Кузнецов Антон **20223** Свёрточные нейронные сети: **MNIST** </h3>

В этом ноутбке мы научимся писать свои свёрточные нейросети на фреймворке **PyTorch,** и протестируем их работу на датасете **MNIST.**

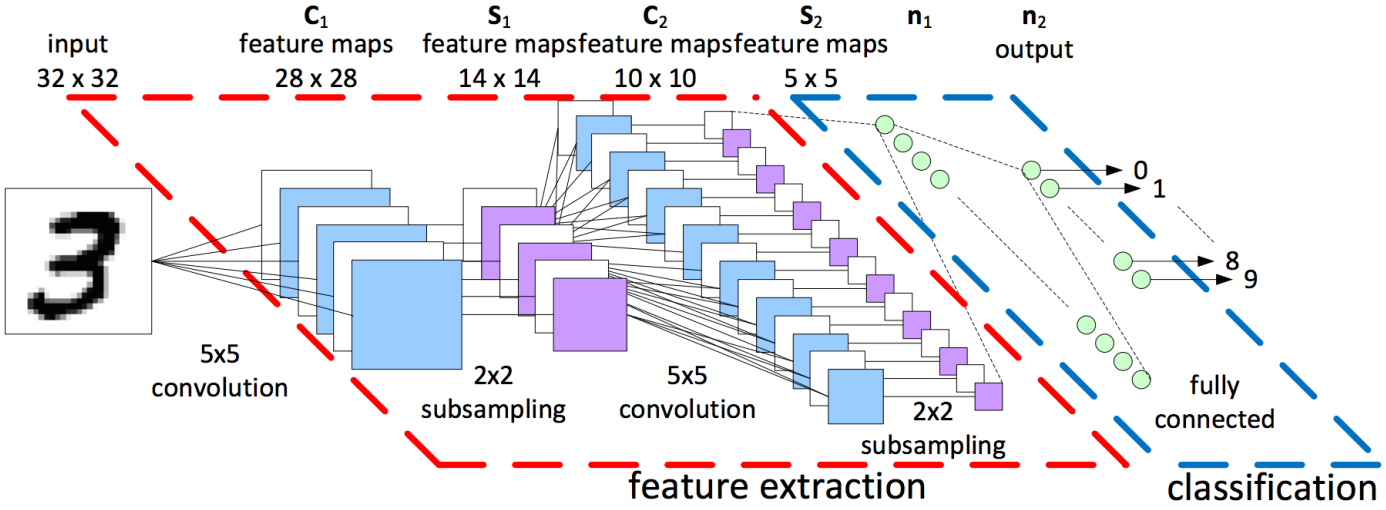
ВНИМАНИЕ: Рассматривается ***задача классификации изображений*.**

Свёрточная нейросеть (Convolutional Neural Network, CNN) **-** это многослойная нейросеть**,** имеющая в своей архитектуре помимо *полносвязных слоёв* **(**а иногда их может и не быть**)** ещё и **свёрточные слои (Conv Layers)** и **pooling-**слои **(Pool Layers).**

Собственно**,** название такое эти сети получили потому**,** что в основе их работы лежит операция **свёртки.**

Сразу же стоит сказать**,** что свёрточные нейросети **были придуманы прежде всего для задач, связанных с изображениями,** следовательно**,** на вход они тоже "ожидают" изображение**.**

- Например**,** вот так выглядит неглубокая свёрточная нейросеть**,** имеющая такую архитектуру**:**
  `Input -> Conv 5x5 -> Pool 2x2 -> Conv 5x5 -> Pool 2x2 -> FC -> Output`



Свёрточные нейросети **(**простые**,** есть и намного более продвинутые**)** почти всегда строятся по следующему правилу**:**

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*L -> FC
```

то есть**:**

1). ***Входной слой*** **(batch** картинок **--** тензор размера `(batch_size, H, W, C)` **)**

2). M блоков **(M** $\geq$ **0)** из свёрток и **pooling-**ов**,** причём именно в том порядке**,** как в формуле выше**.** Все эти M блоков вместе называют ***feature extractor*** свёрточной нейросети**,** потому что эта часть сети отвечает непосредственно за формирование новых**,** более сложных признаков поверх тех**,** которые подаются **(**то есть**,** по аналогии с **MLP,** мы опять же переходим к новому признаковому пространству**,** однако здесь оно строится сложнее**,** чем в обычных многослойных сетях**,** поскольку используется операция свёртки**)**

3). L штук **FullyConnected-**слоёв **(**с активациями**).** Эту часть из L **FC-**слоёв называют ***classificator*****,** поскольку эти слои отвечают непосредственно за предсказание нужно класса **(**сейчас рассматривается задача классификации изображений**).**

Свёрточная нейросеть на **PyTorch**

Ешё раз напомним про основные компоненты нейросети**:**

- непосредственно**,** сама **архитектура** нейросети **(**сюда входят типы функций активации у каждого нейрона**);**
- начальная **инициализация** весов каждого слоя**;**
- метод **оптимизации** нейросети **(**сюда ещё входит метод изменения `learning_rate`**);**
- размер **батчей (**`batch_size`**);**
- количетсво **эпох** обучения **(**`num_epochs`**);**
- **функция потерь (**`loss`**);**
- тип **регуляризации** нейросети **(**для каждого слоя можно свой**);**

То**,** что связано с ***данными и задачей***:

- само **качество** выборки **(**непротиворечивость**,** чистота**,** корректность постановки задачи**);**
- **размер** выборки**;**

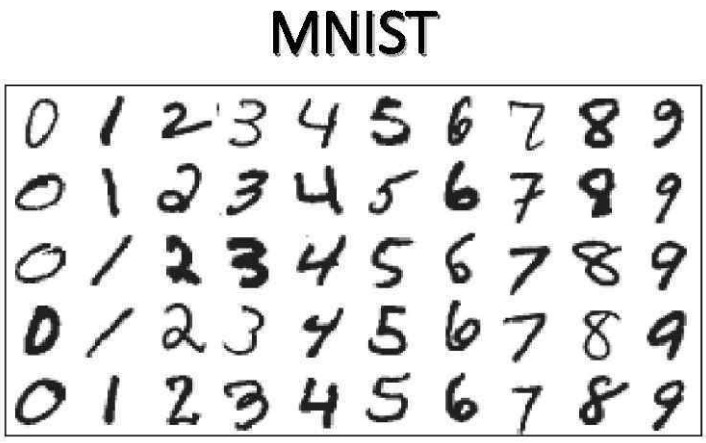
Так как мы сейчас рассматриваем **архитектуру CNN,** то**,** помимо этих компонент**,** в свёрточной нейросети можно настроить следующие вещи**:**

- **(**в каждом **ConvLayer) размер фильтров (окна свёртки) (**`kernel_size`**)**
- **(**в каждом **ConvLayer) количество фильтров (**`out_channels`**)**
- **(**в каждом **ConvLayer)** размер **шага окна свёртки (stride) (**`stride`**)**
- **(**в каждом **ConvLayer) тип padding'а (**`padding`**)**

- **(**в каждом **PoolLayer) размер окна pooling'а (**`kernel_size`**)**
- **(**в каждом **PoolLayer) шаг окна pooling'а (**`stride`**)**
- **(**в каждом **PoolLayer) тип pooling'а (**`pool_type`**)**
- **(**в каждом **PoolLayer) тип padding'а (**`padding`**)**

Какими их берут обычно **--** будет показано в примере ниже**.** По крайней мере**,** можно начинать с этих настроек**,** чтобы понять**,** какое качество "из коробки" будет у простой модели**.**

Посмотрим**,** как работает **CNN** на **MNIST**'е и на **CIFAR**'е**:**



MNIST: это набор из **70k** картинок рукописных цифр от **0** до **9,** написанных людьми**, 60k** из которых являются тренировочной выборкой **(**`train` **dataset)),** и ещё **10k** выделены для тестирования модели **(** `test` **dataset).**

In [3]:

```
#!pip install torch torchvision
```

In [4]:

```
import torch
import torchvision
from torchvision import transforms

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Скачаем и загрузим данные в `DataLoader`'ы:

Обратите внимание на аргумент `batch_size`: именно он будет отвечать за размер батча, который будет подаваться при оптимизации нейросети

In [5]:

```
transform = transforms.Compose(
    [transforms.ToTensor()])

trainset = torchvision.datasets.MNIST(root='./data', train=True,
                                      download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.MNIST(root='./data', train=False,
                                     download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = tuple(str(i) for i in range(10))
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data\MNIST\r
aw\train-images-idx3-ubyte.gz


Extracting ./data\MNIST\raw\train-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data\MNIST\r
aw\train-labels-idx1-ubyte.gz


Extracting ./data\MNIST\raw\train-labels-idx1-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data\MNIST\ra
w\t10k-images-idx3-ubyte.gz


Extracting ./data\MNIST\raw\t10k-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data\MNIST\ra
w\t10k-labels-idx1-ubyte.gz


Extracting ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw
```

Сами данные лежат в полях `trainloader.dataset.train_data` и `testloader.dataset.test_data`:

In [6]:

```
trainloader.dataset.train_data.shape
```

```
C:\Users\koshi8bit\anaconda3\lib\site-packages\torchvision\datasets\mnist.py:62: UserWarn
ing: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
```

Out[6]:

```
torch.Size([60000, 28, 28])
```

```
torch.Size([60000, 28, 28])
```

In [7]:

```
testloader.dataset.test_data.shape
```

```
C:\Users\koshi8bit\anaconda3\lib\site-packages\torchvision\datasets\mnist.py:67: UserWarn
ing: test_data has been renamed data
  warnings.warn("test_data has been renamed data")
```

Out[7]:

```
torch.Size([10000, 28, 28])
```

Выведем первую картинку:

In [8]:

```
trainloader.dataset.train_data[0]
```

Out[8]:

```
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,   0,   0,   0,   0,   0,   3,  18,
          18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
         253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253, 253,
         253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253, 253,
         198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253, 205,
          11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,  90,
           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, 139, 253, 190,
           2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190, 253,
          70,   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,  35, 241,
         225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  81,
         240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,  16,  93, 252, 253, 187,   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, 249, 253, 249,  64,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39, 148,
         229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221, 253,
         253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253, 253,
         253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253, 195,
          80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,  11,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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]],
       dtype=torch.uint8)
```

Посмотрим, как она выглядит:

In [9]:

```
# преобразовать тензор в np.array
numpy_img = trainloader.dataset.train_data[0].numpy()
```

In [10]:

```
trainloader.dataset.train_data[0]
```

Out[10]:

```
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,   0,   0,   0,   0,   0,   3,  18,
          18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
         253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253, 253,
         253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253, 253,
         198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253, 205,
          11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,  90,
           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, 139, 253, 190,
           2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190, 253,
          70,   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,  35, 241,
         225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  81,
         240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,  16,  93, 252, 253, 187,   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, 249, 253, 249,  64,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39, 148,
         229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221, 253,
         253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253, 253,
         253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253, 195,
          80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,  11,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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]],
      dtype=torch.uint8)
```

In [11]:

```
numpy_img.shape
```

Out[11]:

```
(28, 28)
```

In [12]:

```
numpy_img
```

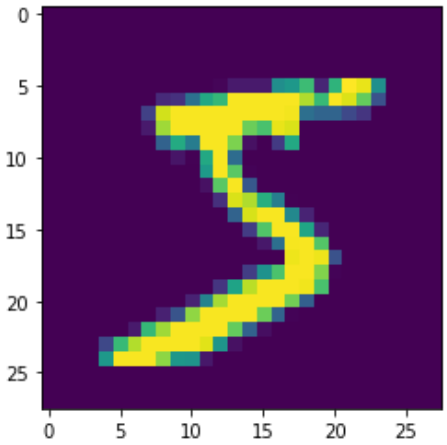
Out[12]:

```
array([[  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,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   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, 139, 253,
        190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   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,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   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, 249, 253, 249,  64,   0,   0,   0,   0,   0,
       0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
       0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
       0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
     148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
       0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
     253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
       0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
     253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
       0,   0],
      [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
     195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
       0,   0],
      [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
      11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
       0,   0],
      [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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]], dtype=uint8)
```

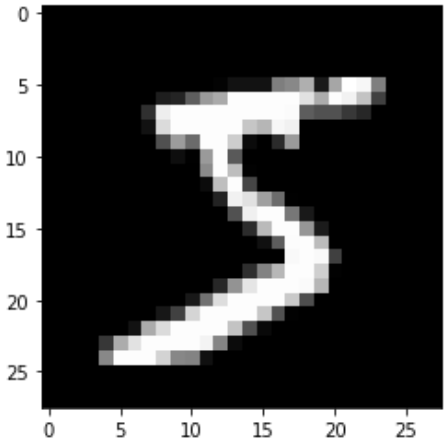
In [13]:

```
plt.imshow(numpy_img);
```



In [14]:

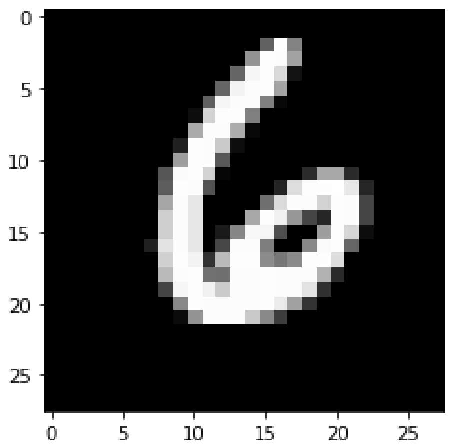
```
plt.imshow(numpy_img, cmap='gray');
```



Отрисовка заданной цифры:

In [15]:

```python
# случайный индекс от 0 до размера тренировочной выборки
i = np.random.randint(low=0, high=60000)

plt.imshow(trainloader.dataset.train_data[i].numpy(), cmap='gray');
```



Как итерироваться по данным с помощью `loader'`а? Очень просто:

In [16]:

```python
i=0
for data in trainloader:
    print(len(data))
    print('Images:',data[0])
    print('Labels:', data[1])
    #i = i+1
    #if i==5:
    break
```

```
2
Images: 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.]]]])
Labels: tensor([5, 5, 5, 2])
```

То есть мы имеем дело с кусочками данных размера **batch_size (**в данном случае **= 4),** причём в каждом батче есть как объекты**,** так и ответы на них **(**то есть и X**,** и y**).**

Теперь вернёмся к тому**,** что в **PyTorch** есть две "парадигмы" построения нейросетей **--** `Functional` и `Seuquential`**.** Со второй мы уже хорошенько разобрались в предыдущих ноутбуках по нейросетям**,** теперь мы используем именно `Functional` парадигму**,** потому что при построении свёрточных сетей это намного удобнее**:**

In [17]:

```
import torch.nn as nn
import torch.nn.functional as F  # Functional
```

In [18]:

```
# Заметьте: класс наследуется от nn.Module
class SimpleConvNet(nn.Module):
    def __init__(self):
        # вызов конструктора предка
        super(SimpleConvNet, self).__init__()
        # необходмо заранее знать, сколько каналов у картинки (сейчас = 1),
        # которую будем подавать в сеть, больше ничего
        # про входящие картинки знать не нужно
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.fc1 = nn.Linear(4 * 4 * 16, 120)  # !!!
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        # print(x.shape)
        x = x.view(-1, 4 * 4 * 16)  # !!!
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Важное примечание: Вы можете заметить**,** что в строчках с `#!!!` есть не очень понятное сходу число `4 * 4 * 16`**.** Это **--** размерность тензора перед **FC-**слоями **(H x W x C),** тут её приходиться высчитывать вручную **(**в **Keras,** например**,** `.Flatten()` всё делает за Вас**).** Однако есть один *лайфхак* **--** просто сделайте в `forward()` `print(x.shape)` **(**закомментированная строка**).** Вы увидите размер `(batch_size, C, H, W)` **--** нужно перемножить все**,** кроме первого **(batch_size),** это и будет первая размерность `Linear()`**,** и именно в `C * H * W` нужно "развернуть" **x** перед подачей в `Linear()`**.**

То есть нужно будет запустить цикл с обучением первый раз с `print()` и сделать после него `break`**,** посчитать размер**,** вписать его в нужные места и стереть `print()` и `break`**.**

Код обучения слоя**:**

In [19]:

```
from tqdm import tqdm_notebook
```

In [20]:

```
# объявляем сеть
net = SimpleConvNet()
```

```
# выбираем функцию потерь
loss_fn = torch.nn.CrossEntropyLoss()

# выбираем алгоритм оптимизации и learning_rate
learning_rate = 1e-4
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

# итерируемся
for epoch in tqdm_notebook(range(3)):

    running_loss = 0.0
    for i, batch in enumerate(tqdm_notebook(trainloader)):
        # так получаем текущий батч
        X_batch, y_batch = batch

        # обнуляем веса
        optimizer.zero_grad()

        # forward + backward + optimize
        y_pred = net(X_batch)
        loss = loss_fn(y_pred, y_batch)
        loss.backward()
        optimizer.step()

        # выведем текущий loss
        running_loss += loss.item()
        # выведем качество каждые 2000 батчей
        if i % 2000 == 1999:
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Обучение закончено')
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/1038078463.py:12: TqdmDeprecationWarn
ing: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(3)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/1038078463.py:15: TqdmDeprecationWarn
ing: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 0.926
[1,  4000] loss: 0.351
[1,  6000] loss: 0.270
[1,  8000] loss: 0.242
[1, 10000] loss: 0.205
[1, 12000] loss: 0.180
[1, 14000] loss: 0.151

[2,  2000] loss: 0.140
[2,  4000] loss: 0.124
[2,  6000] loss: 0.132
[2,  8000] loss: 0.118
[2, 10000] loss: 0.101
[2, 12000] loss: 0.104
[2, 14000] loss: 0.100

[3,  2000] loss: 0.094
[3,  4000] loss: 0.091
[3,  6000] loss: 0.080
[3,  8000] loss: 0.100
[3, 10000] loss: 0.082
[3, 12000] loss: 0.077
[3, 14000] loss: 0.073
Обучение закончено
```

Протестируем на всём тестовом датасете, используя метрику **accuracy_score:**

In [21]:

```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

with torch.no_grad():
    for data in testloader:
        images, labels = data
        y_pred = net(images)
        _, predicted = torch.max(y_pred, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of     0 : 99 %
Accuracy of     1 : 99 %
Accuracy of     2 : 98 %
Accuracy of     3 : 97 %
Accuracy of     4 : 98 %
Accuracy of     5 : 98 %
Accuracy of     6 : 97 %
Accuracy of     7 : 95 %
Accuracy of     8 : 96 %
Accuracy of     9 : 96 %
```

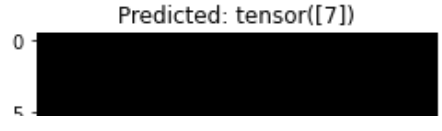
Два свёрточных слоя победили многослойную нейросеть (из ноутбука с домашним заданием). Это показывает эффективность применения операции свёртки при работе с изображениями.

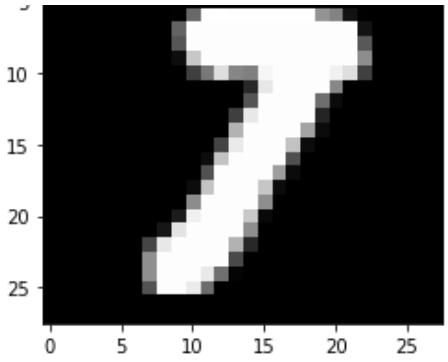
Протестируем эту нейросеть на отдельных картинках из тестового датасета: напишием функцию, которая принимает индекс картинки в тестовом датасете, отрисовывает её, потом запускает на ней модель (нейросеть) и выводит результат предсказания.

In [25]:

```
i = np.random.randint(low=0, high=10000)

def visualize_result(index):
    image = testloader.dataset.test_data[index].numpy()
    plt.imshow(image, cmap='gray')

    y_pred = net(torch.Tensor(image).view(1, 1, 28, 28))
    print(y_pred)
    pred, predicted = torch.max(y_pred, 1)
    print(pred)

    #batch = testloader.dataset.test_data[index]
    #print(batch[1])
    #loss = loss_fn(y_pred, y_batch)

    plt.title(f'Predicted: {predicted}')

visualize_result(i)
```

```
tensor([[ -728.0153,  -117.8322,   333.3349,   918.8973, -2620.1575,  -914.6388,
         -4951.9702,  2141.2395,   151.9646,   -19.3077]],
       grad_fn=<AddmmBackward0>)
tensor([2141.2395], grad_fn=<MaxBackward0>)
```





Можете запускать ячейку выше много раз **(**нажимая **Ctrl+Enter)** и видеть**,** что предсказывает нейросеть в зависимости от поданной на вход картинки**.**

Полезные ссылки

1). *Примеры написания нейросетей на* ***PyTorch*** *(офийиальные туториалы)* ***(****на английском****):***
https://pytorch.org/tutorials/beginner/pytorch_with_examples.html#examples
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

2). Курс Стэнфорда**:** **http://cs231n.github.io/**

3). Практически исчерпывающая информация по основам свёрточных нейросетей **(**из **cs231n) (**на английском**):**

http://cs231n.github.io/convolutional-networks/
http://cs231n.github.io/understanding-cnn/
http://cs231n.github.io/transfer-learning/

4). Видео о **Computer Vision** от **Andrej Karpathy:** **https://www.youtube.com/watch?v=u6aEYuemt0M**

In [33]:

```
def check_network(net):
  class_correct = list(0. for i in range(10))
  class_total = list(0. for i in range(10))

  with torch.no_grad():
      for data in testloader:
          images, labels = data
          y_pred = net(images)
          _, predicted = torch.max(y_pred, 1)
          c = (predicted == labels).squeeze()
          for i in range(4):
              label = labels[i]
              class_correct[label] += c[i].item()
              class_total[label] += 1

  for i in range(10):
      print('Accuracy of %2s : %2d %%' % (
          classes[i], 100 * class_correct[i] / class_total[i]))

  class_correct_t = sum(class_correct)
  class_total_t = sum(class_total)

  print('Средняя точность:', (100. * class_correct_t / class_total_t))
```

In [ ]:

```
def train(nett, learning_rate = 1e-4, num_epochs = 3):
    loss_fn = torch.nn.CrossEntropyLoss()
    tmp = []
    optimizer = torch.optim.Adam(nett.parameters(), lr=learning_rate)
    # итерируемся
    for epoch in tqdm_notebook(range(num_epochs)):
```

```
        running_loss = 0.0
        for i, batch in enumerate(tqdm_notebook(trainloader)):
            # так получаем текущий батч
            X_batch, y_batch = batch

            # обнуляем веса
            optimizer.zero_grad()

            # forward + backward + optimize
            y_pred = nett(X_batch)
            loss = loss_fn(y_pred, y_batch)
            loss.backward()
            optimizer.step()

            # выведем текущий loss
            running_loss += loss.item()
            # выведем качество каждые 2000 батчей
            if i % 2000 == 1999:
                tmp.append(running_loss / 2000)
                print('[%d, %5d] loss: %.3f' %
                      (epoch + 1, i + 1, running_loss / 2000))
                running_loss = 0.0

        x = np.array(range(len(tmp)))
        y = np.array(tmp)
        print(x, y)
        plt.plot(x, y)
        plt.show()
    print('OK')
```

In [58]:

```
import matplotlib.pyplot as plt
import numpy as np

# класс наследуется от nn.Module
class SimpleConvNet1(nn.Module):
    def __init__(self, channels1, channels2, kernel_size1, kernel_size2, is_max_pool = T
rue, f=F.relu):
        # вызов конструктора предка
        super(SimpleConvNet1, self).__init__()
        self.f = f
        # необходмо заранее знать, сколько каналов у картинки (сейчас = 1),
        # которую будем подавать в сеть, больше ничего
        # про входящие картинки знать не нужно
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=channels1, kernel_size=5)
        new_size = 28 - kernel_size1 + 1
        if is_max_pool:
          self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        else:
          self.pool = nn.AvgPool2d(kernel_size=2, stride=2)
        new_size = new_size // 2
        self.conv2 = nn.Conv2d(in_channels=channels1, out_channels=channels2, kernel_siz
e=5)
        new_size = new_size - kernel_size2 + 1
        new_size = new_size // 2

        self.fc1_size = new_size * new_size * channels2

        self.fc1 = nn.Linear(new_size * new_size * channels2, 120)  # !!!
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(self.f(self.conv1(x)))
        #print(x.shape)
        x = self.pool(self.f(self.conv2(x)))
        #print(x.shape)
        x = x.view(-1, self.fc1_size)  # !!!
        x = self.f(self.fc1(x))
        x = self.f(self.fc2(x))
```

```
        x = self.fc3(x)
        return x
```

Пробую **MaxPool2d** с **3** эпохами и ядрами **5, 5**

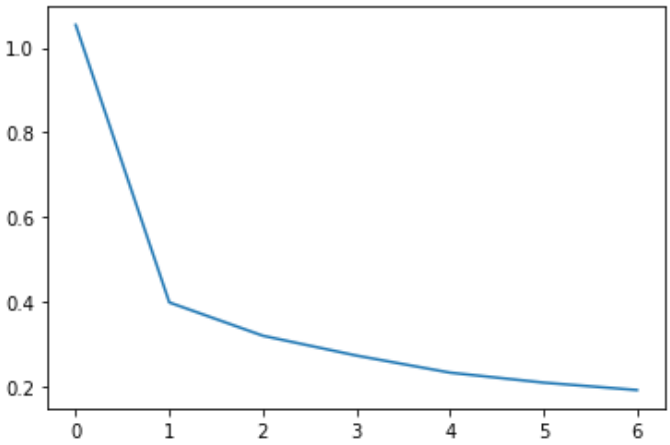
In [59]:

```
net = SimpleConvNet1(6, 16, 5, 5, True)
train(net, num_epochs=1)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2944825031.py:46: TqdmDeprecationWarn
ing: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2944825031.py:48: TqdmDeprecationWarn
ing: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 1.054
[1,  4000] loss: 0.398
[1,  6000] loss: 0.319
[1,  8000] loss: 0.273
[1, 10000] loss: 0.232
[1, 12000] loss: 0.209
[1, 14000] loss: 0.191
[0 1 2 3 4 5 6] [1.05385665 0.39821254 0.31948146 0.272645   0.23228549 0.20869579
 0.19111498]
```




```
OK
Accuracy of  0 : 98 %
Accuracy of  1 : 98 %
Accuracy of  2 : 96 %
Accuracy of  3 : 94 %
Accuracy of  4 : 97 %
Accuracy of  5 : 96 %
Accuracy of  6 : 95 %
Accuracy of  7 : 94 %
Accuracy of  8 : 94 %
Accuracy of  9 : 93 %
Средняя точность: 96.03
```

Результат удалось повторить, в целом не плохо. Попробуем заменить ***MaxPool2d*** на ***AvgPool2d***

In [36]:

```
net = SimpleConvNet1(6, 16, 5, 5, False)
train(net)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/324471750.py:41: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
```

```
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/324471750.py:43: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 1.088
[1,  4000] loss: 0.480
[1,  6000] loss: 0.406
[1,  8000] loss: 0.350
[1, 10000] loss: 0.301
[1, 12000] loss: 0.268
[1, 14000] loss: 0.237

[2,  2000] loss: 0.199
[2,  4000] loss: 0.187
[2,  6000] loss: 0.163
[2,  8000] loss: 0.163
[2, 10000] loss: 0.148
[2, 12000] loss: 0.146
[2, 14000] loss: 0.127

[3,  2000] loss: 0.116
[3,  4000] loss: 0.118
[3,  6000] loss: 0.108
[3,  8000] loss: 0.113
[3, 10000] loss: 0.112
[3, 12000] loss: 0.103
[3, 14000] loss: 0.101
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 97 %
Accuracy of  3 : 97 %
Accuracy of  4 : 99 %
Accuracy of  5 : 96 %
Accuracy of  6 : 98 %
Accuracy of  7 : 96 %
Accuracy of  8 : 95 %
Accuracy of  9 : 93 %
Средняя точность: 97.33
```

Точность упала на **1%.** Пробуем дальше. Меняем размер ядра

In [38]:

```
net = SimpleConvNet1(6, 16, 7, 3, True)
train(net)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/324471750.py:41: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/324471750.py:43: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 1.100
[1,  4000] loss: 0.411
[1,  6000] loss: 0.325
[1,  8000] loss: 0.268
[1, 10000] loss: 0.217
[1, 12000] loss: 0.197
[1, 14000] loss: 0.172
```

```
[2,  2000] loss: 0.153
[2,  4000] loss: 0.152
[2,  6000] loss: 0.125
[2,  8000] loss: 0.120
[2, 10000] loss: 0.123
[2, 12000] loss: 0.113
[2, 14000] loss: 0.112

[3,  2000] loss: 0.100
[3,  4000] loss: 0.094
[3,  6000] loss: 0.100
[3,  8000] loss: 0.080
[3, 10000] loss: 0.096
[3, 12000] loss: 0.078
[3, 14000] loss: 0.081
OK
Accuracy of  0 : 98 %
Accuracy of  1 : 99 %
Accuracy of  2 : 98 %
Accuracy of  3 : 97 %
Accuracy of  4 : 98 %
Accuracy of  5 : 98 %
Accuracy of  6 : 98 %
Accuracy of  7 : 98 %
Accuracy of  8 : 97 %
Accuracy of  9 : 95 %
Средняя точность: 98.12
```

Улучшили результат на **0.1%** Пробую разные функции активации

In [39]:

```
functions = [F.relu, F.elu, F.softsign]
```

In [42]:

```
for f in functions:
    net = SimpleConvNet1(6, 16, 7, 3, True, f=f)
    train(net)
    check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/3625193883.py:42: TqdmDeprecationWarn
ing: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/3625193883.py:44: TqdmDeprecationWarn
ing: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 1.090
[1,  4000] loss: 0.395
[1,  6000] loss: 0.292
[1,  8000] loss: 0.261
[1, 10000] loss: 0.217
[1, 12000] loss: 0.182
[1, 14000] loss: 0.160

[2,  2000] loss: 0.143
[2,  4000] loss: 0.136
[2,  6000] loss: 0.126
[2,  8000] loss: 0.110
[2, 10000] loss: 0.105
[2, 12000] loss: 0.108
[2, 14000] loss: 0.087
```

```
[3,  2000] loss: 0.091
[3,  4000] loss: 0.084
[3,  6000] loss: 0.087
[3,  8000] loss: 0.082
[3, 10000] loss: 0.078
[3, 12000] loss: 0.078
[3, 14000] loss: 0.076
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 98 %
Accuracy of  3 : 98 %
Accuracy of  4 : 98 %
Accuracy of  5 : 96 %
Accuracy of  6 : 96 %
Accuracy of  7 : 97 %
Accuracy of  8 : 98 %
Accuracy of  9 : 96 %
Средняя точность: 98.01


[1,  2000] loss: 0.899
[1,  4000] loss: 0.331
[1,  6000] loss: 0.258
[1,  8000] loss: 0.216
[1, 10000] loss: 0.178
[1, 12000] loss: 0.155
[1, 14000] loss: 0.146

[2,  2000] loss: 0.128
[2,  4000] loss: 0.119
[2,  6000] loss: 0.121
[2,  8000] loss: 0.106
[2, 10000] loss: 0.087
[2, 12000] loss: 0.094
[2, 14000] loss: 0.086

[3,  2000] loss: 0.081
[3,  4000] loss: 0.083
[3,  6000] loss: 0.075
[3,  8000] loss: 0.068
[3, 10000] loss: 0.072
[3, 12000] loss: 0.068
[3, 14000] loss: 0.066
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 97 %
Accuracy of  2 : 99 %
Accuracy of  3 : 98 %
Accuracy of  4 : 97 %
Accuracy of  5 : 98 %
Accuracy of  6 : 98 %
Accuracy of  7 : 95 %
Accuracy of  8 : 98 %
Accuracy of  9 : 96 %
Средняя точность: 97.79


[1,  2000] loss: 1.343
[1,  4000] loss: 0.454
[1,  6000] loss: 0.287
[1,  8000] loss: 0.230
[1, 10000] loss: 0.181
[1, 12000] loss: 0.163
[1, 14000] loss: 0.142

[2,  2000] loss: 0.122
[2,  4000] loss: 0.111
[2,  6000] loss: 0.109
[2,  8000] loss: 0.101
[2, 10000] loss: 0.091
```

```
[2, 12000] loss: 0.096
[2, 14000] loss: 0.084

[3,  2000] loss: 0.082
[3,  4000] loss: 0.077
[3,  6000] loss: 0.077
[3,  8000] loss: 0.080
[3, 10000] loss: 0.067
[3, 12000] loss: 0.065
[3, 14000] loss: 0.063
OK
Accuracy of  0 : 98 %
Accuracy of  1 : 99 %
Accuracy of  2 : 97 %
Accuracy of  3 : 97 %
Accuracy of  4 : 98 %
Accuracy of  5 : 98 %
Accuracy of  6 : 98 %
Accuracy of  7 : 97 %
Accuracy of  8 : 96 %
Accuracy of  9 : 95 %
Средняя точность: 97.92
```

Лучший результат **98.01.** Не сильно это помогло Попробуем менять связи в полносвязных слоях

In [61]:

```
class SimpleConvNet2(nn.Module):
    def __init__(self, channels1, channels2, kernel_size1, kernel_size2, fc1, fc2, is_ma
x_pool = True):
        # вызов конструктора предка
        super(SimpleConvNet2, self).__init__()
        # необходмо заранее знать, сколько каналов у картинки (сейчас = 1),
        # которую будем подавать в сеть, больше ничего
        # про входящие картинки знать не нужно
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=channels1, kernel_size=kernel
_size1)
        new_size = 28 - kernel_size1 + 1
        if is_max_pool:
          self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        else:
          self.pool = nn.AvgPool2d(kernel_size=2, stride=2)
        new_size = new_size // 2
        self.conv2 = nn.Conv2d(in_channels=channels1, out_channels=channels2, kernel_siz
e=kernel_size2)
        new_size = new_size - kernel_size2 + 1
        new_size = new_size // 2

        self.fc1_size = new_size * new_size * channels2

        self.fc1 = nn.Linear(new_size * new_size * channels2, fc1)  # !!!
        self.fc2 = nn.Linear(fc1, fc2)
        self.fc3 = nn.Linear(fc2, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        #print(x.shape)
        x = self.pool(F.relu(self.conv2(x)))
        #print(x.shape)
        x = x.view(-1, self.fc1_size)  # !!!
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

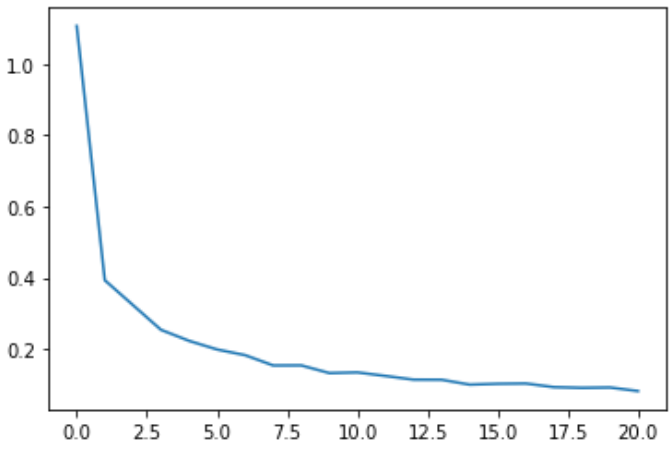
In [63]:

```
net = SimpleConvNet2(6, 16, 7, 3, 128, 64, True)
train(net)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 1.106
[1,  4000] loss: 0.393
[1,  6000] loss: 0.323
[1,  8000] loss: 0.254
[1, 10000] loss: 0.223
[1, 12000] loss: 0.199
[1, 14000] loss: 0.183

[2,  2000] loss: 0.154
[2,  4000] loss: 0.154
[2,  6000] loss: 0.132
[2,  8000] loss: 0.134
[2, 10000] loss: 0.124
[2, 12000] loss: 0.114
[2, 14000] loss: 0.114

[3,  2000] loss: 0.100
[3,  4000] loss: 0.102
[3,  6000] loss: 0.103
[3,  8000] loss: 0.093
[3, 10000] loss: 0.091
[3, 12000] loss: 0.092
[3, 14000] loss: 0.082
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20] [1.10635867 0.39284468 0
.32334759 0.25381277 0.22282145 0.19855838
 0.18259483 0.15374522 0.15392724 0.13232191 0.13417696 0.12424648
 0.11393988 0.1136172  0.09990608 0.10225069 0.1030757  0.0928519
 0.09093168 0.09185333 0.0818762 ]
```




```
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 98 %
Accuracy of  3 : 96 %
Accuracy of  4 : 98 %
Accuracy of  5 : 97 %
Accuracy of  6 : 96 %
Accuracy of  7 : 95 %
Accuracy of  8 : 98 %
Accuracy of  9 : 96 %
Средняя точность: 97.8
```

In [ ]:

Лучше не стало. Пробуем дообучить? тк хорошая скорость падения loss

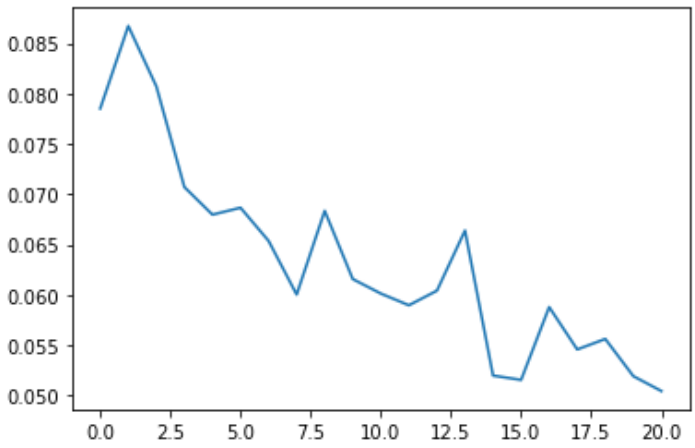
In [64]:

```
train(net)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 0.079
[1,  4000] loss: 0.087
[1,  6000] loss: 0.081
[1,  8000] loss: 0.071
[1, 10000] loss: 0.068
[1, 12000] loss: 0.069
[1, 14000] loss: 0.065

[2,  2000] loss: 0.060
[2,  4000] loss: 0.068
[2,  6000] loss: 0.062
[2,  8000] loss: 0.060
[2, 10000] loss: 0.059
[2, 12000] loss: 0.060
[2, 14000] loss: 0.066

[3,  2000] loss: 0.052
[3,  4000] loss: 0.052
[3,  6000] loss: 0.059
[3,  8000] loss: 0.055
[3, 10000] loss: 0.056
[3, 12000] loss: 0.052
[3, 14000] loss: 0.050
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20] [0.07850639 0.08670993 0
.08070727 0.07069688 0.06796322 0.06865135
 0.06535694 0.06002488 0.06834221 0.06156231 0.06012347 0.05895763
 0.06041084 0.06638558 0.0519699  0.05154478 0.05877975 0.05456855
 0.05561262 0.05191402 0.05043515]
```




```
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 99 %
Accuracy of  3 : 98 %
Accuracy of  4 : 96 %
Accuracy of  5 : 98 %
Accuracy of  6 : 98 %
Accuracy of  7 : 99 %
```

```
Accuracy of  8 : 97 %
Accuracy of  9 : 98 %
Средняя точность: 98.5
```

ПРостое дообучение дало прирост в **0.8%.** Пропробую втупую дообучить

In [65]:

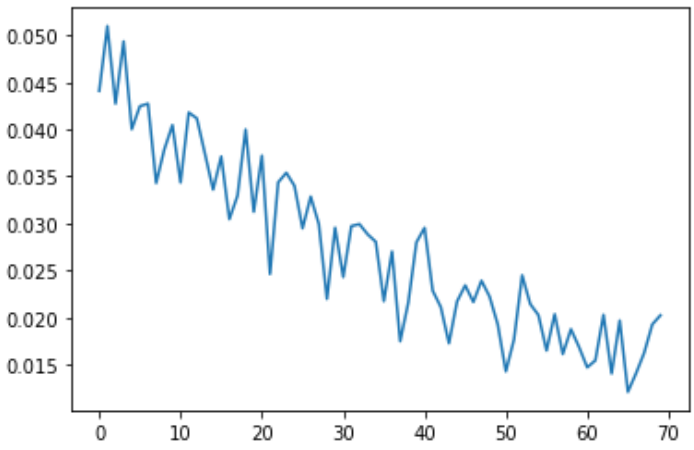
```
train(net, num_epochs = 10)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 0.044
[1,  4000] loss: 0.051
[1,  6000] loss: 0.043
[1,  8000] loss: 0.049
[1, 10000] loss: 0.040
[1, 12000] loss: 0.042
[1, 14000] loss: 0.043

[2,  2000] loss: 0.034
[2,  4000] loss: 0.038
[2,  6000] loss: 0.040
[2,  8000] loss: 0.034
[2, 10000] loss: 0.042
[2, 12000] loss: 0.041
[2, 14000] loss: 0.037

[3,  2000] loss: 0.034
[3,  4000] loss: 0.037
[3,  6000] loss: 0.030
[3,  8000] loss: 0.033
[3, 10000] loss: 0.040
[3, 12000] loss: 0.031
[3, 14000] loss: 0.037

[4,  2000] loss: 0.025
[4,  4000] loss: 0.034
[4,  6000] loss: 0.035
[4,  8000] loss: 0.034
[4, 10000] loss: 0.029
[4, 12000] loss: 0.033
[4, 14000] loss: 0.030

[5,  2000] loss: 0.022
[5,  4000] loss: 0.030
[5,  6000] loss: 0.024
[5,  8000] loss: 0.030
[5, 10000] loss: 0.030
[5, 12000] loss: 0.029
[5, 14000] loss: 0.028

[6,  2000] loss: 0.022
[6,  4000] loss: 0.027
[6,  6000] loss: 0.017
[6,  8000] loss: 0.022
[6, 10000] loss: 0.028
[6, 12000] loss: 0.030
[6, 14000] loss: 0.023
```

```
[7,  2000] loss: 0.021
[7,  4000] loss: 0.017
[7,  6000] loss: 0.022
[7,  8000] loss: 0.023
[7, 10000] loss: 0.022
[7, 12000] loss: 0.024
[7, 14000] loss: 0.022

[8,  2000] loss: 0.019
[8,  4000] loss: 0.014
[8,  6000] loss: 0.018
[8,  8000] loss: 0.024
[8, 10000] loss: 0.021
[8, 12000] loss: 0.020
[8, 14000] loss: 0.016

[9,  2000] loss: 0.020
[9,  4000] loss: 0.016
[9,  6000] loss: 0.019
[9,  8000] loss: 0.017
[9, 10000] loss: 0.015
[9, 12000] loss: 0.015
[9, 14000] loss: 0.020

[10,  2000] loss: 0.014
[10,  4000] loss: 0.020
[10,  6000] loss: 0.012
[10,  8000] loss: 0.014
[10, 10000] loss: 0.016
[10, 12000] loss: 0.019
[10, 14000] loss: 0.020
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69] [0.04409257 0.05099063
0.0427406  0.04936849 0.04001918 0.0424734
 0.04273683 0.03429136 0.03787285 0.04045789 0.03434446 0.04179652
 0.04117822 0.03740706 0.03359533 0.03711719 0.03044917 0.03295186
 0.03998768 0.03123068 0.03721236 0.02461642 0.0343638  0.03538808
 0.03400558 0.02948091 0.0328534  0.02993437 0.02196312 0.02954811
 0.02429571 0.02971814 0.02992216 0.02883697 0.0280494  0.02172936
 0.02701729 0.01746554 0.02157554 0.02798745 0.02952916 0.02284442
 0.02109808 0.01725003 0.02170738 0.02341155 0.02164072 0.02390157
 0.0222117  0.01924641 0.01426012 0.01766097 0.02448276 0.02141378
 0.02025127 0.01647468 0.02034715 0.01610799 0.0187545  0.01683904
 0.01468666 0.01542503 0.0202825  0.01403485 0.01966809 0.01207773
 0.01404645 0.01622196 0.01925331 0.02022591]
```




```
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 99 %
Accuracy of  3 : 99 %
Accuracy of  4 : 98 %
Accuracy of  5 : 98 %
```

```
Accuracy of  6 : 98 %
Accuracy of  7 : 99 %
Accuracy of  8 : 98 %
Accuracy of  9 : 97 %
Средняя точность: 99.0
```

Вау! **99%** пробую еще

In [66]:

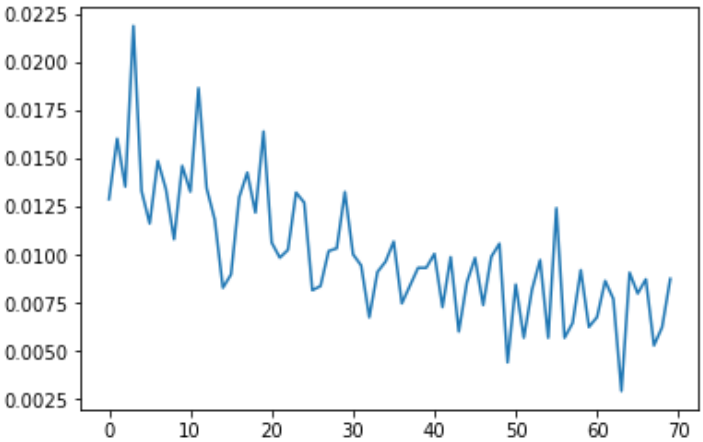
```
train(net, num_epochs = 10)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 0.013
[1,  4000] loss: 0.016
[1,  6000] loss: 0.014
[1,  8000] loss: 0.022
[1, 10000] loss: 0.013
[1, 12000] loss: 0.012
[1, 14000] loss: 0.015

[2,  2000] loss: 0.013
[2,  4000] loss: 0.011
[2,  6000] loss: 0.015
[2,  8000] loss: 0.013
[2, 10000] loss: 0.019
[2, 12000] loss: 0.013
[2, 14000] loss: 0.012

[3,  2000] loss: 0.008
[3,  4000] loss: 0.009
[3,  6000] loss: 0.013
[3,  8000] loss: 0.014
[3, 10000] loss: 0.012
[3, 12000] loss: 0.016
[3, 14000] loss: 0.011

[4,  2000] loss: 0.010
[4,  4000] loss: 0.010
[4,  6000] loss: 0.013
[4,  8000] loss: 0.013
[4, 10000] loss: 0.008
[4, 12000] loss: 0.008
[4, 14000] loss: 0.010

[5,  2000] loss: 0.010
[5,  4000] loss: 0.013
[5,  6000] loss: 0.010
[5,  8000] loss: 0.009
[5, 10000] loss: 0.007
[5, 12000] loss: 0.009
[5, 14000] loss: 0.010

[6,  2000] loss: 0.011
[6,  4000] loss: 0.007
[6,  6000] loss: 0.008
[6,  8000] loss: 0.009
[6, 10000] loss: 0.009
```

```
[6, 12000] loss: 0.010
[6, 14000] loss: 0.007

[7,  2000] loss: 0.010
[7,  4000] loss: 0.006
[7,  6000] loss: 0.009
[7,  8000] loss: 0.010
[7, 10000] loss: 0.007
[7, 12000] loss: 0.010
[7, 14000] loss: 0.011

[8,  2000] loss: 0.004
[8,  4000] loss: 0.008
[8,  6000] loss: 0.006
[8,  8000] loss: 0.008
[8, 10000] loss: 0.010
[8, 12000] loss: 0.006
[8, 14000] loss: 0.012

[9,  2000] loss: 0.006
[9,  4000] loss: 0.006
[9,  6000] loss: 0.009
[9,  8000] loss: 0.006
[9, 10000] loss: 0.007
[9, 12000] loss: 0.009
[9, 14000] loss: 0.008

[10,  2000] loss: 0.003
[10,  4000] loss: 0.009
[10,  6000] loss: 0.008
[10,  8000] loss: 0.009
[10, 10000] loss: 0.005
[10, 12000] loss: 0.006
[10, 14000] loss: 0.009
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69] [0.0128688  0.0160137
0.01352539 0.02186831 0.01329501 0.01161155
 0.01486534 0.01338248 0.01078978 0.01460123 0.01325891 0.01864081
 0.01344423 0.01180948 0.00827098 0.00896169 0.01296181 0.014253
 0.01217645 0.01638659 0.01061779 0.00983549 0.01023887 0.01321368
 0.01269427 0.00814405 0.0083594  0.01017972 0.01031796 0.01324503
 0.01001038 0.00942968 0.00673874 0.00909342 0.00962821 0.01066985
 0.00746272 0.00838446 0.00930782 0.00930801 0.01003196 0.00727009
 0.00986624 0.00599932 0.00855421 0.00982489 0.00737465 0.00988318
 0.01056279 0.00439786 0.00844196 0.00567858 0.00817557 0.00970936
 0.00567337 0.01241052 0.00567654 0.00644982 0.00918948 0.00623491
 0.00674985 0.00863114 0.007695   0.00291212 0.00905763 0.00797626
 0.00871256 0.00527799 0.00622927 0.00873418]
```




```
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 98 %
Accuracy of  3 : 99 %
```

```
Accuracy of  3 : 99 %
Accuracy of  4 : 98 %
Accuracy of  5 : 98 %
Accuracy of  6 : 99 %
Accuracy of  7 : 99 %
Accuracy of  8 : 98 %
Accuracy of  9 : 98 %
Средняя точность: 98.93
```

Точность даже немного упала

Пробую изменить размер полнсвяз. слоев

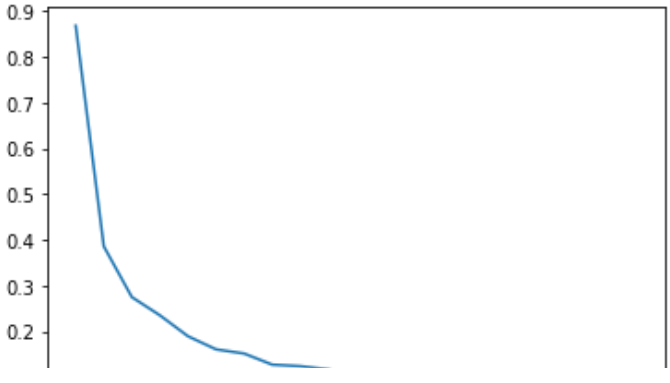
In [71]:

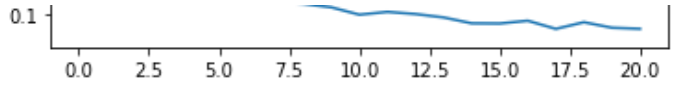
```
net = SimpleConvNet2(6, 16, 7, 3, 256, 128, True)
train(net)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 0.869
[1,  4000] loss: 0.386
[1,  6000] loss: 0.275
[1,  8000] loss: 0.235
[1, 10000] loss: 0.189
[1, 12000] loss: 0.161
[1, 14000] loss: 0.152

[2,  2000] loss: 0.127
[2,  4000] loss: 0.124
[2,  6000] loss: 0.117
[2,  8000] loss: 0.101
[2, 10000] loss: 0.107
[2, 12000] loss: 0.102
[2, 14000] loss: 0.095

[3,  2000] loss: 0.082
[3,  4000] loss: 0.081
[3,  6000] loss: 0.087
[3,  8000] loss: 0.070
[3, 10000] loss: 0.084
[3, 12000] loss: 0.072
[3, 14000] loss: 0.070
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20] [0.86866244 0.38603996 0
.27503126 0.23520501 0.18946835 0.16077025
 0.15170465 0.12736167 0.12411897 0.11672521 0.10071649 0.10679735
 0.10211379 0.09464966 0.08186841 0.08146559 0.08746888 0.06971234
 0.08424432 0.07221007 0.06983942]
```






```
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 96 %
Accuracy of  3 : 97 %
Accuracy of  4 : 99 %
Accuracy of  5 : 97 %
Accuracy of  6 : 97 %
Accuracy of  7 : 98 %
Accuracy of  8 : 97 %
Accuracy of  9 : 95 %
Средняя точность: 98.12
```

Пробую добавить сверточный слой

In [72]:

```
class SimpleConvNet3(nn.Module):
    def __init__(self, channels1, channels2, channels3, kernel_size1, kernel_size2, kern
el_size3, fc1, fc2, is_max_pool = True):
        # вызов конструктора предка
        super(SimpleConvNet3, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=1, out_channels=channels1, kernel_size=kernel
_size1)
        new_size = 28 - kernel_size1 + 1

        if is_max_pool:
          self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        else:
          self.pool = nn.AvgPool2d(kernel_size=2, stride=2)

        new_size = new_size // 2
        self.conv2 = nn.Conv2d(in_channels=channels1, out_channels=channels2, kernel_siz
e=kernel_size2)
        new_size = new_size - kernel_size2 + 1
        new_size = new_size // 2

        self.conv3 = nn.Conv2d(in_channels=channels2, out_channels=channels3, kernel_siz
e=kernel_size3)
        new_size = new_size - kernel_size3 + 1
        #new_size = new_size // 2
        #print(new_size)
        self.fc1_size = new_size * new_size * channels3

        self.fc1 = nn.Linear(new_size * new_size * channels3, fc1)  # !!!
        self.fc2 = nn.Linear(fc1, fc2)
        self.fc3 = nn.Linear(fc2, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        #print(x.shape)
        x = self.pool(F.relu(self.conv2(x)))
        #print(x.shape)
        x = F.relu(self.conv3(x))          #x = self.pool(F.relu(self.conv3(x)))
        #print(x.shape)
        x = x.view(-1, self.fc1_size)  # !!!
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

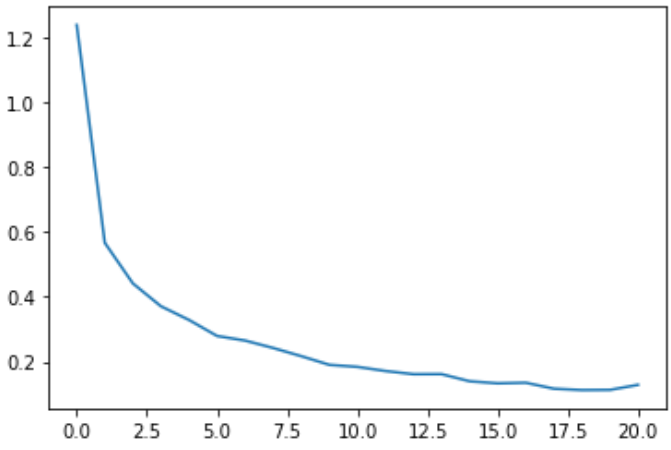
In [73]:

```
net = SimpleConvNet3(6, 16, 30, 7, 3, 3, 128, 64, True)
train(net)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 1.239
[1,  4000] loss: 0.567
[1,  6000] loss: 0.441
[1,  8000] loss: 0.371
[1, 10000] loss: 0.329
[1, 12000] loss: 0.279
[1, 14000] loss: 0.265

[2,  2000] loss: 0.242
[2,  4000] loss: 0.217
[2,  6000] loss: 0.190
[2,  8000] loss: 0.184
[2, 10000] loss: 0.171
[2, 12000] loss: 0.161
[2, 14000] loss: 0.161

[3,  2000] loss: 0.139
[3,  4000] loss: 0.133
[3,  6000] loss: 0.135
[3,  8000] loss: 0.117
[3, 10000] loss: 0.112
[3, 12000] loss: 0.113
[3, 14000] loss: 0.128
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20] [1.23858023 0.56655858 0
.44131596 0.37081455 0.32862795 0.27914481
 0.26503241 0.24208503 0.21692131 0.19015229 0.18385012 0.17104844
 0.16124916 0.16143719 0.13943753 0.13300354 0.13490764 0.11656329
 0.1123383  0.11285852 0.1283829 ]
```




```
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 96 %
Accuracy of  3 : 97 %
Accuracy of  4 : 98 %
Accuracy of  5 : 91 %
Accuracy of  6 : 97 %
Accuracy of  7 : 96 %
Accuracy of  8 : 95 %
Accuracy of  9 : 96 %
Средняя точность: 96.91
```

точность упала на **2%,** пробую дообучить

In [76]:

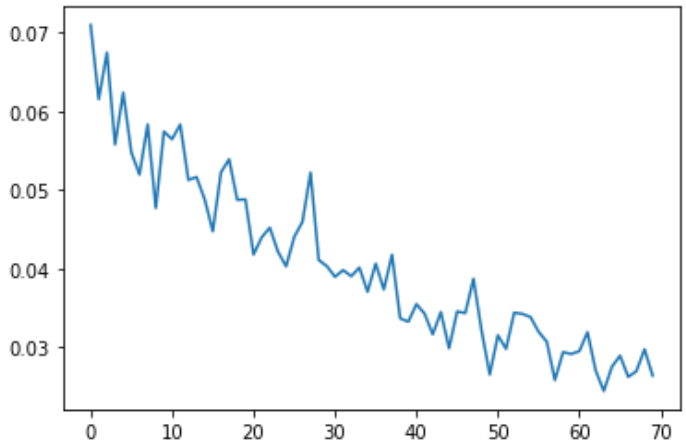
```
train(net, num_epochs = 10)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 0.071
[1,  4000] loss: 0.062
[1,  6000] loss: 0.067
[1,  8000] loss: 0.056
[1, 10000] loss: 0.062
[1, 12000] loss: 0.055
[1, 14000] loss: 0.052

[2,  2000] loss: 0.058
[2,  4000] loss: 0.048
[2,  6000] loss: 0.057
[2,  8000] loss: 0.056
[2, 10000] loss: 0.058
[2, 12000] loss: 0.051
[2, 14000] loss: 0.052

[3,  2000] loss: 0.049
[3,  4000] loss: 0.045
[3,  6000] loss: 0.052
[3,  8000] loss: 0.054
[3, 10000] loss: 0.049
[3, 12000] loss: 0.049
[3, 14000] loss: 0.042

[4,  2000] loss: 0.044
[4,  4000] loss: 0.045
[4,  6000] loss: 0.042
[4,  8000] loss: 0.040
[4, 10000] loss: 0.044
[4, 12000] loss: 0.046
[4, 14000] loss: 0.052

[5,  2000] loss: 0.041
[5,  4000] loss: 0.040
[5,  6000] loss: 0.039
[5,  8000] loss: 0.040
[5, 10000] loss: 0.039
[5, 12000] loss: 0.040
[5, 14000] loss: 0.037

[6,  2000] loss: 0.041
[6,  4000] loss: 0.037
[6,  6000] loss: 0.042
[6,  8000] loss: 0.034
[6, 10000] loss: 0.033
[6, 12000] loss: 0.035
[6, 14000] loss: 0.034

[7,  2000] loss: 0.032
[7,  4000] loss: 0.034
[7,  6000] loss: 0.030
[7,  8000] loss: 0.035
[7, 10000] loss: 0.034
```

```
[7, 12000] loss: 0.039
[7, 14000] loss: 0.032

[8,  2000] loss: 0.027
[8,  4000] loss: 0.032
[8,  6000] loss: 0.030
[8,  8000] loss: 0.034
[8, 10000] loss: 0.034
[8, 12000] loss: 0.034
[8, 14000] loss: 0.032

[9,  2000] loss: 0.031
[9,  4000] loss: 0.026
[9,  6000] loss: 0.029
[9,  8000] loss: 0.029
[9, 10000] loss: 0.029
[9, 12000] loss: 0.032
[9, 14000] loss: 0.027

[10,  2000] loss: 0.024
[10,  4000] loss: 0.028
[10,  6000] loss: 0.029
[10,  8000] loss: 0.026
[10, 10000] loss: 0.027
[10, 12000] loss: 0.030
[10, 14000] loss: 0.026
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69] [0.07098273 0.0615563
0.0674799  0.05577956 0.06238038 0.05476965
 0.05195832 0.05833546 0.0476975  0.05743408 0.05646132 0.05831288
 0.05127378 0.05164718 0.04881592 0.04473736 0.05224669 0.05390148
 0.04874497 0.04879877 0.04179325 0.04396735 0.0452037  0.04214395
 0.04027956 0.04403403 0.0459311  0.05220658 0.04109214 0.04030735
 0.03895394 0.03979319 0.0390327  0.04012109 0.03704846 0.04062098
 0.03734376 0.04176516 0.03366443 0.03323372 0.03548234 0.03424979
 0.03163969 0.03446982 0.02986458 0.03453073 0.03434423 0.03870152
 0.03209493 0.02652187 0.03151029 0.02979314 0.03438224 0.03423421
 0.03383337 0.03195983 0.03069119 0.02582005 0.02936381 0.02913485
 0.02948883 0.0319008  0.02703644 0.02443277 0.02751082 0.02892181
 0.02622109 0.02693816 0.02973699 0.02638084]
```




```
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 98 %
Accuracy of  3 : 99 %
Accuracy of  4 : 99 %
Accuracy of  5 : 98 %
Accuracy of  6 : 99 %
Accuracy of  7 : 98 %
Accuracy of  8 : 98 %
Accuracy of  9 : 98 %
Средняя точность: 98.93
```

Пробую добавить дропаут, может хоть в этой лабе он поможет

In [81]:

```
from torch.nn import Dropout
# класс наследуется от nn.Module
class SimpleConvNet4(nn.Module):
    def __init__(self, channels1, channels2, kernel_size1, kernel_size2, fc1, dropout, i
s_max_pool = True):
        # вызов конструктора предка
        super(SimpleConvNet4, self).__init__()
        # необходмо заранее знать, сколько каналов у картинки (сейчас = 1),
        # которую будем подавать в сеть, больше ничего
        # про входящие картинки знать не нужно
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=channels1, kernel_size=kernel
_size1)
        new_size = 28 - kernel_size1 + 1
        if is_max_pool:
            self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        else:
            self.pool = nn.AvgPool2d(kernel_size=2, stride=2)
        new_size = new_size // 2
        self.conv2 = nn.Conv2d(in_channels=channels1, out_channels=channels2, kernel_siz
e=kernel_size2)
        new_size = new_size - kernel_size2 + 1
        new_size = new_size // 2

        self.fc1_size = new_size * new_size * channels2

        self.fc1 = nn.Linear(new_size * new_size * channels2, fc1)  # !!!
        self.fc3 = nn.Linear(fc1, 10)

        self.dropout1 = Dropout(dropout)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        #print(x.shape)
        x = self.pool(F.relu(self.conv2(x)))
        #print(x.shape)
        x = x.view(-1, self.fc1_size)  # !!!
        x = self.dropout1(F.relu(self.fc1(x)))
        x = self.fc3(x)
        return x
```

In [82]:

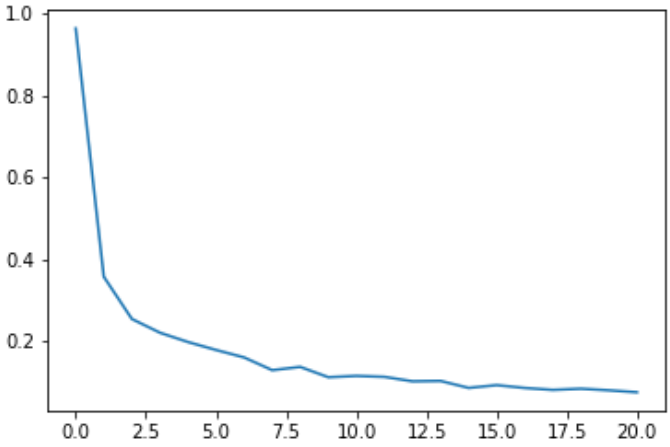
```
net = SimpleConvNet4(6, 16, 7, 3, 200, 0.1, True)
train(net)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 0.964
[1,  4000] loss: 0.357
[1,  6000] loss: 0.254
[1,  8000] loss: 0.220
[1, 10000] loss: 0.197
[1, 12000] loss: 0.178
[1, 14000] loss: 0.160

[2,  2000] loss: 0.129
[2,  4000] loss: 0.137
```

```
[2,  6000] loss: 0.111
[2,  8000] loss: 0.115
[2, 10000] loss: 0.112
[2, 12000] loss: 0.101
[2, 14000] loss: 0.102


[3,  2000] loss: 0.085
[3,  4000] loss: 0.092
[3,  6000] loss: 0.085
[3,  8000] loss: 0.080
[3, 10000] loss: 0.083
[3, 12000] loss: 0.079
[3, 14000] loss: 0.075
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20] [0.96358754 0.35685433 0
.25374047 0.22019807 0.19741353 0.17800704
 0.15979436 0.12870309 0.13664885 0.11143231 0.1146454  0.11230784
 0.10146998 0.10233663 0.08533024 0.09211819 0.08516543 0.08047513
 0.08333916 0.07945136 0.07470621]
```




```
OK
Accuracy of  0 : 99 %
Accuracy of  1 : 99 %
Accuracy of  2 : 98 %
Accuracy of  3 : 97 %
Accuracy of  4 : 98 %
Accuracy of  5 : 98 %
Accuracy of  6 : 98 %
Accuracy of  7 : 97 %
Accuracy of  8 : 97 %
Accuracy of  9 : 96 %
Средняя точность: 98.18
```

Пробую изменить дропаут

In [85]:

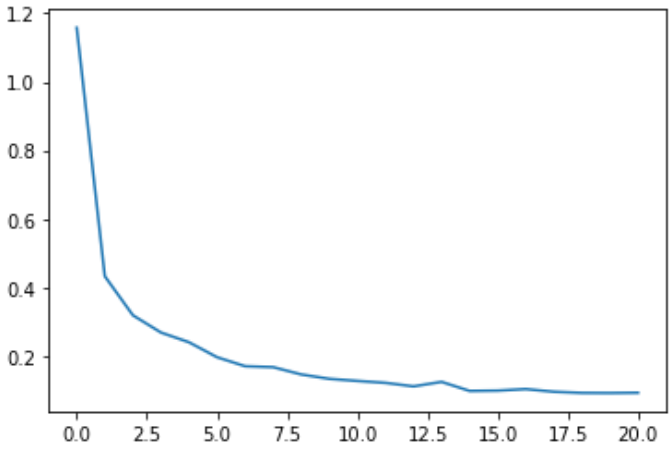
```
net = SimpleConvNet4(6, 16, 7, 3, 200, 0.3, True)
train(net)
check_network(net)
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:6: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(num_epochs)):
```

```
C:\Users\KOSHI8~1\AppData\Local\Temp/ipykernel_7356/2118872430.py:8: TqdmDeprecationWarni
ng: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for i, batch in enumerate(tqdm_notebook(trainloader)):
```

```
[1,  2000] loss: 1.158
[1,  4000] loss: 0.434
[1,  6000] loss: 0.320
[1,  8000] loss: 0.270
[1, 10000] loss: 0.241
```

```
[1, 10000] loss: 0.241
[1, 12000] loss: 0.198
[1, 14000] loss: 0.171

[2,  2000] loss: 0.169
[2,  4000] loss: 0.147
[2,  6000] loss: 0.135
[2,  8000] loss: 0.128
[2, 10000] loss: 0.123
[2, 12000] loss: 0.113
[2, 14000] loss: 0.126

[3,  2000] loss: 0.099
[3,  4000] loss: 0.100
[3,  6000] loss: 0.105
[3,  8000] loss: 0.097
[3, 10000] loss: 0.094
[3, 12000] loss: 0.093
[3, 14000] loss: 0.094
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20] [1.15786507 0.43355909 0
.31988913 0.26953541 0.24116653 0.19752988
 0.17128725 0.16858621 0.14727219 0.13455264 0.12842832 0.12283802
 0.11254986 0.12570842 0.0993134  0.10028807 0.10462305 0.0969286
 0.09356108 0.09324017 0.09405559]
```




```
OK
Accuracy of  0 : 97 %
Accuracy of  1 : 98 %
Accuracy of  2 : 98 %
Accuracy of  3 : 98 %
Accuracy of  4 : 96 %
Accuracy of  5 : 97 %
Accuracy of  6 : 97 %
Accuracy of  7 : 96 %
Accuracy of  8 : 96 %
Accuracy of  9 : 98 %
Средняя точность: 97.53
```

In [ ]:

```
Итог
Пробую изменить размер полнсвяз. слоев - не помогло
Попробуем заменить *MaxPool2d* на *AvgPool2d* - точность упала на 1%
Пробую изменить размер полнсвяз. слоев - эффекта нет
Пробую добавить сверточный слой - достиг 90%, примерно за такое же кол-во дообучений, ка
к и SimpleConvNet2
Пробую добавить дропаут, может хоть в этой лабе он поможет - 0.1 не помог; 0.3 сделал ещ
е хуже.

Лучшее значение - 90%
```