SIECI NEURONOWE

Raport z laboratorium 6

Student: Van Hien Le - 257795

Narzędzia i technologie

- Python, Pandas, Numpy, Scikit-learn, PyTorch
- Jupyter Notebook

Dataset

Zbiorem danych jest kolekcja FashionMNIST, domyślnie dostępna w PyTorch.

Dostęp do zbioru danych: data = torchvision.datasets.FashionMNIST('path', download=True)

Architektura sieci

W tym ćwiczeniu skupiłem się na zbudowaniu sieci neuronowej wykorzystującej warstwy konwolucyjne CNN i operator max pooling. Poniżej znajdują się kluczowe elementy architektury:

- **Warstwa Konwolucyjna (Conv2d):** nn.Conv2d(in_channels=1, out_channels=num_output_channels, kernel_size=filter_size) służy do wykrywania cech w obrazach.
- **Max Pooling:** nn.MaxPool2d(kernel_size=pool_size) redukuje wymiary obrazu, zachowując istotne informacje.
- **Warstwa Liniowa (LazyLinear):** nn.LazyLinear(out_features=num_classes) użyta do klasyfikacji. Automatycznie dostosowuje wymiary na podstawie danych wejściowych.

Hiperparametry

Eksperymenty przeprowadzono z różnymi ustawieniami:

- **Liczba kanałów wyjściowych:** testowano różne wartości 16, 32 i 128
- **Rozmiar filtra:** eksperymentowano z różnymi rozmiarami 3x3, 5x5 i 11x11
- **Rozmiar okna poolingu:** testowano różne rozmiary 2x2, 3x3, 4x4
- **Zaburzenia danych:** dodawano szum gaussowski o różnych odchyleniach standardowych do danych treningowych i/lub testowych.

Domyślne hiperparametry:

- **Learning Rate:** 0.001
- **Number of Epochs:** 20
- **Optimizer:** Adam
- **Loss Function:** Cross-Entropy loss

Konfiguracja i procedura eksperymentów

- Ładowanie i przetwarzanie danych: FashionMNIST załadowany za pomocą DataLoader, obrazy przekształcone w tensory i znormalizowane.

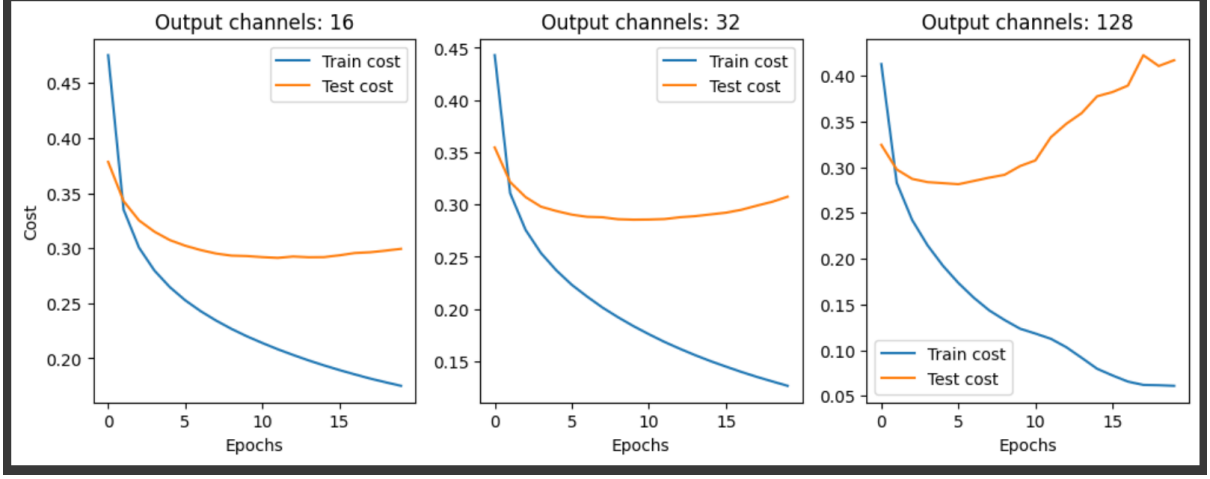
- Trenowanie cieci: Uczenie sieci z różnymi konfiguracjami hiperparametrów.

- Testowanie sieci: Ocena wydajności sieci na danych testowych z i bez dodanego szumu gaussowskiego.

Wyniki eksperymentów

- Liczba kanałów wyjściowych: testowano różne wartości 16, 32 i 128

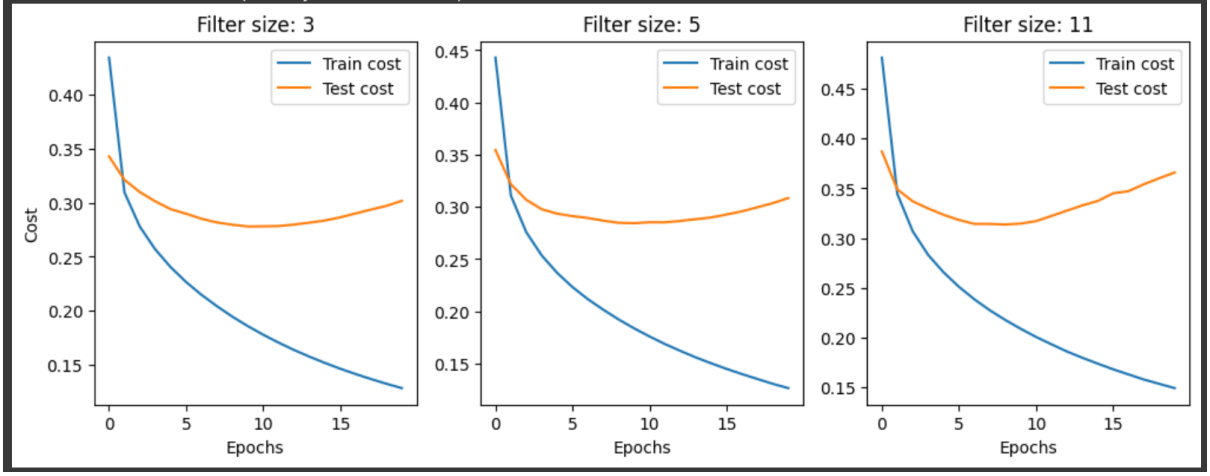
```
Output channels: 16 :  tensor(0.8988, device='cuda:0')
Output channels: 32 :  tensor(0.9045, device='cuda:0')
Output channels: 128 :  tensor(0.9067, device='cuda:0')
```



Tutaj widzimy, że wraz ze wzrostem liczby kanałów wyjściowych zwiększa się odstęp między krzywą treningową a krzywą testową. Model z liczbą kanałów 128 wykazuje overffiting, jednak ma najwyższy wynik Accuracy – 90,6 proc.

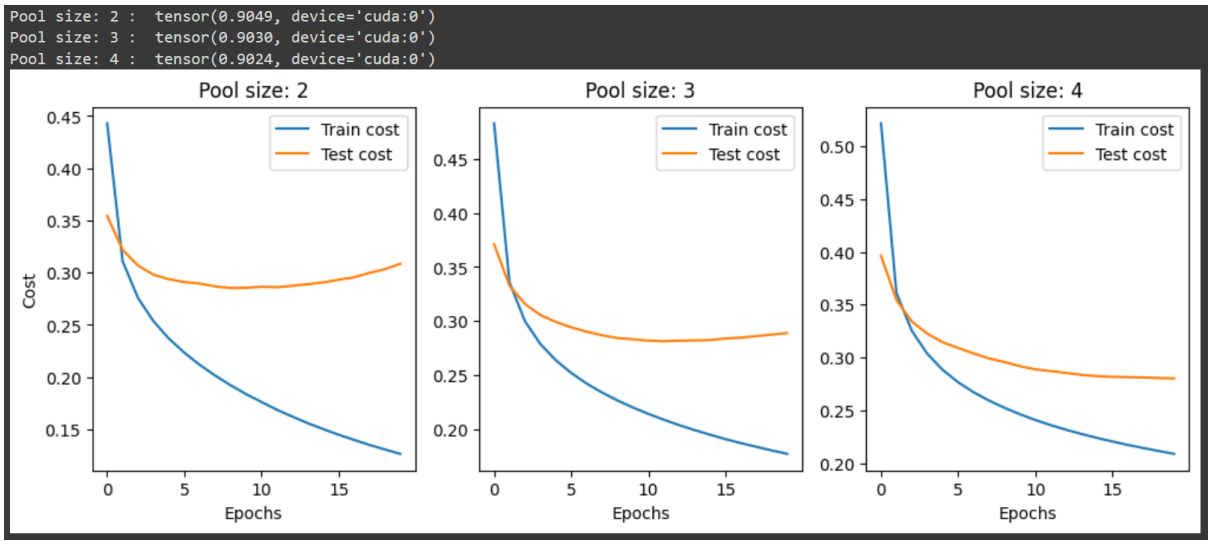
- Rozmiar filtra: eksperymentowano z różnymi rozmiarami 3x3, 5x5 i 11x11

```
Filter size: 3 :  tensor(0.9026, device='cuda:0')
Filter size: 5 :  tensor(0.9048, device='cuda:0')
Filter size: 11 :  tensor(0.8950, device='cuda:0')
```



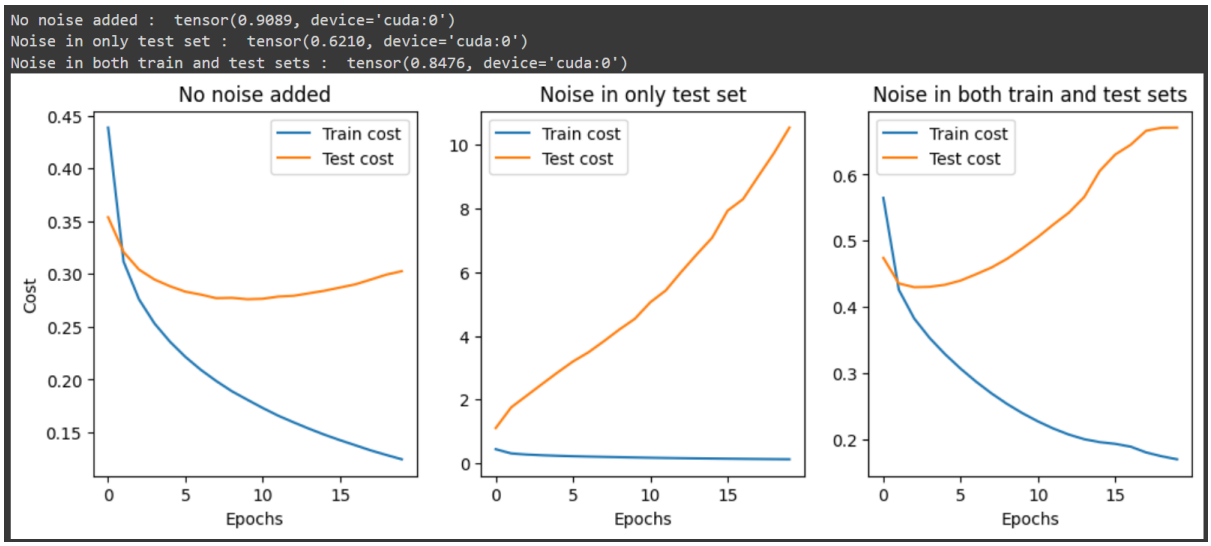
W eksperymencie z różnymi rozmiarami filtrów widać, że procesy uczenia są do siebie dość podobne, jednak model z rozmiarem filtra 5 wydaje się najbardziej optymalny, gdyż ma najwyższy wynik Accuracy – 90,4 proc.

- Rozmiar okna poolingu: testowano różne rozmiary 2x2, 3x3, 4x4



Tutaj wraz ze wzrostem rozmiaru okna proces uczenia się staje się wolniejszy, zwiększają się także koszty trenowania i testowania. Natomiast, najwyższy wynik uzyskał pierwszy model z rozmiarem okna 2x2

- Zaburzenia danych: dodawano szum gaussowski o różnych odchyleniach standardowych do danych treningowych i/lub testowych.



W przypadku dodania szumu Gaussa tylko do zbioru danych testowych koszty testowania są niezwykle wysokie.

W przypadku dodania szumu do obu zbiorów danych, wyniki stają się lepsze pod względem kosztów testowania, a także wyniku Accuracy.