Metody Inteligencji Obliczeniowej
Laboratorium 4

Sieci konwulsyjne

Kamil Pyla

Wyniki zadania 1:
Jako pierwsze wnioski zanim pojawiły się pierwsze wyniki, mogę zapisać, że najistotniejszą kwestią przy projektowaniu sieci konwulsyjnej jest odpowiedni dobór warstw oraz konwerterów, który jest uzależniony od danych wejściowych. Zbiór FashionMNIST zawiera zdjęcia czarno-białe, zatem, nie można było posłużyć się siecią użytą w przykładzie do klasyfikacji zdjęć kolorowych (3 wymiary RGB).
Drugim wnioskiem jest konieczność użynia odpowiedniego transformera przy pobieraniu danych, w tym przypadku, ze względu na jednobarwność zdjęć, transformer też miał jeden wymiar:

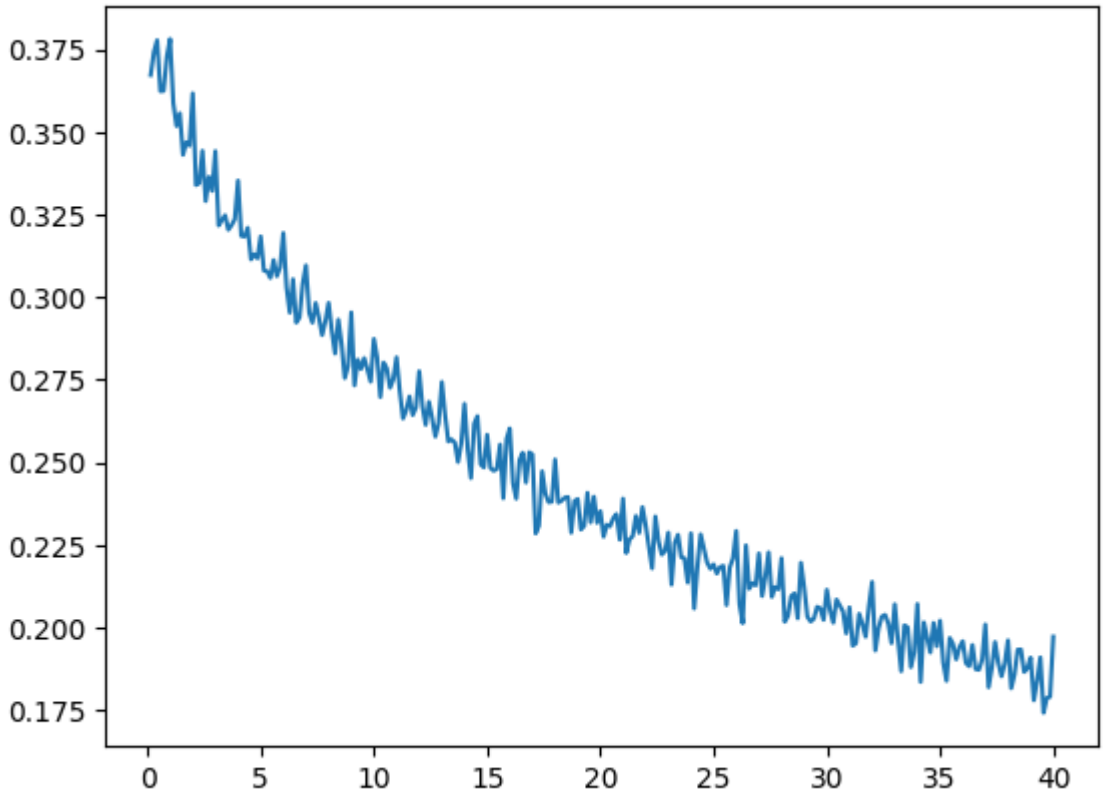
```
# zdjęcia są czarno-białe mają jeden wymiar
transform =  transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5), (0.5))
])
```

Parametry pierwszej sieci (kod klasy sieci):

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

```
    self.fc1 = nn.Linear(28*28, 256)
    self.fc2 = nn.Linear(256, 128)
    self.fc3 = nn.Linear(128, 64)
    self.fc4 = nn.Linear(64, 10)
    self.dropout = nn.Dropout(0.2)

  def forward(self, x):
    x = x.view(x.shape[0], -1)
    x = self.dropout(F.relu(self.fc1(x)))
    x = self.dropout(F.relu(self.fc2(x)))
    x = self.dropout(F.relu(self.fc3(x)))
    x = self.fc4(x)
    return x
```

Cztery warstwy ukryte. Wartość straty zapisywałem co 2000 * 4 zjęć (rozmiar batcha miał 4) dla każdej epoki.



Rys 1. wykres straty w zależności od liczy epok

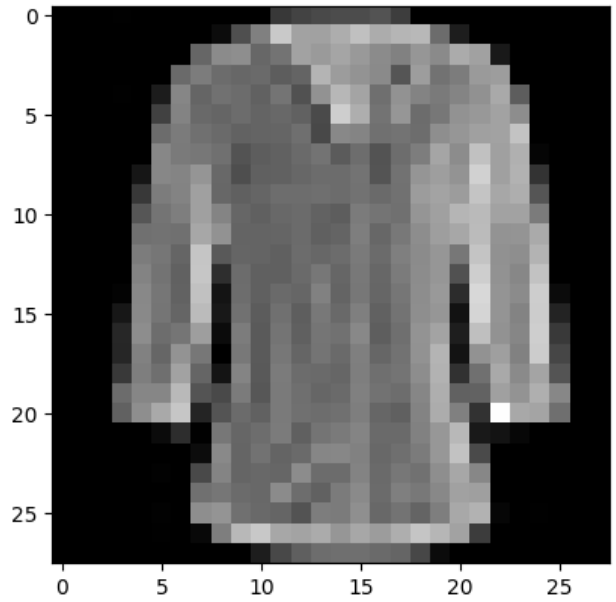
Następnie wykonałem operacje na zbiorze testowym, sprawdzając dokładność dopasowania, oraz przykładowe obrazy, które zostały niewłaściwie sklasyfikowane.

Dane testowe sprawdzałem w batch’ach wielkości 1000 zdjęć, dla każdego wypisałem macierz pomyłek oraz przykładowe źle sklasyfikowane zdjęcie razem z jego poprawną klasyfikacją

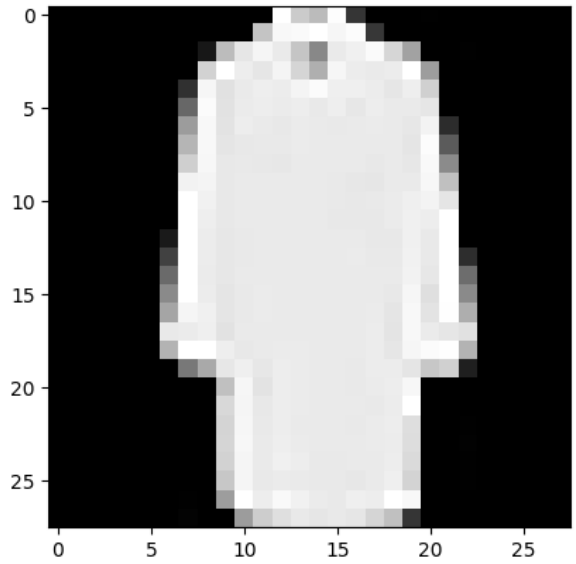
'T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot'

```
[[ 93   0   2   3   0   0   7   1   1   0]
 [  0 101   0   4   0   0   0   0   0   0]
 [  2   0  88   2   6   0  13   0   0   0]
 [  3   0   1  86   1   0   2   0   0   0]
 [  0   0   8   7  92   0   8   0   0   0]
 [  0   0   0   0   0  83   0   2   0   2]
 [ 12   0  10   2   5   0  68   0   0   0]
 [  0   0   0   0   0   1   0  93   0   1]
 [  2   0   0   0   0   1   1   0  91   0]
 [  0   0   0   0   0   1   0   3   0  91]]
```

dokladnosc dopasowania: 0.886



poprawna klasa: Shirt, klasyfikacja: T-shirt/top



poprawna klasa: Pullover, klasyfikacja: Dress

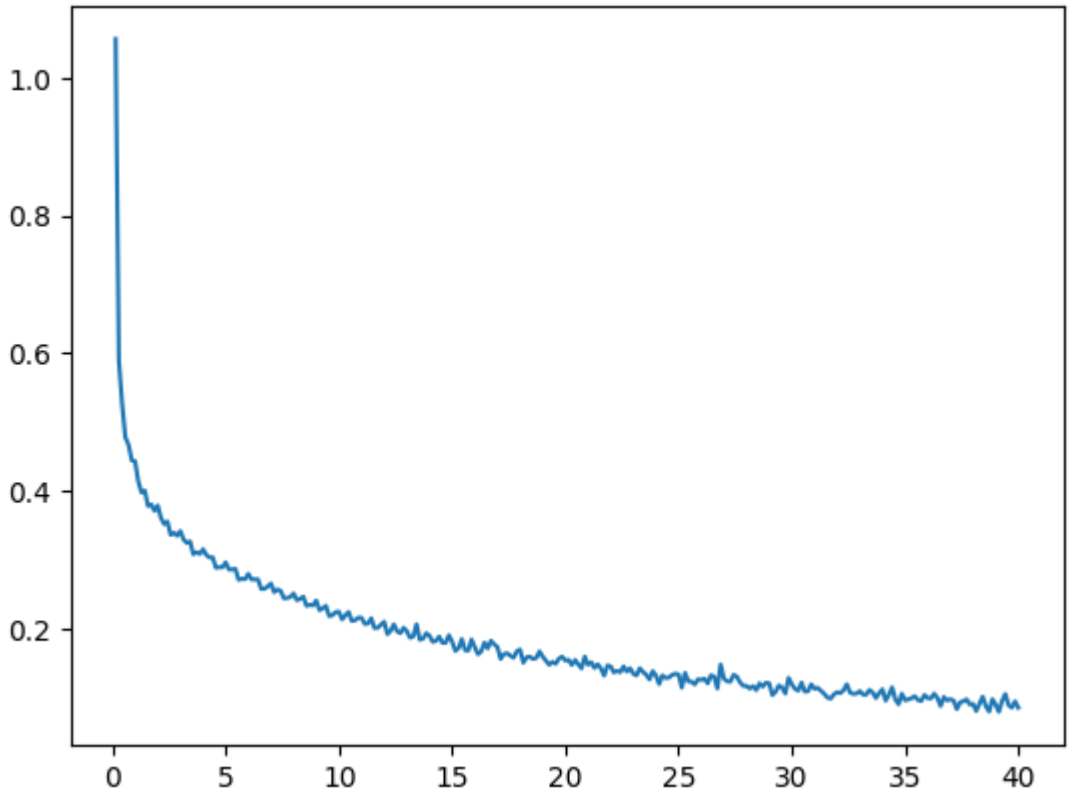
dokladnosc w całym zbiorze: 0.8917

Dokładność klasyfikacji jest bardzo dobra, jak widać z macierzy pomyłek T-shirt jest mylony z koszulą, sukieną, sweter z koszulą, generalnie pomyłki zachodzą w podobnych częściach garderoby, co jest bardzo dobrym osiągnięciem.

Parametry drugiej sieci:

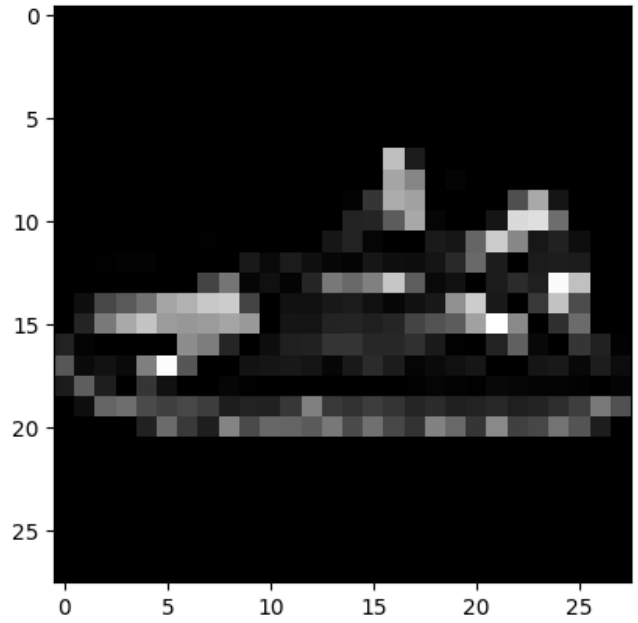
```
class Net_1(nn.Module):
  def __init__(self):
    super().__init__()
    self.fc1 = nn.Linear(28*28, 256)
    self.fc2 = nn.Linear(256, 128)
    self.fc3 = nn.Linear(128, 64)
    self.fc4 = nn.Linear(64, 10)

  def forward(self, x):
    x = x.view(x.shape[0], -1)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = F.relu(self.fc3(x))
```

```
    x = self.fc4(x)
    return x
```

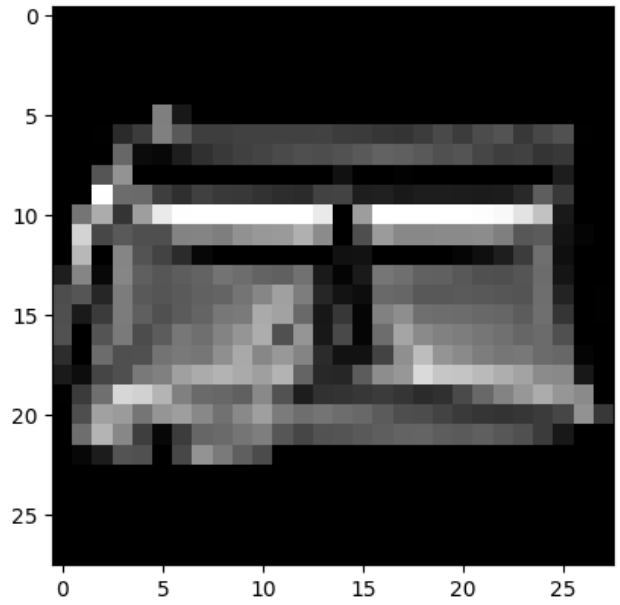


wykres straty w zależności od liczy epok dla sieci 2

```
[[ 94   0   1   3   0   0   3   0   0   0]
 [  0  85   0   1   0   0   0   0   0   0]
 [  2   0  83   1  13   0   9   0   0   0]
 [  1   0   2  96   4   0   0   0   1   0]
 [  1   0   9   3  89   0   5   0   0   0]
 [  0   0   0   0   0  97   0   2   0   2]
 [ 23   1   6   3   4   0  57   0   0   0]
 [  0   0   0   0   0   4   0  81   0   1]
 [  2   0   2   0   0   0   1   1 104   0]
 [  0   0   0   0   0   3   0   6   0  94]]
```



poprawna klasa: Sneaker, klasyfikacja: Sandal



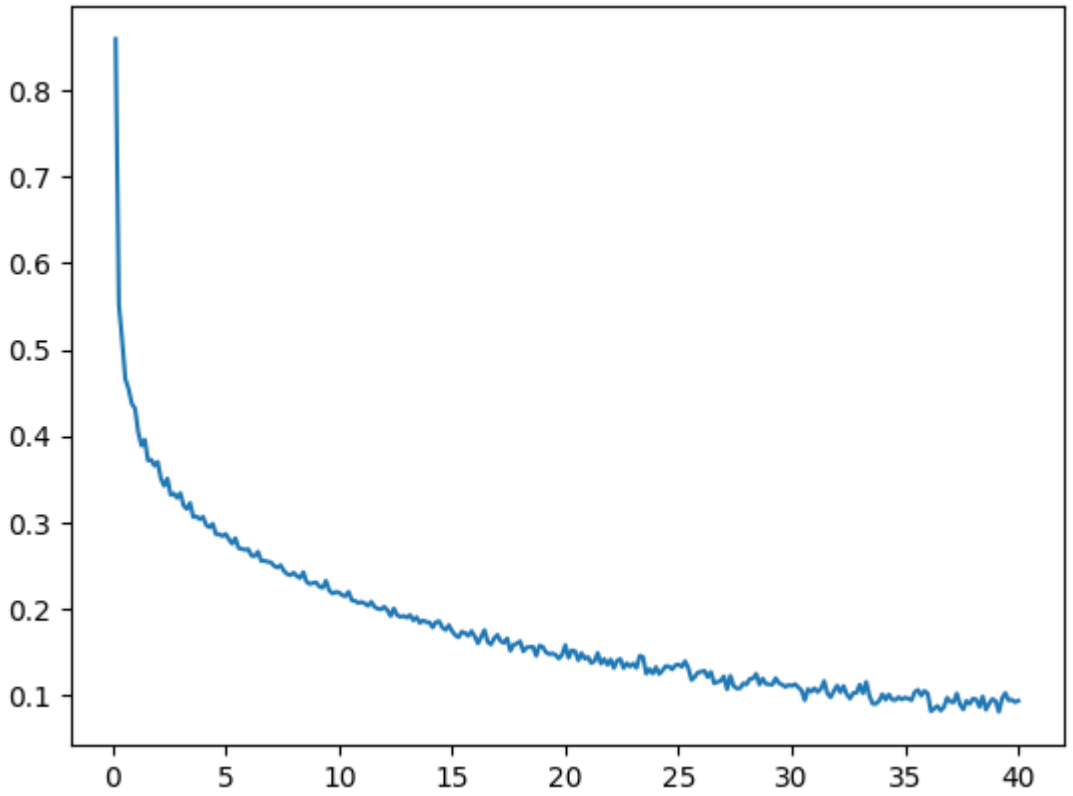
poprawna klasa: Bag, klasyfikacja: Sandal

Dokladnosc w calym zbiorze: 0.8781

Pomyłki w klasyfikacji również dotyczyły podobnych akrtykułów. Buty do biegania - sandały.

```
class Net_2(nn.Module):
  def __init__(self):
    super().__init__()
    self.fc1 = nn.Linear(28*28, 256)
    self.fc2 = nn.Linear(256, 64)
    self.fc3 = nn.Linear(64, 10)

  def forward(self, x):
    x = x.view(x.shape[0], -1)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

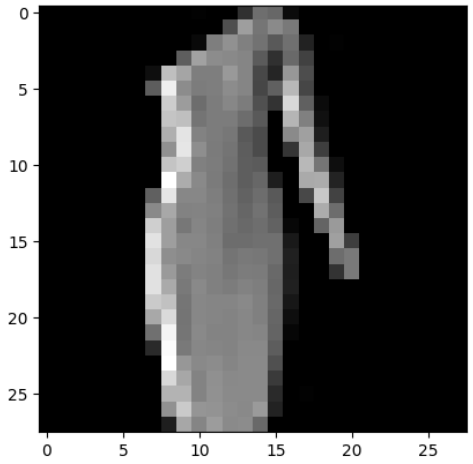
3 sieć, 3 warstwy ukryte



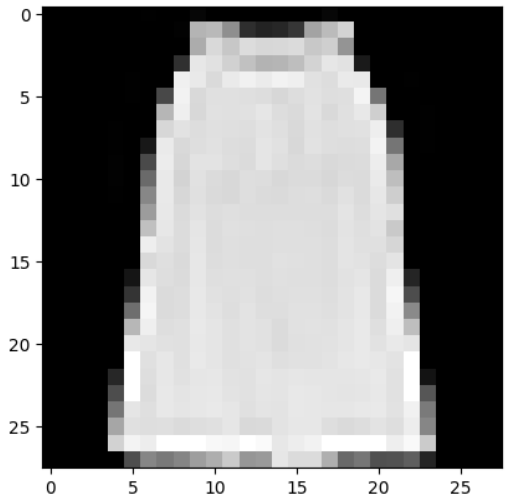
wykres straty w zależności od liczy epok dla sieci 2

[[76 0 3 5 0 0 9 0 0 0]
 [1 93 0 4 0 0 0 0 0 0]

```
[  3   0  91   1   4   0   4   0   0   0]
[  2   1   0  90   2   0   1   0   1   0]
[  1   0  12   3  83   0   5   0   0   0]
[  0   0   0   0   0 105   0   3   0   0]
[ 20   0   4   1   4   0  71   0   0   0]
[  0   0   0   0   0   3   0 102   0   0]
[  1   0   1   1   0   2   1   1  92   0]
[  0   0   0   0   0   0   0   6   0  87]]
```



poprawna klasa: Dress, klasyfikacja: Coat



poprawna klasa: Shirt, klasyfikacja: T-shirt/top

dokladnosc w calym zbiorze: 0.8773

Wnioski: najważniejszą rzeczą przy projektowaniu sieci konwulsujnych jest odpowiedni dobór funkcji w warstwach sieci ukrytych. Wejściowa warsta powinna mieć rozmiar pikseli na jednym obrazie (28*28 = 784) a ostasnia ilości klas (10) Z wykresu straty można odczytać, że wartość straty maleje coraz wolniej w zależności od ilości epok, a ilość epok przekłada się wprost, na czas uczenia sieci, który w tym przypadku wynosił około 35 - 45 minut - do całkiem dużo, dlatego należy dokonywać optymalizacji w tym kierunku.

Link do repozytorium z kodem:
https://github.com/KamilPyla/MIO_2023/tree/master/lab_04