Klasyfikacja Fashion MNIST przy użyciu CNN

Sprawozdanie z projektu: Podstawy Sieci Neuronowych

Maksymilian Lech (280136)
Kornel Lewandowski (280101)
Wiktor Hebdowski (275964)

Politechnika Wrocławska

22 stycznia 2026

Agenda

1. Analiza problemu
2. Architektura rozwiązania
3. Eksperymenty i Wyniki
4. Analiza i Wnioski

Cel projektu i opis danych

Cel: Stworzenie modelu klasyfikującego obrazy ubrań.

Zbiór danych: Fashion MNIST

- **Rozmiar:** 70 000 obrazów (60k trening, 10k test).
- **Format:** 28×28 pikseli, skala szarości.
- **Klasy:** 10 kategorii (np. T-shirt, Spodnie, Buty).

Nasze podejście: Wybraliśmy **Sieć Konwolucyjną (CNN)**, ponieważ:

- Efektywnie wykrywa lokalne wzorce (krawędzie, tekstury).
- Zapewnia odporność na przesunięcia obiektów.
- Posiada mniej parametrów niż sieci w pełni połączone.

Architektura naszej sieci (FashionCNN)

Zaprojektowaliśmy model składający się z dwóch głównych sekcji:

1. **Ekstrakcja cech (Konwolucje):**
   - 3 warstwy Conv2d ($32 \rightarrow 64 \rightarrow 128$ filtrów).
   - **Batch Normalization** po każdej warstwie (stabilizacja).
   - **Max Pooling** (redukcja wymiarów).
   - Funkcja aktywacji: ReLU.
2. **Klasyfikacja (Fully Connected):**
   - Spłaszczenie danych (Flatten).
   - Warstwa ukryta: 256 neuronów.
   - **Dropout (0.5)** – zapobieganie overfittingowi.
   - Wyjście: 10 neuronów (klasy).

Implementacja i Narzędzia

Do realizacji projektu wykorzystaliśmy:

- **PyTorch Torchvision:** Budowa modelu i transformacje danych.
- **CUDA:** Przeniesienie obliczeń na GPU (skrócenie czasu treningu z minut do sekund).
- **Modułowość:**
  - `config.py` – oddzielenie parametrów od kodu.
  - `experiments.py` – automatyzacja serii testów.
- **Early Stopping:** Przerwanie treningu, gdy walidacja przestaje się poprawiać (parametr `patience`).

Przeprowadzone eksperymenty

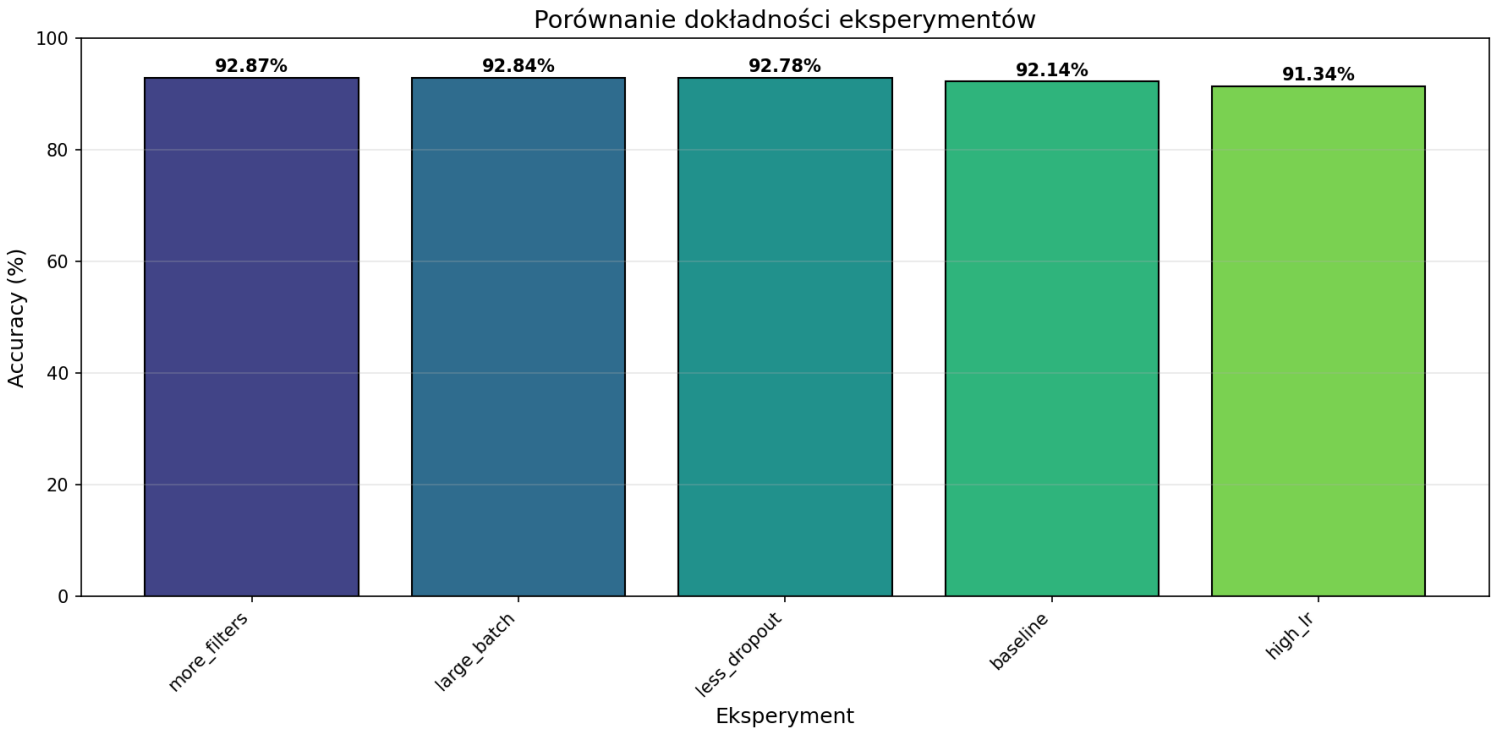
Zrealizowaliśmy 5 scenariuszy testowych, aby dobrać optymalne parametry:

Eksperyment	**Test Acc**	**Epoki**	**Wniosek**
Baseline	92.14%	13	Dobry punkt wyjścia
High LR (0.01)	91.34%	20	Niestabilny trening
Large Batch (128)	92.84%	19	Stabilniejsze gradienty
More Filters	**92.87%**	**17**	**Najlepszy wynik**
Less Dropout	92.78%	17	Ryzyko przeuczenia

Tabela: Zestawienie wyników (posortowane wg Accuracy).

Porównanie skuteczności

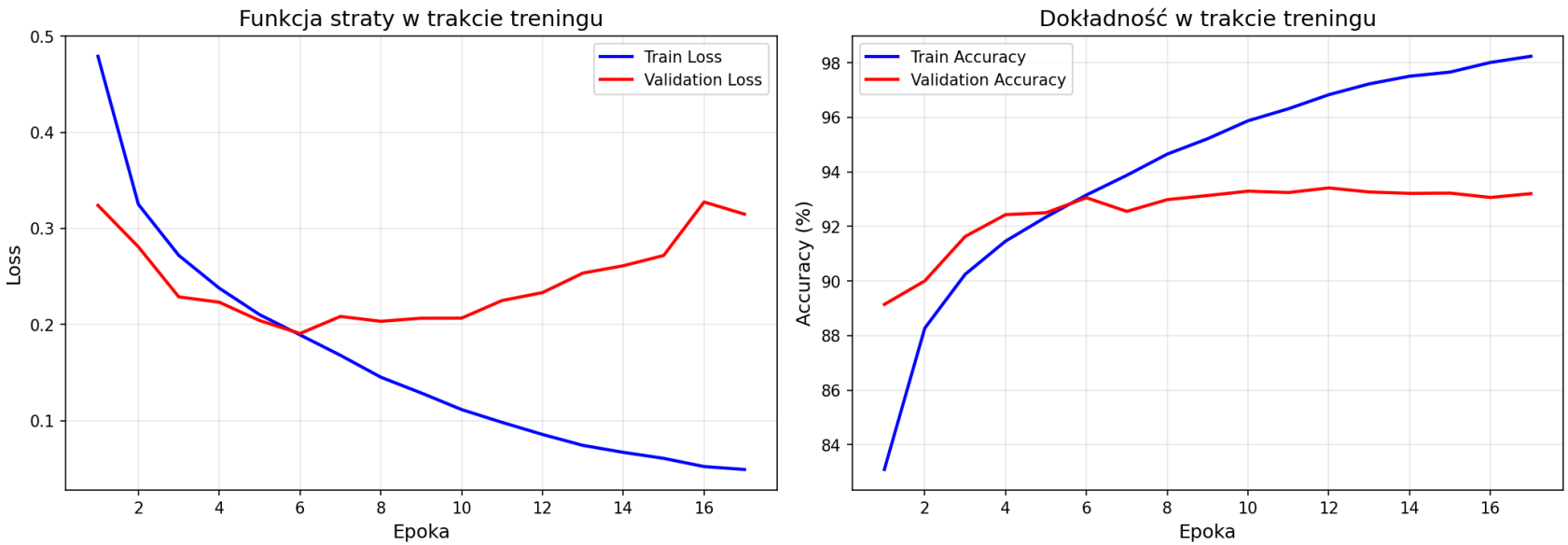
Wykres porównujący dokładność testową wszystkich modeli.



Rysunek: Porównanie Accuracy dla różnych konfiguracji.

Proces uczenia (Najlepszy model)

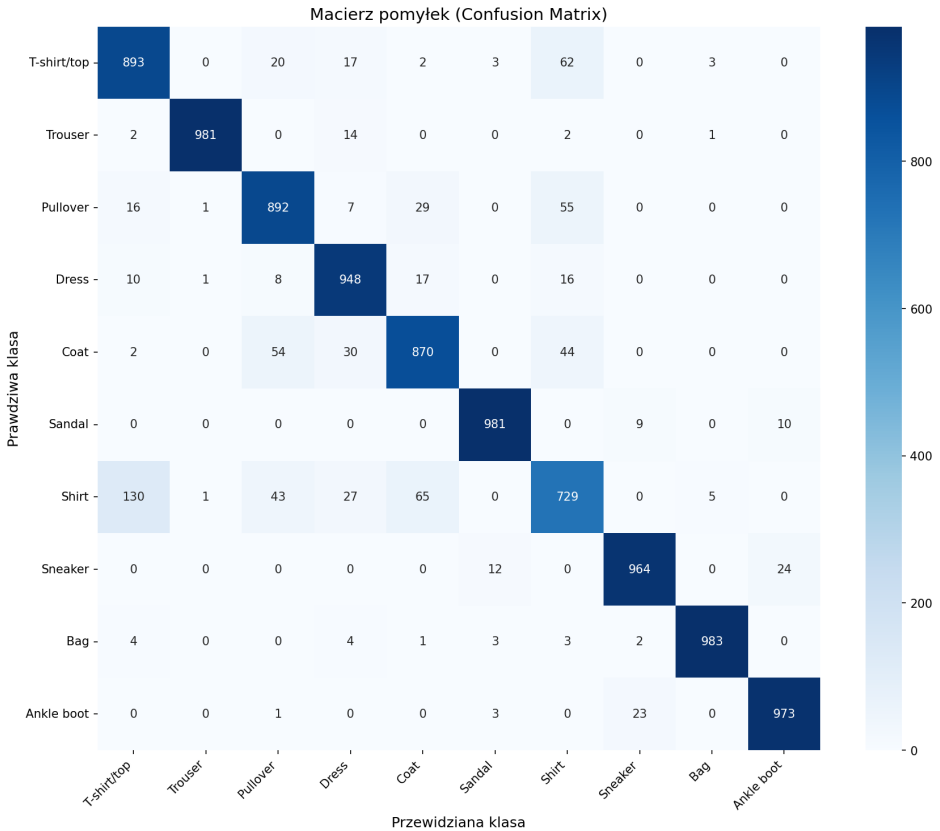
Przebieg funkcji straty i dokładności dla modelu *More Filters*.



Rysunek: Krzywe uczenia: Train vs Validation.

Analiza błędów (Macierz pomyłek)

Gdzie model popełnia błędy?



Wnioski z analizy:

- Model osiąga wysoką skuteczność (>92%).
- Najczęstsze pomyłki dotyczą klas podobnych wizualnie:
  - T-shirt $\leftrightarrow$ Shirt (Koszula)
  - Pullover $\leftrightarrow$ Coat (Płaszcz)
- Klasy wyróżniające się (np. Buty, Torebki) są rozpoznawane bezbłędnie.

Napotkane problemy i sukcesy

Problemy i rozwiązania:

- Problem: Błędy przy wczytywaniu plików Gzip.
  $\rightarrow$ **Rozwiązanie:** Automatyczne wykrywanie formatu.
- Problem: Wolny trening na CPU.
  $\rightarrow$ **Rozwiązanie:** Implementacja obsługi GPU/CUDA.

Kluczowe sukcesy:

- Osiągnięcie dokładności **92.87%** na zbiorze testowym.
- Skuteczna implementacja mechanizmu Early Stopping.
- Pełna automatyzacja procesu eksperymentowania.

Podsumowanie

Wnioski końcowe

1. **Architektura:** Zwiększenie liczby filtrów (model *More Filters*) przyniosło największą poprawę wyników.
2. **Stabilizacja:** Batch Normalization jest kluczowy dla szybkiej zbieżności sieci.
3. **Hiperparametry:** Learning Rate rzędu 0.001 okazał się optymalny; wyższe wartości destabilizowały proces.

Dziękujemy za uwagę!