Laboratorijska vježba - Obrada informacija - Neuronske mreže

```
# !pip install torch===1.7.1+cu110 torchvision===0.8.2+cu110 torchaudio===0.7.2 -f https://download.pytorch.org/whl/torch_stable.html --no-dependencies
!pip install torch==1.7.1+cu110 torchvision==0.8.2+cu110 torchaudio==0.7.2 -f https://download.pytorch.org/whl/torch_stable.html --no-dependencies
!pip install torchsummary --no-dependencies
!pip install numpy matplotlib opencv-python --no-dependencies
!pip install tqdm wandb --no-dependencies
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.7.1+cu110
  Downloading https://download.pytorch.org/whl/cu110/torch-1.7.1%2Bcu110-cp38-cp38-linux_x86_64.whl (1156.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 GB 107.1 MB/s eta 0:00:01tcmalloc: large alloc 1156767744 bytes == 0x2e3e000 @  0x7fa3fe72d1e7 0x4d30a
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 GB 95.2 MB/s eta 0:00:01tcmalloc: large alloc 1445961728 bytes == 0x47d6c000 @  0x7fa3fe72e615 0x5d6f4
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 GB 1.2 MB/s eta 0:00:00
Collecting torchvision==0.8.2+cu110
  Downloading https://download.pytorch.org/whl/cu110/torchvision-0.8.2%2Bcu110-cp38-cp38-linux_x86_64.whl (12.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.9/12.9 MB 75.7 MB/s eta 0:00:00
Collecting torchaudio==0.7.2
  Downloading torchaudio-0.7.2-cp38-cp38-manylinux1_x86_64.whl (7.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 70.6 MB/s eta 0:00:00
Installing collected packages: torchvision, torchaudio, torch
  Attempting uninstall: torchvision
    Found existing installation: torchvision 0.14.0+cu116
    Uninstalling torchvision-0.14.0+cu116:
      Successfully uninstalled torchvision-0.14.0+cu116
  Attempting uninstall: torchaudio
    Found existing installation: torchaudio 0.13.0+cu116
    Uninstalling torchaudio-0.13.0+cu116:
      Successfully uninstalled torchaudio-0.13.0+cu116
  Attempting uninstall: torch
    Found existing installation: torch 1.13.0+cu116
    Uninstalling torch-1.13.0+cu116:
Saved successfully!      torch-1.13.0+cu116
                         7.1+cu110 torchaudio-0.7.2 torchvision-0.8.2+cu110
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: torchsummary in /usr/local/lib/python3.8/dist-packages (1.5.1)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (1.21.6)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (3.2.2)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.8/dist-packages (4.6.0.66)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (4.64.1)
Collecting wandb
  Downloading wandb-0.13.7-py2.py3-none-any.whl (1.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 48.9 MB/s eta 0:00:00
Installing collected packages: wandb
Successfully installed wandb-0.13.7
```

```
import torch
import torch.nn as nn
import torchvision
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import TensorDataset, Dataset, DataLoader

from torchsummary import summary

import cv2
import numpy as np
import matplotlib.pyplot as plt

!nvcc --version
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
%load_ext nvcc_plugin
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Sun_Feb_14_21:12:58_PST_2021
Cuda compilation tools, release 11.2, V11.2.152
Build cuda_11.2.r11.2/compiler.29618528_0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-siluanxd
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-siluanxd
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit aac710a35f52bb78ab34d2e52517237941399eff
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl size=4304 sha256=2d48aba6525fd5bcf75703f848553cd47cec7d8a27cb328e160119ee26505473
  Stored in directory: /tmp/pip-ephem-wheel-cache-w07usogn/wheels/f3/08/cc/e2b5b0e1c92df07dbb50a6f024a68ce090f5e7b2316b41756d
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
created output directory at /content/src
Out bin /content/result.out
```

```
print("Number of GPUs:", torch.cuda.device_count()) if torch.cuda.is_available() else print("CUDA is not available.")

print("PyTorch version:", torch.__version__)
```

```
Number of GPUs: 1
PyTorch version: 1.7.1+cu110
```

Zadatak 1 - Klasifikacija slike rukom pisanih znamenki

U prvom zadatku ove laboratorijske vježbe želimo analizirati utjecaj arhitekture mreže i drugih hiperparametara na uspješnost predikcije. Vaš zadatak je složiti nekoliko modela različitih karakteristika, te će te te modele istrenirati na problemu klasifikacije rukom pisanih znamenki.

Veliki dio koda koji je potreban za provođenje vježbe je dan. Vi ćete riješiti zadatak nadopunjavanjem koda. Također ste slobodni izmjeniti predloženi kod, ali ne preporuča se. Za labos je potreban Python 3.6+ i PyTorch 1.6+.

Učitavanje podataka

Sljedeći kod priprema MNIST Dataset objekte koji dolaze s PyTorch paketom. Također instanciramo i DataLoader objekte koji rukuju sa mješanjem i batchanjem skupa podataka.

```
batch_size_train = 64
batch_size_test = 64

train_set = torchvision.datasets.MNIST('./files/', train=True, download=True,
                                       transform=torchvision.transforms.Compose([
                                           torchvision.transforms.ToTensor(),
                                           torchvision.transforms.Normalize((0.1307,), (0.3081,))])
                                      )


test_set = torchvision.datasets.MNIST('./files/', train=False, download=True,
                                      transform=torchvision.transforms.Compose([
                                          torchvision.transforms.ToTensor(),
                                          torchvision.transforms.Normalize((0.1307,), (0.3081,))])
                                     )
```

Saved successfully!

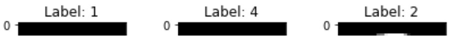
```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./files/MNIST/raw/train-images-idx3-ubyte.gz
 34%     9920512/? [01:41<00:00, 32998091.17it/s]
Extracting ./files/MNIST/raw/train-images-idx3-ubyte.gz to ./files/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./files/MNIST/raw/train-labels-idx1-ubyte.gz
    32768/? [01:39<00:00, 329.74it/s]
Extracting ./files/MNIST/raw/train-labels-idx1-ubyte.gz to ./files/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./files/MNIST/raw/t10k-images-idx3-ubyte.gz
    1654784/? [00:00<00:00, 1386400.66it/s]
Extracting ./files/MNIST/raw/t10k-images-idx3-ubyte.gz to ./files/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./files/MNIST/raw/t10k-labels-idx1-ubyte.gz
    8192/? [00:00<00:00, 28417.17it/s]
Extracting ./files/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./files/MNIST/raw
Processing...
/usr/local/lib/python3.8/dist-packages/torchvision/datasets/mnist.py:480: UserWarning: The given NumPy array is not writeable, and PyTorch does not support
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
Done!
```

```
train_loader = DataLoader(train_set, batch_size=batch_size_train, shuffle=True)
test_loader = DataLoader(test_set, batch_size=batch_size_test, shuffle=True)
```

Podzadatak a)

Prikažite nekoliko primjera iz skupa za testiranje. Sliku pokažite pomoću `matplotlib` funkcije `imshow`. Neka `title` prikazane slike bude labela uzorka.

```
examples = enumerate(test_loader) # creates an iterator for the test set
batch_idx, (example_data, example_targets) = next(examples) # retrieves the next element from the iterator and assigns its value to the variables on the left

fig = plt.figure()  # creates a new figure object

for i in range(6):
    plt.subplot(2,3,i+1)    # creates a subplot within the figure object
    plt.tight_layout()      # adjusts the spacing between subplots to minimize the amount of overlap

    # ----------------
    plt.imshow(example_data[i].squeeze(), cmap='gray', interpolation='none')    # displays the image for the current iteration of the for loop
                                                                                # 'cmap' argument displays the image in grayscale, and 'interpolation' prevents f
    plt.title("Label: {}".format(example_targets[i]))                           # sets the title of the plot
    # ----------------
```



Pomoćne metode za treniranje neuronskih mreža

Nakon podzadataka ove sekcije postoji skup testova na kojima možete provjeriti točnost vaših pomoćnih funkcija. Bez točno riješenih pomoćnih funkcija ostatak labosa ne možete riješiti.

Podzadatak b) - Funkcija za određivanje broja parametara PyTorch modela

Jedna od metoda usporedbe naših modela će biti po broju parametara koji čine taj model. Radi toga je potrebno napisati metodu `get_number_of_model_parameters(model)` koja za predani model `model` vraća ukupni broj parametara tog modela. Svaki PyTorch model sadrži implementaciju metode `.parameters()` koja vraća iterator nad parametrima modela. Ti parametri su tipa `torch.nn.parameter.Parameter`, čije dimenzije možemo dobiti pomoću `.shape` propertya. Dovrište traženu metodu.

```
def get_number_of_model_parameters(model):
    model_parameters = filter(lambda p: p.requires_grad, model.parameters())
    return sum([np.prod(p.size()) for p in model_parameters])
```

Podzadatak c) - Funkcija za treniranje modela

Model se trenira u četiri koraka.

1. Izračuna se prolaz unaprijed nad jednim batchom.
2. Na temelju dobivenog izlaza i točnih labela se računa gubitak. Kako je pokazano u demonstracijskoj bilježnici,
3. Izračunata greška se propagira unazad kroz mrežu radi računanja gradijenata.
4. Na temelju gradijenata, vrijednosti parametara i parametrima optimizatora (koji optimizator se koristi, kolika je stopa učenja, momentum i slično) se računa nova vrijednost parametara modela.

Saved successfully!

`...ain_step(train_loader, epoch, device, verbose)`.

Napomene:

- Grešku koju trebate računati je "negative log likelihood loss", za koju PyTorch nudi implementaciju. Preporučamo da koristite gotovu implementaciju loss funkcije.
- Računanje gradijenata pomoću propagacije greške u nazad se računa pomoću metode `.backward()`. Nad kojim elementom pozivamo tu metodu?
- Korak optimizacije se radi pomoću `.step()` metode optimizator objekta. Pretpostavite da postoji objekt `optimizer` u globalnom scopeu.
- Pripazite da Vam se gradijenti ne akumuliraju kroz više koraka optimizacije. PyTorch modeli nude metodu `.zero_grad()` koja postavlja vrijednosti svih gradijenata nekog modela na 0.

```
def train_step(network, train_loader, epoch, device, verbose=True):
    train_losses = []
    train_counter = []

    network.train()

    for batch_idx, (data, target) in enumerate(train_loader):
        data = data.to(device)
        target = target.to(device)

        # -------
        network.zero_grad()
        output = network(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        # ----------

        if (batch_idx % log_interval == 0):
            if verbose:
                print('Train Epoch: {:5d} [{:5d}/{:5d} ({:2.0f}%)]\tLoss: {:.6f}'.format(
                    epoch,
                    batch_idx * len(data),
                    len(train_loader.dataset),
                    100. * batch_idx / len(train_loader),
                    loss.item()))

            train_losses.append(loss.item())
            train_counter.append((batch_idx*64) + ((epoch-1)*len(train_loader.dataset)))

        optimizer.step()

    return train_losses, train_counter
```

Podzadatak d) - Funkcija za evaluaciju modela

Uspješnost učenja određujemo pomoću metrika točnosti. U ovoj laboratoriskoj vježbi pratimo dvije metrike - `negative log likelihood` i `accuracy`. Sa NLLLoss smo se već susreli; accuracy definiramo kao:

$$accuracy = \frac{\text{number of correctly classified samples}}{\text{total number of samples}}$$

Nadopunite funkciju `test(network, test_loader, device, verbose)` tako da se model evaluira za navedene metrike.

U predloženom kodu se koristi `with torch.no_grad()`. Kako tijekom evaluacije ne mjenjamo parametre modela, gradijent nam nije potreban. Time ubrzavamo računanje (ne računa se gradijent), štedimo memoriju (izračunati gradijent se ne sprema) i sprječavamo buduće probleme (npr. ostanu gradijenti do sljedeće faze treniranja, gdje se gradijenti test seta iskoriste za učenje).

```
def test(network, test_loader, device, verbose=True):
    network.eval()

    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data = data.to(device)
            target = target.to(device)

            # -----
            output = network(data)
            loss = F.nll_loss(output, target, reduction="sum").item()
            test_loss += loss

            pred = output.max(1, keepdim=True)[1].unsqueeze(1)
            # pred = output.max(1, keepdim=True)
            correct += pred.eq(target.data.view_as(pred)).sum()
            # -------

    test_loss /= len(test_loader.dataset)
    accuracy = 100. * correct / len(test_loader.dataset)

    if verbose:
        print('\nTest set: Avg. loss: {:.4f}, Accuracy: {:5d}/{:5d} ({:2.2f}%)\n'.format(
            test_loss,
            correct,
            len(test_loader.dataset),
            accuracy))

    return test_loss, accuracy.item()
```


vršavamo eksperimente, spremamo rezultate i uspoređujemo. Rezultate ćemo spremati u mapi `results`, tako da će `key` mape biti naziv eksperimenta, a vrijednost će biti `tuple` koji sadrži vrijednosti po kojima se model uspoređuje.

```
results = dict()
```

Podzadatak e) - Funkcija za provođenje cijelokupnog eksperimenta nad jednim modelom

Sada je vrijeme da se koraci iz prethodnih podzadataka objedine. Funkcija `train_network(network, train_loader, test_loader, device)` radi po sljedećem principu:

- Pretpostavlja se da u globalnom scopeu postoji varijabla imena `n_epochs` koja nam govori koliko epoha će se eksperiment izvršavati
- Liste `train_losses` i `test_losses` skupljaju loss vrijednosti tijekom treniranja, dok `train_counter` i `test_counter` skupljaju trenutke u kojima se metrika zabilježila (drugim riječima, to su X i Y os na grafu "loss po vremenu")
- prije samog treniranja se vrši testiranje modela, da se utvrdi performansa slučajnog modela
- U svakoj epohi se model trenira, testira i rezultati se zapisuju u odgovarajuće liste
- Funkcija vraća te liste na kraju

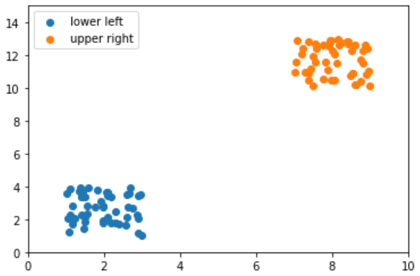
```
def train_network(network, train_loader, test_loader, device='cpu'):
    train_losses = []
    train_counter = []
    test_losses = []
    test_counter = [i*len(train_loader.dataset) for i in range(n_epochs + 1)]

    # -----
    network.train()
    test_loss, test_accuracy = test(network, test_loader, device)
    # ------------
    test_losses.append(test_loss)

    for epoch in range(1, n_epochs + 1):
        # ----
        new_train_losses, new_train_counter = train_step(network, train_loader, epoch, device)
        test_loss, test_accuracy = test(network, test_loader, device)
        # ---------

        train_losses.extend(new_train_losses)
        train_counter.extend(new_train_counter)
        test_losses.append(test_loss)

    return train_losses, train_counter, test_losses, test_counter, test_accuracy
```

Testovi za utvrđivanje točnosti rada pomoćnih funkcija

Sljedeći kod služi kao pomoć za provjeru ispravnosti gore traženih pomoćnih funkcija. Generira se dataset u dva odvojena skupa, i cilj je naučiti model koji klasificira iz kojeg skupa točka dolazi. Prvo generiramo podatke i slažemo DataLoader:

```
data_x = np.hstack([np.random.uniform(1, 3, 50), np.random.uniform(7, 9, 50)])
data_y = np.hstack([np.random.uniform(1, 4, 50), np.random.uniform(10, 13, 50)])
labels = [0 if x < 50 else 1 for x in range(0, 100)]

for idx, unique_label in enumerate(["lower left", "upper right"]):
    if idx == 0:
        plt.scatter(data_x[0:50], data_y[0:50], label=unique_label)
    if idx == 1:
```

```
        plt.scatter(data_x[50:], data_y[50:], label=unique_label)

plt.legend()
plt.xlim(0, 10)
plt.ylim(0, 15)


tensor_x = torch.Tensor(np.dstack([data_x, data_y]).reshape(100, 2).astype(np.float32))
tensor_y = torch.Tensor(labels).to(dtype=torch.int64)

toy_dataset = TensorDataset(tensor_x, tensor_y)
toy_dataloader = DataLoader(toy_dataset, batch_size=1, shuffle=True)
```



Nakon toga definiramo naš model. U ovom slučaju je model dvoslojna mreža sa dva potpuno povezana sloja.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 4)
```



```
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)

        return F.log_softmax(x, dim=1)
```

Podešavamo parametre koje naše pomoćne funkcije očekivaju, te instanciramo model i optimizator.

```
n_epochs = 30
learning_rate = 0.01
log_interval = 33

network = Net().to('cuda')
optimizer = optim.SGD(network.parameters(), lr=learning_rate)
```

I sada možemo trenirati naš model. Vaše funkcije su ispravne ako točnost doesgne 100% (ili barem jako blizu). **Bez ispravnih pomoćnih funkcija nećete moći riješiti ostatak labosa.**

```
test_accuracy = train_network(network, toy_dataloader, toy_dataloader, 'cuda')[-1]
test_accuracy
```

```
Train Epoch:    28 [   66/  100 (66%)]  Loss: 0.000954
Train Epoch:    28 [   99/  100 (99%)]  Loss: 0.420354

Test set: Avg. loss: 0.2257, Accuracy:   100/  100 (100.00%)

Train Epoch:    29 [    0/  100 ( 0%)]  Loss: 0.418004
Train Epoch:    29 [   33/  100 (33%)]  Loss: 0.424397
Train Epoch:    29 [   66/  100 (66%)]  Loss: 0.434570
Train Epoch:    29 [   99/  100 (99%)]  Loss: 1.076615

Test set: Avg. loss: 0.2132, Accuracy:   100/  100 (100.00%)

Train Epoch:    30 [    0/  100 ( 0%)]  Loss: 0.001574
Train Epoch:    30 [   33/  100 (33%)]  Loss: 0.405888
Train Epoch:    30 [   66/  100 (66%)]  Loss: 0.421313
Train Epoch:    30 [   99/  100 (99%)]  Loss: 1.083964

Test set: Avg. loss: 0.2225, Accuracy:   100/  100 (100.00%)

100.0
```

Provođenje eksperimenata i analiza rezultata

Podzadatak f) - Eksperimenti

Sljedeća faza labosa je korištenje naših funkcija u okviru eksperimenata. Potrebno je testirati sljedeće modele:

- **Plitki model sa uskim slojevima**
  - Model je plitak po tome što nema puno slojeva (ne ide u dubinu) i uzak po tome što sami slojevi nemaju veliki broj elemenata (npr. 1 sloj sa 100 neurona umjesto 10 slojeva sa 10 neurona)
  - Arhitektura modela je sljedeća:
    - Konvolucijski sloj 5x5x10
    - ciju)
    - Max pooling
    - ReLU aktivacija
    - Potpuno povezani sloj sa 20 neurona, ReLU aktivacija
    - Potpuno povezani sloj za klasifikaciju u 10 klasa, log softmax aktivacijska funkcija
  - U `results` mapi se sprema pod ključem `shallow_and_narrow_{stopa učenja}`

Saved successfully!

- **Plitki model sa širokim slojevima**
  - Ovaj model također nema puno slojeva, ali ti slojevi imaju puno elemenata
  - Arhitektura modela je sljedeća:
    - Konvolucijski sloj 5x5x40
    - Dropout (za regularizaciju)
    - Max pooling
    - ReLU aktivacija
    - Potpuno povezani sloj sa 64 neurona, ReLU aktivacija
    - Potpuno povezani sloj za klasifikaciju u 10 klasa, log softmax aktivacijska funkcija
  - U `results` mapi se sprema pod ključem `shallow_and_wide_{stopa učenja}`
- **Duboki model sa uskim slojevima**
  - Ovaj model ima puno slojeva, ali su ti slojevi ograničeni u svojoj širini
  - Arhitektura modela je sljedeća:
    - Konvolucijski sloj 5x5x10, ReLU aktivacijska funkcija
    - Max pooling
    - Konvolucijski sloj 5x5x20, ReLU aktivacijska funkcija
    - Max pooling
    - Potpuno povezani sloj sa 64 neurona, ReLU aktivacija
    - Dropout (za regularizaciju)
    - Potpuno povezani sloj za klasifikaciju u 10 klasa, log softmax aktivacijska funkcija
  - U `results` mapi se sprema pod ključem `deep_and_narrow_{stopa učenja}`
- **Duboki model sa širokim slojevima**
  - Model koji ima sve komponente dobro (ili previše?) zastupljene.
  - Arhitektura modela je sljedeća:
    - Konvolucijski sloj 5x5x32, ReLU aktivacijska funkcija
    - Max pooling
    - Konvolucijski sloj 5x5x64, ReLU aktivacijska funkcija
    - Max pooling
    - Potpuno povezani sloj sa 50 neurona, ReLU aktivacija
    - Dropout (za regularizaciju)
    - Potpuno povezani sloj za klasifikaciju u 10 klasa, log softmax aktivacijska funkcija
  - U `results` mapi se sprema pod ključem `deep_and_wide_{stopa učenja}`

Implementirajte `.__init__(self)` i `.forward(self, x)` metode za svaki od opisanih modela, trenirajte ih, evaluirajte i spremite metrike.
Ponovite taj postupak za 3 različite stope učenja: 0.0000001, 0.01 i 1.

Sve potrebne slojeve za ostvarenje navedenih modela možete pronaći u torch.nn modulu. Detalje možete pronaći u službenoj PyTorch dokumentaciji: https://pytorch.org/docs/stable/index.html

Spremite si najbolji model. Biti će potreban u podzadatku h).

Prvo je potrebno podesiti parametre. Parametri su sljedeći:

- n_epochs - broj epoha eksperimenta
- learning_rate - stopa učenja
- log_interval - broj koraka između dva ispisa tijekom treniranja (ispis se dešava samo ako se funkcija poziva s argumentom verbose=True)
- device - oznaka na kojem se uređaju izvršava eksperiment; "cuda" za GPU, "cpu" za CPU

```
n_epochs = 3
learning_rate = 0.01
log_interval = 100
device = 'cuda'
```

Naš model definiramo u klasi "Net" koja nasljeđuje nn.Module. Nadjačajte metode `__init__(self)` i `forward(self, x)` kako je opisano u tekstu zadatka.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)  # 32 -> number of output channels for the 'conv1' layer

        # -------
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5) # slicno ...

        self.fc1 = nn.Linear(1024, 50)  # 1024 and 50 are number of input and output units for fc1 layer

        self.dropout = nn.Dropout(p=0.5)  # p ->  probability of dropping out an input unit

        self.fc2 = nn.Linear(50, 10)  # opet, 50 je input size, 10 je output size (jer 10 znamenki)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))

        # ----------
        x = F.relu(F.max_pool2d(self.conv2(x), 2))  # prati upute
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        # ----------

        return F.log_softmax(x, dim=1)  # zadano je da se koristi softmax za aktivacijsku fju output layera
```


Da bi trenirali naš model, potrebno je napraviti instancu mreže i optimizatora. Koristite Stohastic Gradient Descent optimizator iz torch.optim modula. Detalji se mogu pronaći u službenoj dokuemntaciji PyTorcha za optim modul: https://pytorch.org/docs/stable/optim.html

```
network = Net().to('cuda')     # inicijalizacija mreze
# ----
optimizer = optim.SGD(network.parameters(), lr=learning_rate) # inicijalizacija sgd optimatora
# ---------
```

Iskoristimo našu pripremljenu funkciju za izvođenje eksperimenta:

```
train_losses, train_counter, test_losses, test_counter, test_accuracy = train_network(network, train_loader, test_loader, device)
```

```
Test set: Avg. loss: 2.3148, Accuracy:   895/10000 (8.95%)

Train Epoch:     1 [     0/60000 ( 0%)]  Loss: 2.305971
Train Epoch:     1 [  6400/60000 (11%)]  Loss: 1.727602
Train Epoch:     1 [12800/60000 (21%)]  Loss: 0.921389
Train Epoch:     1 [19200/60000 (32%)]  Loss: 0.623399
Train Epoch:     1 [25600/60000 (43%)]  Loss: 0.347317
Train Epoch:     1 [32000/60000 (53%)]  Loss: 0.440131
Train Epoch:     1 [38400/60000 (64%)]  Loss: 0.634782
Train Epoch:     1 [44800/60000 (75%)]  Loss: 0.405163
Train Epoch:     1 [51200/60000 (85%)]  Loss: 0.441703
Train Epoch:     1 [57600/60000 (96%)]  Loss: 0.595428

Test set: Avg. loss: 0.1630, Accuracy:  9524/10000 (95.24%)

Train Epoch:     2 [     0/60000 ( 0%)]  Loss: 0.463810
Train Epoch:     2 [  6400/60000 (11%)]  Loss: 0.209255
Train Epoch:     2 [12800/60000 (21%)]  Loss: 0.284397
Train Epoch:     2 [19200/60000 (32%)]  Loss: 0.274458
Train Epoch:     2 [25600/60000 (43%)]  Loss: 0.227036
Train Epoch:     2 [32000/60000 (53%)]  Loss: 0.301212
Train Epoch:     2 [38400/60000 (64%)]  Loss: 0.144657
Train Epoch:     2 [44800/60000 (75%)]  Loss: 0.228603
Train Epoch:     2 [51200/60000 (85%)]  Loss: 0.297365
Train Epoch:     2 [57600/60000 (96%)]  Loss: 0.215866

Test set: Avg. loss: 0.1004, Accuracy:  9701/10000 (97.01%)

Train Epoch:     3 [     0/60000 ( 0%)]  Loss: 0.180298
Train Epoch:     3 [  6400/60000 (11%)]  Loss: 0.283273
Train Epoch:     3 [12800/60000 (21%)]  Loss: 0.232441
Train Epoch:     3 [19200/60000 (32%)]  Loss: 0.203630
Train Epoch:     3 [25600/60000 (43%)]  Loss: 0.182355
```

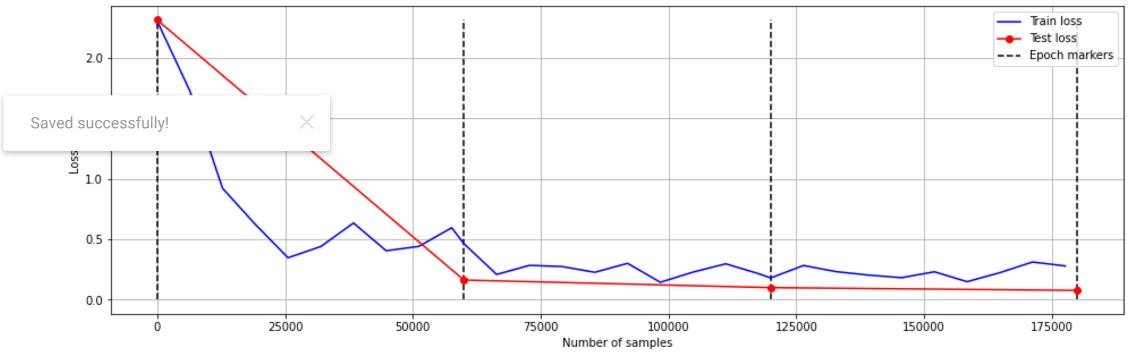
```
Train Epoch:     3 [32000/60000 (53%)]  Loss: 0.231308
Train Epoch:     3 [38400/60000 (64%)]  Loss: 0.149345
Train Epoch:     3 [44800/60000 (75%)]  Loss: 0.223853
Train Epoch:     3 [51200/60000 (85%)]  Loss: 0.312233
Train Epoch:     3 [57600/60000 (96%)]  Loss: 0.279645

Test set: Avg. loss: 0.0773, Accuracy:  9759/10000 (97.59%)
```

Spremimo rezultate u mapu `results` kako je navedeno u zadatku. Također nam je potreban broj parametara mreže, što možemo izračunati u ovom koraku.

```
number_of_parameters = get_number_of_model_parameters(network)
results[f'deep_and_wide_{learning_rate}'] = (train_counter, train_losses, test_counter, test_losses, test_accuracy, number_of_parameters)
```

Prikažimo rezultate za ovaj eksperiment:

```
fig = plt.figure(figsize=(16, 5))
plt.plot(train_counter, train_losses, color='blue', label='Train loss')
plt.plot(test_counter, test_losses, color='red',  marker='o', label='Test loss')
plt.vlines(test_counter, 0, max(train_losses + test_losses), linestyles='dashed', label='Epoch markers')

plt.legend(loc='upper right')
plt.xlabel('Number of samples')
plt.ylabel('Loss')
plt.grid()
```




```
class PlitakUzak(nn.Module):
    def __init__(self):
        super(PlitakUzak, self).__init__()

        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.dropout = nn.Dropout(p=0.5)  # p ->  probability of dropping out an input unit

        self.fc1 = nn.Linear(1440, 20)  # 1024 and 50 are number of input and output units for fc1 layer
        self.fc2 = nn.Linear(20, 10)  # opet, 50 je input size, 10 je output size (jer 10 znamenki)
        #-------------

    def forward(self, x):
        x = self.conv1(x)
        x = self.dropout(x)

        x = F.relu(F.max_pool2d(x, 2))  # ne znam zasto ne ide self.conv2(x), ali tako baca error :(

        x = x.view(x.size(0), -1)   # reshaping the tensor

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))

        return F.log_softmax(x, dim=1)

for learning_rate in [1, 0.01, 0.0000001]:

  # kao i u primjeru:
  network = PlitakUzak().to('cuda')    # inicijalizacija mreze
  optimizer = optim.SGD(network.parameters(), lr=learning_rate) # inicijalizacija sgd optimatora
  train_losses, train_counter, test_losses, test_counter, test_accuracy = train_network(network, train_loader, test_loader, device)

  params = get_number_of_model_parameters(network)  # broj parametara
  results[f'plitak_i_uzak {learning_rate}'] = (train_counter, train_losses, test_counter, test_losses, test_accuracy, params)
  torch.save(network, f'plitak_i_uzak_model_{learning_rate}.pt')
```

```
Test set: Avg. loss: 2.3052, Accuracy:  1007/10000 (10.07%)

Train Epoch:     1 [    0/60000 ( 0%)]  Loss: 2.282038
Train Epoch:     1 [ 6400/60000 (11%)]  Loss: 2.302585
Train Epoch:     1 [12800/60000 (21%)]  Loss: 2.302585
Train Epoch:     1 [19200/60000 (32%)]  Loss: 2.302585
Train Epoch:     1 [25600/60000 (43%)]  Loss: 2.302585
Train Epoch:     1 [32000/60000 (53%)]  Loss: 2.302585
Train Epoch:     1 [38400/60000 (64%)]  Loss: 2.302585
Train Epoch:     1 [44800/60000 (75%)]  Loss: 2.302585
Train Epoch:     1 [51200/60000 (85%)]  Loss: 2.302585
Train Epoch:     1 [57600/60000 (96%)]  Loss: 2.302585
```

```
Test set: Avg. loss: 2.3026, Accuracy:   980/10000 (9.80%)

Train Epoch:     2 [    0/60000 ( 0%)]  Loss: 2.302585
Train Epoch:     2 [ 6400/60000 (11%)]  Loss: 2.302585
Train Epoch:     2 [12800/60000 (21%)]  Loss: 2.302585
Train Epoch:     2 [19200/60000 (32%)]  Loss: 2.302585
Train Epoch:     2 [25600/60000 (43%)]  Loss: 2.302585
Train Epoch:     2 [32000/60000 (53%)]  Loss: 2.302585
Train Epoch:     2 [38400/60000 (64%)]  Loss: 2.302585
Train Epoch:     2 [44800/60000 (75%)]  Loss: 2.302585
Train Epoch:     2 [51200/60000 (85%)]  Loss: 2.302585
Train Epoch:     2 [57600/60000 (96%)]  Loss: 2.302585

Test set: Avg. loss: 2.3026, Accuracy:   980/10000 (9.80%)

Train Epoch:     3 [    0/60000 ( 0%)]  Loss: 2.302585
Train Epoch:     3 [ 6400/60000 (11%)]  Loss: 2.302585
Train Epoch:     3 [12800/60000 (21%)]  Loss: 2.302585
Train Epoch:     3 [19200/60000 (32%)]  Loss: 2.302585
Train Epoch:     3 [25600/60000 (43%)]  Loss: 2.302585
Train Epoch:     3 [32000/60000 (53%)]  Loss: 2.302585
Train Epoch:     3 [38400/60000 (64%)]  Loss: 2.302585
Train Epoch:     3 [44800/60000 (75%)]  Loss: 2.302585
Train Epoch:     3 [51200/60000 (85%)]  Loss: 2.302585
Train Epoch:     3 [57600/60000 (96%)]  Loss: 2.302585

Test set: Avg. loss: 2.3026, Accuracy:   980/10000 (9.80%)


Test set: Avg. loss: 2.3103, Accuracy:  1008/10000 (10.08%)

Train Epoch:     1 [    0/60000 ( 0%)]  Loss: 2.314342
Train Epoch:     1 [ 6400/60000 (11%)]  Loss: 1.177599
Train Epoch:     1 [12800/60000 (21%)]  Loss: 1.134222
Train Epoch:     1 [19200/60000 (32%)]  Loss: 1.170314
Train Epoch:     1 [25600/60000 (43%)]  Loss: 0.727645
Train Epoch:     1 [32000/60000 (53%)]  Loss: 0.962406
Saved successfully!  (64%)]  Loss: 1.044161
                     (75%)]  Loss: 0.791670
Train Epoch:     1 [51200/60000 (85%)]  Loss: 0.930717
Train Epoch:     1 [57600/60000 (96%)]  Loss: 0.979139

Test set: Avg. loss: 1.0125, Accuracy:  6489/10000 (64.89%)
```

```
class PlitakSirok(nn.Module):
    def __init__(self):
        super(PlitakSirok, self).__init__()

        self.conv1 = nn.Conv2d(1, 40, kernel_size=5)
        self.dropout = nn.Dropout(p=0.5)  # p ->  probability of dropping out an input unit

        self.fc1 = nn.Linear(1440 * 4, 64)  # 1024 and 50 are number of input and output units for fc1 layer
        self.fc2 = nn.Linear(64, 10)  # opet, 50 je input size, 10 je output size (jer 10 znamenki)
        #-------------

    def forward(self, x):
        x = self.conv1(x)
        x = self.dropout(x)

        x = F.relu(F.max_pool2d(x, 2))  # ne znam zasto ne ide self.conv2(x), ali tako baca error :(

        x = x.view(x.size(0), -1)   # reshaping the tensor

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))

        return F.log_softmax(x, dim=1)

for learning_rate in [1, 0.01, 0.0000001]:

  # kao i u primjeru:
  network = PlitakSirok().to('cuda')    # inicijalizacija mreze
  optimizer = optim.SGD(network.parameters(), lr=learning_rate) # inicijalizacija sgd optimatora
  train_losses, train_counter, test_losses, test_counter, test_accuracy = train_network(network, train_loader, test_loader, 'cuda')

  params = get_number_of_model_parameters(network)  # broj parametara
  results[f'plitak_i_sirok {learning_rate}'] = (train_counter, train_losses, test_counter, test_losses, test_accuracy, params)
  torch.save(network, f'plitak_i_sirok_model_{learning_rate}.pt')
```

```
Train Epoch:     1 [    0/60000 ( 0%)]  Loss: 2.318891
Train Epoch:     1 [ 6400/60000 (11%)]  Loss: 2.270370
Train Epoch:     1 [12800/60000 (21%)]  Loss: 2.311050
Train Epoch:     1 [19200/60000 (32%)]  Loss: 2.277033
Train Epoch:     1 [25600/60000 (43%)]  Loss: 2.295301
Train Epoch:     1 [32000/60000 (53%)]  Loss: 2.329367
Train Epoch:     1 [38400/60000 (64%)]  Loss: 2.332945
Train Epoch:     1 [44800/60000 (75%)]  Loss: 2.270557
Train Epoch:     1 [51200/60000 (85%)]  Loss: 2.272485
Train Epoch:     1 [57600/60000 (96%)]  Loss: 2.308181

Test set: Avg. loss: 2.2958, Accuracy:  1083/10000 (10.83%)

Train Epoch:     2 [    0/60000 ( 0%)]  Loss: 2.298876
Train Epoch:     2 [ 6400/60000 (11%)]  Loss: 2.299907
Train Epoch:     2 [12800/60000 (21%)]  Loss: 2.284943
Train Epoch:     2 [19200/60000 (32%)]  Loss: 2.285243
Train Epoch:     2 [25600/60000 (43%)]  Loss: 2.277253
Train Epoch:     2 [32000/60000 (53%)]  Loss: 2.290719
Train Epoch:     2 [38400/60000 (64%)]  Loss: 2.308682
Train Epoch:     2 [44800/60000 (75%)]  Loss: 2.301548
Train Epoch:     2 [51200/60000 (85%)]  Loss: 2.295236
Train Epoch:     2 [57600/60000 (96%)]  Loss: 2.313673

Test set: Avg. loss: 2.2955, Accuracy:  1093/10000 (10.93%)

Train Epoch:     3 [    0/60000 ( 0%)]  Loss: 2.306702
Train Epoch:     3 [ 6400/60000 (11%)]  Loss: 2.297184
Train Epoch:     3 [12800/60000 (21%)]  Loss: 2.304343
Train Epoch:     3 [19200/60000 (32%)]  Loss: 2.292874
Train Epoch:     3 [25600/60000 (43%)]  Loss: 2.277747
Train Epoch:     3 [32000/60000 (53%)]  Loss: 2.298656
Train Epoch:     3 [38400/60000 (64%)]  Loss: 2.276345
Train Epoch:     3 [44800/60000 (75%)]  Loss: 2.305908
Train Epoch:     3 [51200/60000 (85%)]  Loss: 2.260150
Train Epoch:     3 [57600/60000 (96%)]  Loss: 2.318052

Test set: Avg. loss: 2.2952, Accuracy:  1102/10000 (11.02%)
```

Saved successfully!

```
class DubokUzak(nn.Module):
    def __init__(self):

        super(DubokUzak, self).__init__()

        self.conv1 = nn.Conv2d(1, 10, kernel_size = 5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size = 5)

        self.fc1 = nn.Linear(320, 64)

        self.dropout = nn.Dropout(p=0.5)  # p ->  probability of dropping out an input unit

        self.fc2 = nn.Linear(64, 10)
        #-------------

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))  # ne znam zasto ne ide self.conv2(x), ali tako baca error :(
        x = F.relu(F.max_pool2d(self.conv2(x), 2))

        x = x.view(x.size(0), -1)       # reshaping the tensor

        x = F.relu(self.fc1(x))

        x = self.dropout(x)
        x = self.fc2(x)

        return F.log_softmax(x, dim=1)

for learning_rate in [1, 0.01, 0.0000001]:

  # kao i u primjeru:
  network = DubokUzak().to('cuda')    # inicijalizacija mreze
  optimizer = optim.SGD(network.parameters(), lr=learning_rate) # inicijalizacija sgd optimatora
  train_losses, train_counter, test_losses, test_counter, test_accuracy = train_network(network, train_loader, test_loader, 'cuda')

  params = get_number_of_model_parameters(network)  # broj parametara
  results[f'dubok_i_uzak {learning_rate}'] = (train_counter, train_losses, test_counter, test_losses, test_accuracy, params)
  torch.save(network, f'dubok_i_uzak_model_{learning_rate}.pt')
```

```
    Train Epoch:     1 [12800/60000 (21%)]  Loss: 2.298060
    Train Epoch:     1 [19200/60000 (32%)]  Loss: 2.305351
    Train Epoch:     1 [25600/60000 (43%)]  Loss: 2.322127
    Train Epoch:     1 [32000/60000 (53%)]  Loss: 2.322348
    Train Epoch:     1 [38400/60000 (64%)]  Loss: 2.313998
    Train Epoch:     1 [44800/60000 (75%)]  Loss: 2.334235
    Train Epoch:     1 [51200/60000 (85%)]  Loss: 2.306312
    Train Epoch:     1 [57600/60000 (96%)]  Loss: 2.315800

    Test set: Avg. loss: 2.3127, Accuracy:   842/10000 (8.42%)

    Train Epoch:     2 [    0/60000 ( 0%)]  Loss: 2.292514
    Train Epoch:     2 [ 6400/60000 (11%)]  Loss: 2.303262
    Train Epoch:     2 [12800/60000 (21%)]  Loss: 2.322382
    Train Epoch:     2 [19200/60000 (32%)]  Loss: 2.328591
    Train Epoch:     2 [25600/60000 (43%)]  Loss: 2.301528
    Train Epoch:     2 [32000/60000 (53%)]  Loss: 2.325546
    Train Epoch:     2 [38400/60000 (64%)]  Loss: 2.314133
    Train Epoch:     2 [44800/60000 (75%)]  Loss: 2.305403
    Train Epoch:     2 [51200/60000 (85%)]  Loss: 2.284111
    Train Epoch:     2 [57600/60000 (96%)]  Loss: 2.327922

    Test set: Avg. loss: 2.3127, Accuracy:   842/10000 (8.42%)

    Train Epoch:     3 [    0/60000 ( 0%)]  Loss: 2.302614
    Train Epoch:     3 [ 6400/60000 (11%)]  Loss: 2.307801
    Train Epoch:     3 [12800/60000 (21%)]  Loss: 2.284662
    Train Epoch:     3 [19200/60000 (32%)]  Loss: 2.325511
    Train Epoch:     3 [25600/60000 (43%)]  Loss: 2.319789
    Train Epoch:     3 [32000/60000 (53%)]  Loss: 2.316765
    Train Epoch:     3 [38400/60000 (64%)]  Loss: 2.293073
    Train Epoch:     3 [44800/60000 (75%)]  Loss: 2.323866
    Train Epoch:     3 [51200/60000 (85%)]  Loss: 2.335666
    Train Epoch:     3 [57600/60000 (96%)]  Loss: 2.298187

    Test set: Avg. loss: 2.3127, Accuracy:   842/10000 (8.42%)
```

Saved successfully!

```
    def __init__(self):

        super(DubokSirok, self).__init__()

        self.conv1 = nn.Conv2d(1, 32, kernel_size = 5)
        self.conv2 = nn.Conv2d(32, 64, kernel_size = 5)

        self.fc1 = nn.Linear(1024, 50)

        self.dropout = nn.Dropout(p=0.5)  # p ->  probability of dropping out an input unit

        self.fc2 = nn.Linear(50, 10)
        #-------------

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))  # ne znam zasto ne ide self.conv2(x), ali tako baca error :(
        x = F.relu(F.max_pool2d(self.conv2(x), 2))

        x = x.view(x.size(0), -1)       # reshaping the tensor

        x = F.relu(self.fc1(x))

        x = self.dropout(x)
        x = self.fc2(x)

        return F.log_softmax(x, dim=1)

for learning_rate in [1, 0.01, 0.0000001]:

  # kao i u primjeru:
  network = DubokSirok().to('cuda')    # inicijalizacija mreze
  optimizer = optim.SGD(network.parameters(), lr=learning_rate) # inicijalizacija sgd optimatora
  train_losses, train_counter, test_losses, test_counter, test_accuracy = train_network(network, train_loader, test_loader, 'cuda')

  params = get_number_of_model_parameters(network)  # broj parametara
  results[f'dubok_i_sirok {learning_rate}'] = (train_counter, train_losses, test_counter, test_losses, test_accuracy, params)
  torch.save(network, f'dubok_i_sirok_model_{learning_rate}.pt')
```

```
Train Epoch:     1 [25600/60000 (43%)]  Loss: 2.310548
Train Epoch:     1 [32000/60000 (53%)]  Loss: 2.332008
Train Epoch:     1 [38400/60000 (64%)]  Loss: 2.307702
Train Epoch:     1 [44800/60000 (75%)]  Loss: 2.303057
Train Epoch:     1 [51200/60000 (85%)]  Loss: 2.306766
Train Epoch:     1 [57600/60000 (96%)]  Loss: 2.290762

Test set: Avg. loss: 2.3067, Accuracy:  1237/10000 (12.37%)

Train Epoch:     2 [    0/60000 ( 0%)]  Loss: 2.326222
Train Epoch:     2 [ 6400/60000 (11%)]  Loss: 2.287430
Train Epoch:     2 [12800/60000 (21%)]  Loss: 2.319835
Train Epoch:     2 [19200/60000 (32%)]  Loss: 2.355309
Train Epoch:     2 [25600/60000 (43%)]  Loss: 2.302278
Train Epoch:     2 [32000/60000 (53%)]  Loss: 2.324193
Train Epoch:     2 [38400/60000 (64%)]  Loss: 2.337049
Train Epoch:     2 [44800/60000 (75%)]  Loss: 2.307425
Train Epoch:     2 [51200/60000 (85%)]  Loss: 2.278863
Train Epoch:     2 [57600/60000 (96%)]  Loss: 2.282593

Test set: Avg. loss: 2.3067, Accuracy:  1237/10000 (12.37%)

Train Epoch:     3 [    0/60000 ( 0%)]  Loss: 2.306205
Train Epoch:     3 [ 6400/60000 (11%)]  Loss: 2.326730
Train Epoch:     3 [12800/60000 (21%)]  Loss: 2.301656
Train Epoch:     3 [19200/60000 (32%)]  Loss: 2.323455
Train Epoch:     3 [25600/60000 (43%)]  Loss: 2.308084
Train Epoch:     3 [32000/60000 (53%)]  Loss: 2.301461
Train Epoch:     3 [38400/60000 (64%)]  Loss: 2.266500
Train Epoch:     3 [44800/60000 (75%)]  Loss: 2.333015
Train Epoch:     3 [51200/60000 (85%)]  Loss: 2.295228
Train Epoch:     3 [57600/60000 (96%)]  Loss: 2.296752

Test set: Avg. loss: 2.3066, Accuracy:  1237/10000 (12.37%)
```

Nadopunite bilježnicu sa svim traženim arhitekturama i learning rateovima zadanim u ovom podzadatku.

Saved successfully!

```
with open('results', 'wb') as f:
  pickle.dump(results, f, protocol=pickle.HIGHEST_PROTOCOL)

with open('results', 'rb') as f:
  results = pickle.load(f)
results
```

```
103856)}
```

Podzadatak g) - Usporedba rezultata

Nakon što smo izvršili sve eksperimente potrebno ih je usporediti. Nacrtajte tražene grafove, te pomoću njih odgovorite na pitanja postavljena na Moodleu.

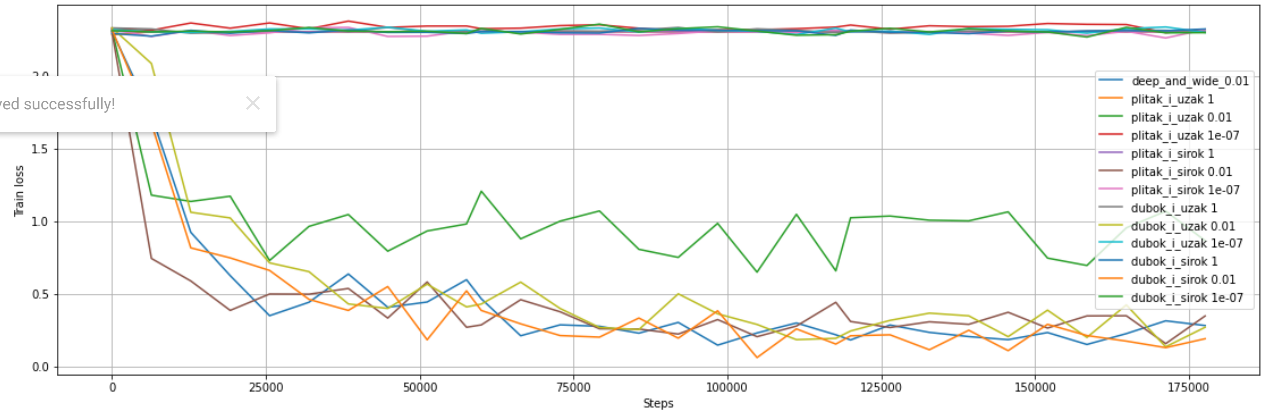
Nacrtajte graf gdje je X os vrijeme (odgovara na pitanje: koji korak treniranja?), a Y os je loss za **trening** skup podataka.

Odgovorite na sljedeća pitanja:

1. Radi li se o konzistentnom padu iz koraka u korak?
2. Jesu li neke arhitekture u startu značajno bolje od drugih?
3. Koji model je najnestabilniji tijekom treniranja?

```
plt.figure(figsize=(19, 6))

for model_key in results:
    train_counter, train_losses, test_counter, test_losses, test_accuracy, number_of_parameters = results[model_key]
    plt.plot(train_counter, train_losses, label=model_key)

plt.legend()
plt.grid()
plt.ylabel("Train loss")
plt.xlabel("Steps")
```

```
Text(0.5, 0, 'Steps')
```



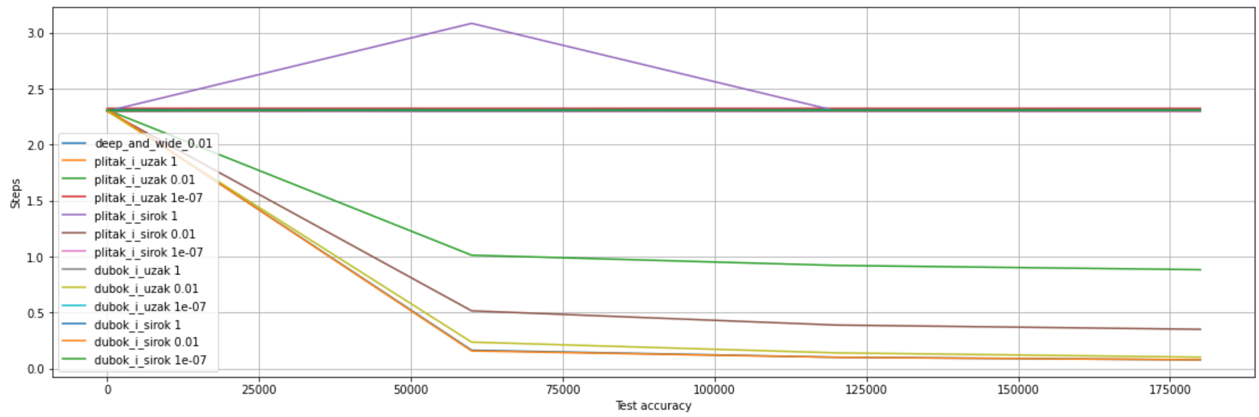
Nacrtajte graf gdje je X os vrijeme (odgovara na pitanje: koji korak treniranja?), a Y os je loss za **test** skupu podataka.

Odgovorite na sljedeća pitanja:

4. Radi li se o konzistentnom padu iz koraka u korak?
5. Jesu li neke arhitekture u startu značajno bolje od drugih?

```
plt.figure(figsize=(19, 6))

for model_key in results:
    train_counter, train_losses, test_counter, test_losses, test_accuracy, number_of_parameters = results[model_key]
    plt.plot(test_counter, test_losses, label=model_key)

plt.legend()
plt.grid()
plt.xlabel("Test accuracy")
plt.ylabel("Steps")
```

```
Text(0, 0.5, 'Steps')
```



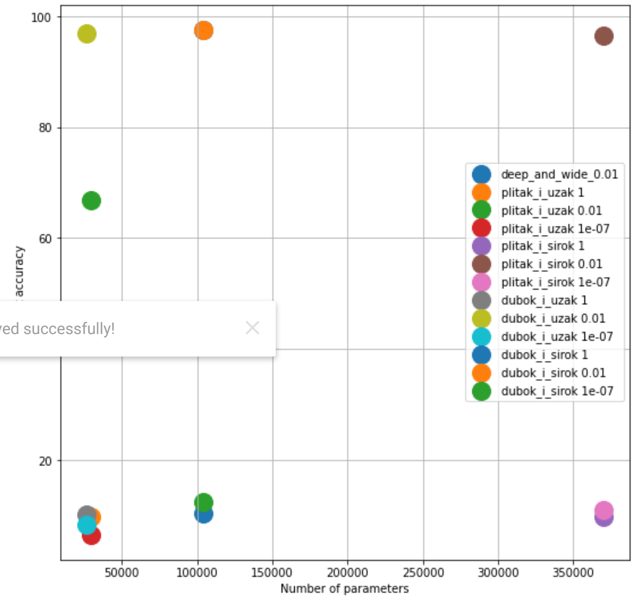
Nacrtajte graf (scatter plot) gdje je X os broj parametara modela, a Y os je točnost koju model ostvaruje na test skupu.

Odgovorite na sljedeća pitanja:

6. Koji je najbolji model?
7. Kakvi su duboki modeli u usporedbu s plitkim modelima?
8. Kakvi su široki modeli u usporedbi s uskima?

```
plt.figure(figsize=(9, 9))

for model_key in results:
    train_counter, train_losses, test_counter, test_losses, test_accuracy, number_of_parameters = results[model_key]
    plt.scatter(number_of_parameters, test_accuracy, label=model_key, s=256)

plt.legend()
plt.grid()
plt.ylabel("Test accuracy")
plt.xlabel("Number of parameters")
```

```
Text(0.5, 0, 'Number of parameters')
```



Saved successfully!

```
for model_key in results:
  train_counter, train_losses, test_counter, test_losses, test_accuracy, number_of_parameters = results[model_key]

  formatted_model_key = f"{model_key:20}"
  print(f"{formatted_model_key}: {test_accuracy}%")
```

```
deep_and_wide_0.01  : 97.58999633789062%
plitak_i_uzak 1     : 9.800000190734863%
plitak_i_uzak 0.01  : 66.8699951171875%
plitak_i_uzak 1e-07 : 6.569999694824219%
plitak_i_sirok 1    : 9.800000190734863%
plitak_i_sirok 0.01 : 96.54000091552734%
plitak_i_sirok 1e-07: 11.019999504089355%
dubok_i_uzak 1      : 10.09999942779541%
dubok_i_uzak 0.01   : 96.87999725341797%
dubok_i_uzak 1e-07  : 8.420000076293945%
dubok_i_sirok 1     : 10.279999732971191%
dubok_i_sirok 0.01  : 97.47999572753906%
dubok_i_sirok 1e-07 : 12.369999885559082%
```

Podzadatak h) - Evaluacija na neviđenom skupu podataka

Preuzmite skup podataka za ocjenjivanje sa sljedeće poveznice: http://zver6.zesoi.fer.hr:18080/labos_oi/submission_z1.zip

Primjer filea kojeg treba generirati možete preuzeti sa: http://zver6.zesoi.fer.hr:18080/labos_oi/zad1_submission_sample.csv

Odredite predikcije Vašeg najboljeg modela nad tim skupom, te ih stavite na Moodle.

```
from google.colab import drive
drive.mount('/content/drive/')
!cp * /content/drive/MyDrive
```

```
Mounted at /content/drive/
cp: -r not specified; omitting directory 'drive'
cp: -r not specified; omitting directory 'files'
cp: -r not specified; omitting directory 'sample_data'
cp: -r not specified; omitting directory 'src'
```

```
import os

image_dir = '/content/drive/MyDrive/Colab Notebooks/submission_z1'
# print(os.path.exists('/content/drive/MyDrive/Colab Notebooks/submission_z1'))


def absoluteFilePaths(directory):
    for dirpath,_,filenames in os.walk(directory):
```

```
        for f in filenames:
            yield os.path.abspath(os.path.join(dirpath, f))

paths = list(absoluteFilePaths(image_dir))
paths = sorted(paths, key=lambda x: int((x.split("_")[-1]).split(".")[0]))

from PIL import Image, ImageOps

transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Grayscale(num_output_channels=1),
    torchvision.transforms.Normalize((0.1307,), (0.3081,))
])

best_network = torch.load('/content/drive/MyDrive/dubok_i_sirok_model_0.01.pt')
# print (os.path.exists('/content/drive/MyDrive/dubok_i_sirok_model_0.01.pt'))
# print(best_network.eval())

images = []
paths_with_outputs = []

for p in paths:
  image = Image.open(p)
  image = ImageOps.grayscale(image)
  t = torchvision.transforms.ToTensor()(image)
  t = t.unsqueeze(0)
  t = t.to(device)
  with torch.no_grad():
    output = best_network(t)
    _, predicted = torch.max(output.data, 1)
    label = predicted.item()
    filename = p[37:]
    print(f"{filename}: {label}")
    paths_with_outputs.append([filename, label])
```

Saved successfully!

```
pd.DataFrame(paths_with_outputs).to_csv("prvi_zad_predaja.csv", header=["image_name", "num_label"])
# pd.DataFrame(paths_with_outputs, columns=["image_name", "true_label"]).to_csv("zad1_submission.csv")
```

```
s/submission_z1/zad1_70.png: 1
s/submission_z1/zad1_71.png: 7
s/submission_z1/zad1_72.png: 1
s/submission_z1/zad1_73.png: 5
s/submission_z1/zad1_74.png: 1
s/submission_z1/zad1_75.png: 4
s/submission_z1/zad1_76.png: 8
s/submission_z1/zad1_77.png: 2
s/submission_z1/zad1_78.png: 4
s/submission_z1/zad1_79.png: 9
s/submission_z1/zad1_80.png: 2
s/submission_z1/zad1_81.png: 0
s/submission_z1/zad1_82.png: 3
s/submission_z1/zad1_83.png: 9
s/submission_z1/zad1_84.png: 0
s/submission_z1/zad1_85.png: 8
s/submission_z1/zad1_86.png: 8
s/submission_z1/zad1_87.png: 7
s/submission_z1/zad1_88.png: 8
s/submission_z1/zad1_89.png: 9
s/submission_z1/zad1_90.png: 6
s/submission_z1/zad1_91.png: 1
s/submission_z1/zad1_92.png: 9
s/submission_z1/zad1_93.png: 2
s/submission_z1/zad1_94.png: 0
s/submission_z1/zad1_95.png: 0
s/submission_z1/zad1_96.png: 9
s/submission_z1/zad1_97.png: 4
s/submission_z1/zad1_98.png: 8
s/submission_z1/zad1_99.png: 1
s/submission_z1/zad1_100.png: 7
s/submission_z1/zad1_101.png: 0
s/submission_z1/zad1_102.png: 8
s/submission_z1/zad1_103.png: 8
s/submission_z1/zad1_104.png: 9
s/submission_z1/zad1_105.png: 6
s/submission_z1/zad1_106.png: 0
s/submission_z1/zad1_107.png: 2
s/submission_z1/zad1_108.png: 7
s/submission_z1/zad1_109.png: 3
s/submission_z1/zad1_110.png: 7
s/submission_z1/zad1_111.png: 8
s/submission_z1/zad1_112.png: 3
s/submission_z1/zad1_113.png: 8
s/submission_z1/zad1_114.png: 7
s/submission_z1/zad1_115.png: 7
s/submission_z1/zad1_116.png: 7
s/submission_z1/zad1_117.png: 7
s/submission_z1/zad1_118.png: 7
s/submission_z1/zad1_119.png: 3
s/submission_z1/zad1_120.png: 4
s/submission_z1/zad1_121.png: 6
s/submission_z1/zad1_122.png: 0
s/submission_z1/zad1_123.png: 8
s/submission_z1/zad1_124.png: 8
s/submission_z1/zad1_125.png: 1
s/submission_z1/zad1_126.png: 4
s/submission_z1/zad1_127.png: 6
```

Zadatak 2 - Pronalazak znamenki na slici i klasifikacija pronađene znamenke

Drugi zadatak je proširenje naučenog u prvom zadatku. Problem se proširuje - umjesto klasifikacije rukom pisane znamenke, naš problem je sada pronalazak rukom pisane znamenke na slici **i** klasifikacija.

Kao i u prethodnoj vježbi, dani su dijeli koda potrebnog za ostvarenje vježbe, a na Vama je da nadopunite dijelove koji nedostaju.

Skup podataka

Da bi mogli trenirati model za klasifikaciju i detekciju objekta na slici, moramo imati odgovarajući dataset. Koristimo postojeći MNIST dataset, a modificiramo ga tako da postavimo originalni MNIST uzorak na slučajnu poziciju na praznoj slici. Sljedeći kod generira takve uzorke, vraćajući modificiranu sliku, oznaku kategorije i poziciju znamenke na slici (*bounding box*).

```
class PositionMNIST(Dataset):

    def __init__(self, image_size=128, transform=None, train_set=False):
        self.image_size = image_size
        self.transform = transform

        self.set = torchvision.datasets.MNIST('./files/', train=train_set, download=True)
        self.position_cache = [-1] * len(self.set)

    def __len__(self):
        return len(self.set)

    def __getitem__(self, idx):
        if self.position_cache[idx] == -1:
            x_pos = int(np.random.uniform(0, self.image_size-29))
            y_pos = int(np.random.uniform(0, self.image_size-29))
                                      = (x_pos, y_pos)
Saved successfully!
                                _cache[idx]

        canvas = np.zeros((self.image_size, self.image_size, 1), dtype=np.uint8)
        canvas[y_pos:(y_pos+28), x_pos:(x_pos+28), 0] = self.set[idx][0]

        x_pos = float(x_pos)
        y_pos = float(y_pos)

        if self.transform is not None:
            canvas = self.transform(canvas)

        return canvas, self.set[idx][1], (x_pos, y_pos, x_pos+28, y_pos+28)
```

```
batch_size_train = 128
batch_size_test = 128
image_size = 128

train_set = PositionMNIST(train_set=True, image_size=image_size, transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
                                                                                    torchvision.transforms.Normalize((0.1307,), (0.3081,))]))

test_set = PositionMNIST(train_set=False, image_size=image_size, transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
                                                                                    torchvision.transforms.Normalize((0.1307,), (0.3081,))]))


train_loader = DataLoader(train_set, batch_size=batch_size_train, shuffle=True)
test_loader = DataLoader(test_set, batch_size=batch_size_test, shuffle=True)
```

Podzadatak a) - Vizualizacija podataka

Uzmite jedan uzorak pomoću data loadera i vizualizirajte ga. Neka u titleu piše klasa i lokacija.

```
examples = enumerate(test_loader)
batch_idx, (example_data, example_label, example_positions) = next(examples)
fig = plt.figure(figsize=(9, 4))
for i in range(6):
    plt.subplot(2,3,i+1)
    plt.tight_layout()
    # ----- OVO TREBA NADOPUNITI
    # čime???
    plt.imshow(example_data[i][0])
    plt.title(f"{example_label[i]}, {example_positions[0][i]}, {example_positions[1][i]}, {example_positions[2][i]}, {example_positions[3][i]}")
    # -----
    plt.xticks([])
    plt.yticks([])
```

8, 88.0, 4.0, 116.0, 32.0 0, 57.0, 7.0, 85.0, 35.0 8, 71.0, 47.0, 99.0, 75.0

Podzadatak b) - Pomoćne funkcije za treniranje

Nadopunite pomoćne funkcije za treniranje neuronskih mreža po principu naučenom u 1. zadatku. Temeljna razlika između pomoćne funkcije iz prethodnog zadatke i pomoćne funkcije u ovom zadatku je:

1. Rukovanje s podacima (ovdje ih ima više)
2. Drugi problem rješavamo, stoga trebamo drugačiju loss funkciju.

Loss funkcija će se u ovom slučaju sastojati od dva dijela - loss za klasifikaciju s kojim smo se već upoznali, i prosječan kvadrat greške (*mean squared error*) za određivanje pozicije. Loss će se računa kao:

$$\mathcal{L} = \text{NLLLoss}(\text{classification output}, \text{target}) + \frac{(x_1-\hat{x}_1)^2+(y_1-\hat{y}_1)^2+(x_2-\hat{x}_2)^2+(y_2-\hat{y}_2)^2}{128\cdot128}$$

Pri čemu su x i y točne pozicije objekta na slici, a $\hat{x}$ i $\hat{y}$ su modelom određene pozicije objekta.

```
def train_step(train_loader, epoch, device, verbose=True):
    train_losses = []
    train_counter = []

    network.train()

    for batch_idx, (data, target, position) in enumerate(train_loader):
        data = data.to(device)
        target = target.to(device)

        # -----
        network.zero_grad()
        output = network(data)
        loss_clsf = F.nll_loss(output[0], target)
        loss_bbox = (F.mse_loss(output[1], position[0]) + F.mse_loss(output[2], position[1]) + F.mse_loss(output[3], position[2]) + F.mse_loss(output[4], positio
```



```
        loss = loss_bbox + loss_clsf
        loss.backward()
        optimizer.step()
        # -------

        if batch_idx % log_interval == 0:
            if verbose:
                print('Train Epoch: {:5d} [{:5d}/{:5d} ({:2.0f}%)]\tLoss: {:.6f}'.format(
                    epoch,
                    batch_idx * len(data),
                    len(train_loader.dataset),
                    100. * batch_idx / len(train_loader),
                    loss.item()))

            train_losses.append(loss.item())
            train_counter.append((batch_idx*64) + ((epoch-1)*len(train_loader.dataset)))

    return train_losses, train_counter
```

Po istom principu iz 1. zadatka nadopunite funkciju za evaluaciju modela. U ovom slučaju mjerimo 3 stvari: sam loss, točnost klasifikacije i posebno loss za detekciju.

```
def test(test_loader, device, verbose=True):
    network.eval()

    test_loss_clsf = 0
    test_loss_bbox = 0
    correct = 0
    with torch.no_grad():
        for data, target, position in test_loader:
            data = data.to(device)
            target = target.to(device)

            # ----

            # ----------

    test_loss_clsf /= len(test_loader.dataset)
    test_loss_bbox /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)


    if verbose:
        print('\n[Test] Classification: Avg. loss: {:.4f}, Accuracy: {:5d}/{:5d} ({:2.2f}%) | Object detection: Avg. loss: {:.4f}\n'.format(
            test_loss_clsf,
            correct,
            len(test_loader.dataset),
            100. * correct / len(test_loader.dataset),
            test_loss_bbox))

    return test_loss_clsf, test_accuracy, correct, test_loss_bbox
```

Pomoćna funkcija za provođene eksperimenata iz prethodnog zadatka je iskoristiva do na praćenje dodatnih metrika. Proširite tu funkciju za ovaj zadatak.

```
def train_network(network, train_loader, test_loader, device='cpu'):
    train_losses = []
    train_counter = []
    test_losses_clsf = []
    test_accuracies = []
    test_losses_bbox = []
    test_counter = [i*len(train_loader.dataset) for i in range(n_epochs + 1)]

    # -----
    # ----------

    for epoch in range(1, n_epochs + 1):
        # ---------
        # ---------

    # ---------
    test_losses_total = [test_losses_clsf[i] + test_losses_bbox[i] for i in range(len(test_losses_clsf))]
    # -----------

    return train_losses, train_counter, test_losses_clsf, test_accuracies, test_losses_bbox, test_counter
```

Provođenje eksperimenata i analiza rezultata

Podzadatak c) - Izrada modela koji točno klasificira i locira objekt na slici

Kao i u prethodnom zadatku, prvo je potrebno podesiti parametre. Parametri su isti, no ponovimo:

- n_epochs - broj epoha eksperimenta
- learning_rate - stopa učenja
- log_interval - broj koraka između dva ispisa tijekom treniranja (ispis se dešava samo ako se funkcija poziva s argumentom verbose=True)
- ...aju izvršava eksperiment; "cuda" za GPU, "cpu" za CPU


```
n_epochs = 3
learning_rate = 0.0005
momentum = 0.9
log_interval = 100
device = 'cuda'
```

Temeljna razlika u arhitekturi modela ovog zadatka i arhitekture modela iz prethodnog zadatka je broj izlaza. Prošla neuronska mreža je imala 10 izlaznih neurona - svaki za jednu klasu. Ova neuronska mreža ima 14 izlaza - 10 za svaku klasu za klasifikacijski problem i 4 za svaku koordinatu rezultirajućeg bounding boxa objekta.

Na temelju iskustva iz 1. zadatka, nadopunite sljedeći model da bi riješili problem:

```
class Net(nn.Module):
    def __init__(self, image_size):
        super(Net, self).__init__()
        self.image_size = image_size

        # --------
        # Ovdje je dan primjer jednog ulaznog conv sloja i oblika izlaznih slojeva za orijentaciju

        self.conv1 = nn.Conv2d(1, 64, kernel_size=3)

        self.obj_x1_out = nn.Linear(1, 1)
        self.obj_y1_out = nn.Linear(1, 1)
        self.obj_x2_out = nn.Linear(1, 1)
        self.obj_y2_out = nn.Linear(1, 1)
        # --------

    def forward(self, x):
        # ---- ovdje nadopunite ostatak mreže
        x = self.max_pool(F.relu(self.conv1(x)))
        x = self.max_pool(F.relu(self.conv2(x)))
        # -------- izlaz za klasifikaciju
        clsf = F.log_softmax(x, dim=1)

        # -------- izlaz za detekciju
        x1 = F.relu(self.obj_x1_out(x1))
        y1 = F.relu(self.obj_y1_out(y1))
        x2 = F.relu(self.obj_x2_out(x2))
        y2 = F.relu(self.obj_y2_out(y2))

        return clsf, x1.squeeze(), y1.squeeze(), x2.squeeze(), y2.squeeze()
```

```
number_of_params = get_number_of_model_parameters(network)
print("Broj parametara u modelu:", number_of_params)
```

```
Broj parametara u modelu: 103856
```

```
network = Net(image_size).to(device)
optimizer = optim.Adam(network.parameters(), lr=learning_rate)
```

```
train_losses, train_counter, test_losses_clsf, test_accuracies, test_losses_bbox, test_counter = train_network(network, train_loader, test_loader, device)
```

Vizualizacija metrika uspješnosti

Vizualizirajte si sve metrike na sljedećem grafu: train_losses, test_losses_total, test_losses_clsf i test_losses_bbox. Pripazite što vam je na x osi!

```
test_losses_total = np.array(test_losses_clsf) + np.array(test_losses_bbox)

fig = plt.figure(figsize=(32, 7))
# ----

# ----------

plt.legend(loc='upper right')
plt.xlabel('Number of samples')
plt.ylabel('Loss')
plt.grid()
```

Vizualni pregled - što model estimira?

[ ] ↳ 2 cells hidden

Podzadatak d) - Evaluacija na neviđenom skupu podataka

Preuzmite skup podataka za ocjenjivanje sa sljedeće poveznice: http://zver6.zesoi.fer.hr:18080/labos_oi/submission_z2.zip

Primjer filea kojeg treba generirati možete preuzeti sa: http://zver6.zesoi.fer.hr:18080/labos_oi/zad2_submission_sample.csv

Odredite predikcije Vašeg najboljeg modela nad tim skupom, te ih stavite na Moodle.

Saved successfully!