
```

```
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(
                in_channels=32, out_channels=64, kernel_size=3, padding=1
            ),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Flatten(),
            nn.Dropout(0.3),
            nn.Linear(64 * (input_shape[1] // 8) * (input_shape[2] // 8),
            nn.ReLU(),
            nn.Linear(128, output_size),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.model(x)
```

In [38]:

```
my_net = FastCNNModel(input_shape, output_size)
my_net = my_net.to(device)

my_loss_fn = nn.CrossEntropyLoss()
my_optimizer = optim.SGD(my_net.parameters(), lr=0.01, momentum=0.9)
```

In [39]:

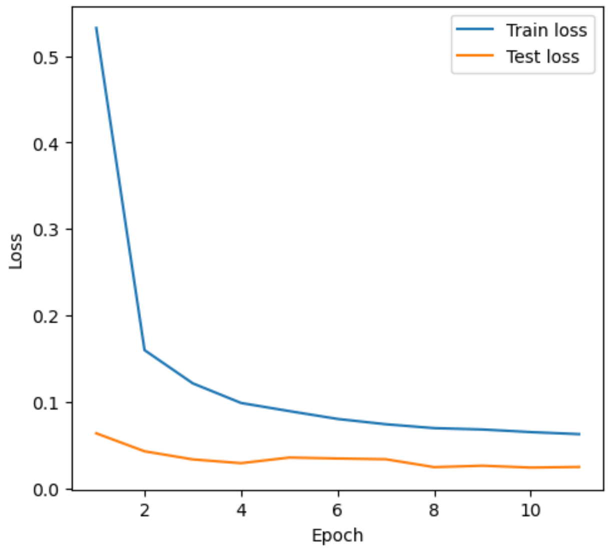
```
my_net, my_history = test_or_train(
    my_net,
    train_loader,
    test_loader,
    my_loss_fn,
    metric,
    my_optimizer,
    mode="both",
    update_period=5,
    epoch_max=100,
    device=device,
    early_stopping_accuracy=0.98,
)
```

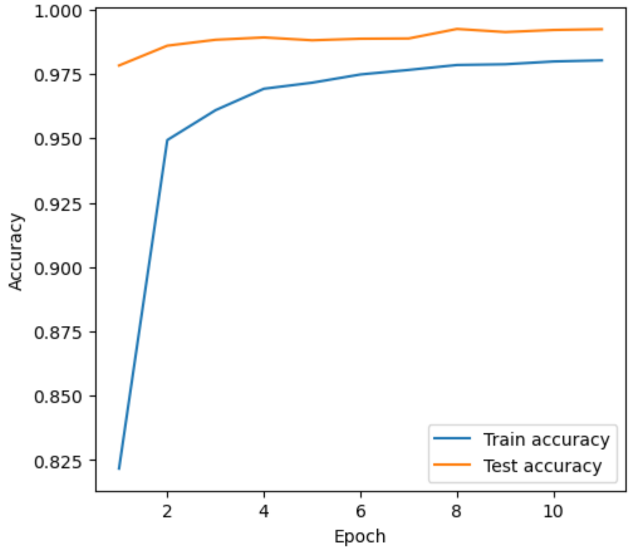
```
Epoch: 1/100
100%|████████████████████████████████| 938/938 [00:19<00:00, 47.52it/s, acc
uracy=0.8217, loss=0.5324]
100%|███████████████████████████████| 157/157 [00:00<00:00, 169.21it/s, acc
uracy=0.9783, loss=0.0637]
Epoch: 2/100
100%|████████████████████████████████| 938/938 [00:20<00:00, 45.74it/s, acc
uracy=0.9493, loss=0.1599]
100%|███████████████████████████████| 157/157 [00:01<00:00, 134.75it/s, acc
uracy=0.9860, loss=0.0429]
Epoch: 3/100
100%|████████████████████████████████| 938/938 [00:21<00:00, 43.72it/s, acc
uracy=0.9609, loss=0.1215]
100%|███████████████████████████████| 157/157 [00:01<00:00, 140.04it/s, acc
uracy=0.9883, loss=0.0335]
```

```
Epoch: 4/100
100%|██████████| 938/938 [00:21<00:00, 43.52it/s, accuracy=0.9693, loss=0.0988]
100%|██████████| 157/157 [00:01<00:00, 147.58it/s, accuracy=0.9892, loss=0.0292]
Epoch: 5/100
100%|██████████| 938/938 [00:22<00:00, 42.43it/s, accuracy=0.9716, loss=0.0894]
100%|██████████| 157/157 [00:01<00:00, 140.32it/s, accuracy=0.9881, loss=0.0357]
Epoch: 6/100
100%|██████████| 938/938 [00:20<00:00, 46.21it/s, accuracy=0.9748, loss=0.0803]
100%|██████████| 157/157 [00:01<00:00, 147.13it/s, accuracy=0.9887, loss=0.0346]
Epoch: 7/100
100%|██████████| 938/938 [00:20<00:00, 45.93it/s, accuracy=0.9766, loss=0.0742]
100%|██████████| 157/157 [00:00<00:00, 157.42it/s, accuracy=0.9888, loss=0.0337]
Epoch: 8/100
100%|██████████| 938/938 [00:20<00:00, 45.29it/s, accuracy=0.9785, loss=0.0697]
100%|██████████| 157/157 [00:01<00:00, 145.98it/s, accuracy=0.9925, loss=0.0246]
Epoch: 9/100
100%|██████████| 938/938 [00:21<00:00, 43.49it/s, accuracy=0.9788, loss=0.0681]
100%|██████████| 157/157 [00:00<00:00, 181.97it/s, accuracy=0.9913, loss=0.0263]
Epoch: 10/100
100%|██████████| 938/938 [00:20<00:00, 46.66it/s, accuracy=0.9799, loss=0.0651]
100%|██████████| 157/157 [00:00<00:00, 191.42it/s, accuracy=0.9921, loss=0.0241]
Epoch: 11/100
100%|██████████| 938/938 [00:18<00:00, 50.54it/s, accuracy=0.9803, loss=0.0627]
100%|██████████| 157/157 [00:00<00:00, 185.16it/s, accuracy=0.9924, loss=0.0248]
Training accuracy of 0.9803 achieved, stopping training.
```

In [40]:

```
my_epochs = range(1, len(my_history["loss_train"]) + 1)
draw_loss_test(my_epochs, my_history)
```





In [41]:
```
class SimpleFastCNNModel(nn.Module):

    def __init__(self, input_shape, output_size) -> None:
        super(SimpleFastCNNModel, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(
                in_channels=input_shape[0],
                out_channels=8,
                kernel_size=3,
                padding=1,
            ),
            nn.BatchNorm2d(8),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(
                in_channels=8, out_channels=16, kernel_size=3, padding=1
            ),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(16, output_size),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.model(x)
```

In [45]:
```
my_net2 = SimpleFastCNNModel(input_shape, output_size)
my_net2 = my_net2.to(device)

my_loss_fn2 = nn.CrossEntropyLoss()
my_optimizer2 = optim.SGD(my_net2.parameters(), lr=0.01, momentum=0.9)
```

In [46]:
```
my_net2, my_history2 = test_or_train(
    my_net2,
```

```
    train_loader,
    test_loader,
    my_loss_fn2,
    metric,
    my_optimizer2,
    mode="both",
    update_period=5,
    epoch_max=100,
    device=device,
    early_stopping_accuracy=0.98,
)
```

```
Epoch: 1/100
100%|██████████| 938/938 [00:11<00:00, 78.62it/s, accuracy=0.3919, loss=1.7501]
100%|██████████| 157/157 [00:00<00:00, 291.11it/s, accuracy=0.6191, loss=1.2049]
Epoch: 2/100
100%|██████████| 938/938 [00:11<00:00, 78.98it/s, accuracy=0.6599, loss=1.0627]
100%|██████████| 157/157 [00:00<00:00, 286.25it/s, accuracy=0.7627, loss=0.7199]
Epoch: 3/100
100%|██████████| 938/938 [00:11<00:00, 79.12it/s, accuracy=0.7495, loss=0.7874]
100%|██████████| 157/157 [00:00<00:00, 284.97it/s, accuracy=0.7657, loss=0.6997]
Epoch: 4/100
100%|██████████| 938/938 [00:11<00:00, 79.11it/s, accuracy=0.7990, loss=0.6453]
100%|██████████| 157/157 [00:00<00:00, 268.17it/s, accuracy=0.8615, loss=0.4619]
Epoch: 5/100
100%|██████████| 938/938 [00:12<00:00, 76.77it/s, accuracy=0.8203, loss=0.5798]
100%|██████████| 157/157 [00:00<00:00, 274.58it/s, accuracy=0.8783, loss=0.4097]
Epoch: 6/100
100%|██████████| 938/938 [00:11<00:00, 78.68it/s, accuracy=0.8342, loss=0.5287]
100%|██████████| 157/157 [00:00<00:00, 277.01it/s, accuracy=0.9102, loss=0.3242]
Epoch: 7/100
100%|██████████| 938/938 [00:11<00:00, 79.33it/s, accuracy=0.8514, loss=0.4839]
100%|██████████| 157/157 [00:00<00:00, 285.39it/s, accuracy=0.8951, loss=0.3536]
Epoch: 8/100
```

```
100%|██████████| 938/938 [00:11<00:00, 79.34it/s, accuracy=0.8563, loss=0.4606]
100%|██████████| 157/157 [00:00<00:00, 287.09it/s, accuracy=0.8962, loss=0.3302]
Epoch: 9/100
100%|██████████| 938/938 [00:11<00:00, 79.44it/s, accuracy=0.8641, loss=0.4383]
100%|██████████| 157/157 [00:00<00:00, 283.92it/s, accuracy=0.9130, loss=0.2996]
Epoch: 10/100
100%|██████████| 938/938 [00:12<00:00, 77.43it/s, accuracy=0.8658, loss=0.4287]
100%|██████████| 157/157 [00:00<00:00, 285.38it/s, accuracy=0.9286, loss=0.2466]
Epoch: 11/100
100%|██████████| 938/938 [00:11<00:00, 79.31it/s, accuracy=0.8717, loss=0.4103]
100%|██████████| 157/157 [00:00<00:00, 290.86it/s, accuracy=0.9125, loss=0.2946]
Epoch: 12/100
100%|██████████| 938/938 [00:11<00:00, 79.35it/s, accuracy=0.8781, loss=0.3975]
100%|██████████| 157/157 [00:00<00:00, 286.46it/s, accuracy=0.9172, loss=0.2728]
Epoch: 13/100
100%|██████████| 938/938 [00:11<00:00, 79.19it/s, accuracy=0.8779, loss=0.3936]
100%|██████████| 157/157 [00:00<00:00, 289.21it/s, accuracy=0.9283, loss=0.2346]
Epoch: 14/100
100%|██████████| 938/938 [00:11<00:00, 79.12it/s, accuracy=0.8764, loss=0.3931]
100%|██████████| 157/157 [00:00<00:00, 287.19it/s, accuracy=0.9288, loss=0.2365]
Epoch: 15/100
100%|██████████| 938/938 [00:11<00:00, 79.14it/s, accuracy=0.8835, loss=0.3729]
100%|██████████| 157/157 [00:00<00:00, 264.25it/s, accuracy=0.9336, loss=0.2269]
Epoch: 16/100
100%|██████████| 938/938 [00:11<00:00, 79.29it/s, accuracy=0.8847, loss=0.3671]
100%|██████████| 157/157 [00:00<00:00, 265.41it/s, accuracy=0.9332, loss=0.2297]
Epoch: 17/100
100%|██████████| 938/938 [00:11<00:00, 78.92it/s, accuracy=0.8867, loss=0.3668]
100%|██████████| 157/157 [00:00<00:00, 279.61it/s, accuracy=0.9403, loss=0.2047]
Epoch: 18/100
```

```
100%|██████████| 938/938 [00:11<00:00, 79.48it/s, accuracy=0.8895, loss=0.3560]
100%|██████████| 157/157 [00:00<00:00, 289.79it/s, accuracy=0.9318, loss=0.2331]
Epoch: 19/100
100%|██████████| 938/938 [00:11<00:00, 80.07it/s, accuracy=0.8928, loss=0.3499]
100%|██████████| 157/157 [00:00<00:00, 290.81it/s, accuracy=0.9371, loss=0.2140]
Epoch: 20/100
100%|██████████| 938/938 [00:11<00:00, 80.02it/s, accuracy=0.8926, loss=0.3443]
100%|██████████| 157/157 [00:00<00:00, 290.61it/s, accuracy=0.9316, loss=0.2264]
Epoch: 21/100
100%|██████████| 938/938 [00:11<00:00, 79.61it/s, accuracy=0.8928, loss=0.3455]
100%|██████████| 157/157 [00:00<00:00, 286.85it/s, accuracy=0.9344, loss=0.2143]
Epoch: 22/100
100%|██████████| 938/938 [00:11<00:00, 79.80it/s, accuracy=0.8948, loss=0.3392]
100%|██████████| 157/157 [00:00<00:00, 290.39it/s, accuracy=0.9379, loss=0.2173]
Epoch: 23/100
100%|██████████| 938/938 [00:11<00:00, 78.67it/s, accuracy=0.8957, loss=0.3343]
100%|██████████| 157/157 [00:00<00:00, 277.56it/s, accuracy=0.9457, loss=0.1908]
Epoch: 24/100
100%|██████████| 938/938 [00:11<00:00, 80.10it/s, accuracy=0.8983, loss=0.3302]
100%|██████████| 157/157 [00:00<00:00, 279.19it/s, accuracy=0.9342, loss=0.2181]
Epoch: 25/100
100%|██████████| 938/938 [00:11<00:00, 80.23it/s, accuracy=0.9003, loss=0.3223]
100%|██████████| 157/157 [00:00<00:00, 295.19it/s, accuracy=0.9448, loss=0.1879]
Epoch: 26/100
100%|██████████| 938/938 [00:39<00:00, 23.76it/s, accuracy=0.8994, loss=0.3249]
100%|██████████| 157/157 [00:00<00:00, 238.36it/s, accuracy=0.9498, loss=0.1875]
Epoch: 27/100
100%|██████████| 938/938 [00:12<00:00, 76.69it/s, accuracy=0.8992, loss=0.3233]
100%|██████████| 157/157 [00:00<00:00, 280.33it/s, accuracy=0.9408, loss=0.1947]
Epoch: 28/100
```

```
100%|██████████| 938/938 [00:11<00:00, 78.66it/s, acc
uracy=0.8994, loss=0.3240]
100%|██████████| 157/157 [00:00<00:00, 266.86it/s, acc
uracy=0.9455, loss=0.1863]
Epoch: 29/100
100%|██████████| 938/938 [00:12<00:00, 73.42it/s, acc
uracy=0.9013, loss=0.3180]
100%|██████████| 157/157 [00:00<00:00, 228.04it/s, acc
uracy=0.9473, loss=0.1765]
Epoch: 30/100
100%|██████████| 938/938 [00:11<00:00, 78.32it/s, acc
uracy=0.9009, loss=0.3175]
100%|██████████| 157/157 [00:00<00:00, 272.28it/s, acc
uracy=0.9401, loss=0.1966]
Epoch: 31/100
100%|██████████| 938/938 [00:11<00:00, 78.54it/s, acc
uracy=0.9046, loss=0.3128]
100%|██████████| 157/157 [00:00<00:00, 275.49it/s, acc
uracy=0.9487, loss=0.1775]
Epoch: 32/100
100%|██████████| 938/938 [00:11<00:00, 78.57it/s, acc
uracy=0.9039, loss=0.3097]
100%|██████████| 157/157 [00:00<00:00, 271.62it/s, acc
uracy=0.9476, loss=0.1753]
Epoch: 33/100
100%|██████████| 938/938 [00:11<00:00, 78.88it/s, acc
uracy=0.9028, loss=0.3137]
100%|██████████| 157/157 [00:00<00:00, 261.39it/s, acc
uracy=0.9442, loss=0.1904]
Epoch: 34/100
100%|██████████| 938/938 [00:12<00:00, 77.23it/s, acc
uracy=0.9043, loss=0.3091]
100%|██████████| 157/157 [00:00<00:00, 280.07it/s, acc
uracy=0.9366, loss=0.2017]
Epoch: 35/100
100%|██████████| 938/938 [00:11<00:00, 78.52it/s, acc
uracy=0.9044, loss=0.3086]
100%|██████████| 157/157 [00:00<00:00, 279.04it/s, acc
uracy=0.9470, loss=0.1807]
Epoch: 36/100
100%|██████████| 938/938 [00:11<00:00, 78.68it/s, acc
uracy=0.9043, loss=0.3066]
100%|██████████| 157/157 [00:00<00:00, 278.18it/s, acc
uracy=0.9395, loss=0.1869]
Epoch: 37/100
100%|██████████| 938/938 [00:11<00:00, 78.90it/s, acc
uracy=0.9063, loss=0.3026]
100%|██████████| 157/157 [00:00<00:00, 261.12it/s, acc
uracy=0.9395, loss=0.1987]
Epoch: 38/100
```

```
100%|| 938/938 [00:11<00:00, 79.06it/s, accuracy=0.9060, loss=0.3025]
100%|| 157/157 [00:00<00:00, 267.94it/s, accuracy=0.9444, loss=0.1877]
Epoch: 39/100
100%|| 938/938 [00:11<00:00, 79.14it/s, accuracy=0.9078, loss=0.3014]
100%|| 157/157 [00:00<00:00, 262.59it/s, accuracy=0.9442, loss=0.1802]
Epoch: 40/100
100%|| 938/938 [00:11<00:00, 79.37it/s, accuracy=0.9058, loss=0.3007]
100%|| 157/157 [00:00<00:00, 270.84it/s, accuracy=0.9515, loss=0.1634]
Epoch: 41/100
100%|| 938/938 [00:11<00:00, 79.38it/s, accuracy=0.9082, loss=0.2974]
100%|| 157/157 [00:00<00:00, 268.06it/s, accuracy=0.9503, loss=0.1633]
Epoch: 42/100
100%|| 938/938 [00:11<00:00, 78.76it/s, accuracy=0.9080, loss=0.2975]
100%|| 157/157 [00:00<00:00, 279.08it/s, accuracy=0.9545, loss=0.1551]
Epoch: 43/100
100%|| 938/938 [00:11<00:00, 79.20it/s, accuracy=0.9086, loss=0.2965]
100%|| 157/157 [00:00<00:00, 276.27it/s, accuracy=0.9538, loss=0.1607]
Epoch: 44/100
100%|| 938/938 [00:11<00:00, 79.30it/s, accuracy=0.9090, loss=0.2916]
100%|| 157/157 [00:00<00:00, 285.40it/s, accuracy=0.9495, loss=0.1726]
Epoch: 45/100
100%|| 938/938 [00:11<00:00, 79.31it/s, accuracy=0.9107, loss=0.2876]
100%|| 157/157 [00:00<00:00, 284.83it/s, accuracy=0.9443, loss=0.1797]
Epoch: 46/100
100%|| 938/938 [00:11<00:00, 79.34it/s, accuracy=0.9093, loss=0.2913]
100%|| 157/157 [00:00<00:00, 283.83it/s, accuracy=0.9464, loss=0.1765]
Epoch: 47/100
100%|| 938/938 [00:12<00:00, 75.77it/s, accuracy=0.9121, loss=0.2861]
100%|| 157/157 [00:00<00:00, 256.36it/s, accuracy=0.9498, loss=0.1664]
Epoch: 48/100
```

```
100%|██████████| 938/938 [00:11<00:00, 78.73it/s, accuracy=0.9101, loss=0.2876]
100%|██████████| 157/157 [00:00<00:00, 269.65it/s, accuracy=0.9518, loss=0.1637]
Epoch: 49/100
100%|██████████| 938/938 [00:11<00:00, 78.38it/s, accuracy=0.9133, loss=0.2798]
100%|██████████| 157/157 [00:00<00:00, 272.37it/s, accuracy=0.9447, loss=0.1825]
Epoch: 50/100
100%|██████████| 938/938 [00:11<00:00, 78.28it/s, accuracy=0.9123, loss=0.2822]
100%|██████████| 157/157 [00:00<00:00, 275.05it/s, accuracy=0.9492, loss=0.1633]
Epoch: 51/100
100%|██████████| 938/938 [00:11<00:00, 78.49it/s, accuracy=0.9132, loss=0.2766]
100%|██████████| 157/157 [00:00<00:00, 274.05it/s, accuracy=0.9496, loss=0.1666]
Epoch: 52/100
100%|██████████| 938/938 [00:12<00:00, 78.14it/s, accuracy=0.9142, loss=0.2793]
100%|██████████| 157/157 [00:00<00:00, 279.13it/s, accuracy=0.9498, loss=0.1651]
Epoch: 53/100
100%|██████████| 938/938 [00:11<00:00, 78.87it/s, accuracy=0.9133, loss=0.2802]
100%|██████████| 157/157 [00:00<00:00, 265.18it/s, accuracy=0.9549, loss=0.1522]
Epoch: 54/100
100%|██████████| 938/938 [00:11<00:00, 78.53it/s, accuracy=0.9164, loss=0.2704]
100%|██████████| 157/157 [00:00<00:00, 269.68it/s, accuracy=0.9501, loss=0.1656]
Epoch: 55/100
100%|██████████| 938/938 [00:11<00:00, 79.07it/s, accuracy=0.9140, loss=0.2793]
100%|██████████| 157/157 [00:00<00:00, 273.85it/s, accuracy=0.9552, loss=0.1503]
Epoch: 56/100
100%|██████████| 938/938 [00:11<00:00, 78.83it/s, accuracy=0.9141, loss=0.2798]
100%|██████████| 157/157 [00:00<00:00, 277.04it/s, accuracy=0.9497, loss=0.1705]
Epoch: 57/100
100%|██████████| 938/938 [00:11<00:00, 79.27it/s, accuracy=0.9131, loss=0.2776]
100%|██████████| 157/157 [00:00<00:00, 279.15it/s, accuracy=0.9550, loss=0.1505]
Epoch: 58/100
```

```
100%|██████████| 938/938 [00:11<00:00, 79.03it/s, accuracy=0.9156, loss=0.2740]
100%|██████████| 157/157 [00:00<00:00, 281.63it/s, accuracy=0.9516, loss=0.1548]
Epoch: 59/100
100%|██████████| 938/938 [00:11<00:00, 79.92it/s, accuracy=0.9160, loss=0.2710]
100%|██████████| 157/157 [00:00<00:00, 278.30it/s, accuracy=0.9539, loss=0.1544]
Epoch: 60/100
100%|██████████| 938/938 [00:11<00:00, 79.63it/s, accuracy=0.9153, loss=0.2707]
100%|██████████| 157/157 [00:00<00:00, 284.80it/s, accuracy=0.9538, loss=0.1531]
Epoch: 61/100
100%|██████████| 938/938 [00:11<00:00, 79.28it/s, accuracy=0.9155, loss=0.2725]
100%|██████████| 157/157 [00:00<00:00, 284.22it/s, accuracy=0.9558, loss=0.1523]
Epoch: 62/100
100%|██████████| 938/938 [00:11<00:00, 79.84it/s, accuracy=0.9145, loss=0.2767]
100%|██████████| 157/157 [00:00<00:00, 279.15it/s, accuracy=0.9558, loss=0.1474]
Epoch: 63/100
100%|██████████| 938/938 [00:11<00:00, 79.46it/s, accuracy=0.9161, loss=0.2694]
100%|██████████| 157/157 [00:00<00:00, 282.57it/s, accuracy=0.9504, loss=0.1637]
Epoch: 64/100
100%|██████████| 938/938 [00:12<00:00, 77.92it/s, accuracy=0.9164, loss=0.2710]
100%|██████████| 157/157 [00:00<00:00, 279.58it/s, accuracy=0.9531, loss=0.1521]
Epoch: 65/100
100%|██████████| 938/938 [00:11<00:00, 79.42it/s, accuracy=0.9165, loss=0.2671]
100%|██████████| 157/157 [00:00<00:00, 275.35it/s, accuracy=0.9518, loss=0.1575]
Epoch: 66/100
100%|██████████| 938/938 [00:11<00:00, 79.64it/s, accuracy=0.9188, loss=0.2664]
100%|██████████| 157/157 [00:00<00:00, 263.34it/s, accuracy=0.9586, loss=0.1440]
Epoch: 67/100
100%|██████████| 938/938 [00:12<00:00, 76.30it/s, accuracy=0.9179, loss=0.2653]
100%|██████████| 157/157 [00:00<00:00, 252.24it/s, accuracy=0.9519, loss=0.1524]
Epoch: 68/100
```

```
100%|██████████| 938/938 [00:11<00:00, 81.12it/s, accuracy=0.9172, loss=0.2642]
100%|██████████| 157/157 [00:00<00:00, 307.49it/s, accuracy=0.9568, loss=0.1405]
Epoch: 69/100
100%|██████████| 938/938 [00:11<00:00, 84.13it/s, accuracy=0.9197, loss=0.2608]
100%|██████████| 157/157 [00:00<00:00, 313.86it/s, accuracy=0.9509, loss=0.1619]
Epoch: 70/100
100%|██████████| 938/938 [00:11<00:00, 83.60it/s, accuracy=0.9193, loss=0.2637]
100%|██████████| 157/157 [00:00<00:00, 248.83it/s, accuracy=0.9503, loss=0.1581]
Epoch: 71/100
100%|██████████| 938/938 [00:11<00:00, 80.68it/s, accuracy=0.9195, loss=0.2611]
100%|██████████| 157/157 [00:00<00:00, 306.41it/s, accuracy=0.9576, loss=0.1441]
Epoch: 72/100
100%|██████████| 938/938 [00:11<00:00, 80.24it/s, accuracy=0.9196, loss=0.2637]
100%|██████████| 157/157 [00:00<00:00, 305.95it/s, accuracy=0.9552, loss=0.1471]
Epoch: 73/100
100%|██████████| 938/938 [00:11<00:00, 84.18it/s, accuracy=0.9211, loss=0.2557]
100%|██████████| 157/157 [00:00<00:00, 301.92it/s, accuracy=0.9552, loss=0.1468]
Epoch: 74/100
100%|██████████| 938/938 [00:10<00:00, 86.07it/s, accuracy=0.9188, loss=0.2623]
100%|██████████| 157/157 [00:00<00:00, 296.90it/s, accuracy=0.9590, loss=0.1391]
Epoch: 75/100
100%|██████████| 938/938 [00:11<00:00, 83.10it/s, accuracy=0.9197, loss=0.2617]
100%|██████████| 157/157 [00:00<00:00, 288.42it/s, accuracy=0.9587, loss=0.1356]
Epoch: 76/100
100%|██████████| 938/938 [00:11<00:00, 81.51it/s, accuracy=0.9198, loss=0.2588]
100%|██████████| 157/157 [00:00<00:00, 271.31it/s, accuracy=0.9547, loss=0.1466]
Epoch: 77/100
100%|██████████| 938/938 [00:12<00:00, 78.13it/s, accuracy=0.9203, loss=0.2599]
100%|██████████| 157/157 [00:00<00:00, 279.63it/s, accuracy=0.9579, loss=0.1417]
Epoch: 78/100
```

```
100%|██████████| 938/938 [00:11<00:00, 83.37it/s, accuracy=0.9197, loss=0.2574]
100%|██████████| 157/157 [00:00<00:00, 308.48it/s, accuracy=0.9563, loss=0.1487]
Epoch: 79/100
100%|██████████| 938/938 [00:11<00:00, 84.60it/s, accuracy=0.9193, loss=0.2587]
100%|██████████| 157/157 [00:00<00:00, 317.58it/s, accuracy=0.9508, loss=0.1600]
Epoch: 80/100
100%|██████████| 938/938 [00:11<00:00, 83.66it/s, accuracy=0.9193, loss=0.2603]
100%|██████████| 157/157 [00:00<00:00, 297.17it/s, accuracy=0.9536, loss=0.1526]
Epoch: 81/100
100%|██████████| 938/938 [00:11<00:00, 82.49it/s, accuracy=0.9226, loss=0.2498]
100%|██████████| 157/157 [00:00<00:00, 289.99it/s, accuracy=0.9576, loss=0.1396]
Epoch: 82/100
100%|██████████| 938/938 [00:11<00:00, 83.22it/s, accuracy=0.9196, loss=0.2582]
100%|██████████| 157/157 [00:00<00:00, 281.37it/s, accuracy=0.9523, loss=0.1561]
Epoch: 83/100
100%|██████████| 938/938 [00:11<00:00, 83.80it/s, accuracy=0.9210, loss=0.2559]
100%|██████████| 157/157 [00:00<00:00, 291.67it/s, accuracy=0.9577, loss=0.1429]
Epoch: 84/100
100%|██████████| 938/938 [00:11<00:00, 84.10it/s, accuracy=0.9206, loss=0.2565]
100%|██████████| 157/157 [00:00<00:00, 321.60it/s, accuracy=0.9603, loss=0.1345]
Epoch: 85/100
100%|██████████| 938/938 [00:11<00:00, 84.23it/s, accuracy=0.9227, loss=0.2518]
100%|██████████| 157/157 [00:00<00:00, 297.64it/s, accuracy=0.9544, loss=0.1461]
Epoch: 86/100
100%|██████████| 938/938 [00:11<00:00, 82.93it/s, accuracy=0.9218, loss=0.2553]
100%|██████████| 157/157 [00:00<00:00, 318.20it/s, accuracy=0.9548, loss=0.1464]
Epoch: 87/100
100%|██████████| 938/938 [00:11<00:00, 80.48it/s, accuracy=0.9214, loss=0.2518]
100%|██████████| 157/157 [00:00<00:00, 278.65it/s, accuracy=0.9566, loss=0.1394]
Epoch: 88/100
```

```
100%|██████████| 938/938 [00:11<00:00, 82.91it/s, accuracy=0.9228, loss=0.2494]
100%|██████████| 157/157 [00:00<00:00, 284.68it/s, accuracy=0.9606, loss=0.1341]
Epoch: 89/100
100%|██████████| 938/938 [00:11<00:00, 84.02it/s, accuracy=0.9235, loss=0.2495]
100%|██████████| 157/157 [00:00<00:00, 307.35it/s, accuracy=0.9598, loss=0.1373]
Epoch: 90/100
100%|██████████| 938/938 [00:11<00:00, 82.54it/s, accuracy=0.9232, loss=0.2499]
100%|██████████| 157/157 [00:00<00:00, 320.00it/s, accuracy=0.9595, loss=0.1363]
Epoch: 91/100
100%|██████████| 938/938 [00:11<00:00, 82.29it/s, accuracy=0.9232, loss=0.2505]
100%|██████████| 157/157 [00:00<00:00, 296.37it/s, accuracy=0.9585, loss=0.1366]
Epoch: 92/100
100%|██████████| 938/938 [00:11<00:00, 82.18it/s, accuracy=0.9236, loss=0.2459]
100%|██████████| 157/157 [00:00<00:00, 296.85it/s, accuracy=0.9590, loss=0.1372]
Epoch: 93/100
100%|██████████| 938/938 [00:11<00:00, 83.82it/s, accuracy=0.9247, loss=0.2484]
100%|██████████| 157/157 [00:00<00:00, 290.85it/s, accuracy=0.9588, loss=0.1363]
Epoch: 94/100
100%|██████████| 938/938 [00:11<00:00, 84.99it/s, accuracy=0.9234, loss=0.2461]
100%|██████████| 157/157 [00:00<00:00, 313.59it/s, accuracy=0.9602, loss=0.1311]
Epoch: 95/100
100%|██████████| 938/938 [00:11<00:00, 80.34it/s, accuracy=0.9250, loss=0.2431]
100%|██████████| 157/157 [00:00<00:00, 277.52it/s, accuracy=0.9570, loss=0.1383]
Epoch: 96/100
100%|██████████| 938/938 [00:12<00:00, 77.95it/s, accuracy=0.9230, loss=0.2475]
100%|██████████| 157/157 [00:00<00:00, 277.87it/s, accuracy=0.9596, loss=0.1384]
Epoch: 97/100
100%|██████████| 938/938 [00:11<00:00, 79.06it/s, accuracy=0.9227, loss=0.2493]
100%|██████████| 157/157 [00:00<00:00, 278.94it/s, accuracy=0.9585, loss=0.1373]
Epoch: 98/100
```

```
100%|██████████| 938/938 [00:11<00:00, 79.43it/s, accuracy=0.9230, loss=0.2503]
100%|██████████| 157/157 [00:00<00:00, 280.35it/s, accuracy=0.9610, loss=0.1310]
Epoch: 99/100
100%|██████████| 938/938 [00:11<00:00, 78.92it/s, accuracy=0.9236, loss=0.2458]
100%|██████████| 157/157 [00:00<00:00, 277.88it/s, accuracy=0.9619, loss=0.1326]
Epoch: 100/100
100%|██████████| 938/938 [00:11<00:00, 78.25it/s, accuracy=0.9233, loss=0.2482]
100%|██████████| 157/157 [00:00<00:00, 269.36it/s, accuracy=0.9606, loss=0.1290]
```

In [47]:
```
my_epochs2 = range(1, len(my_history2["loss_train"]) + 1)
draw_loss_test(my_epochs2, my_history2)
```