AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ MATEMATYKI STOSOWANEJ

ROZPOZNAWANIE UBRAŃ PRZY UŻYCIU SIECI KONWOLUCYJNYCH

Autorzy: Magdalena Leśniowska, Emilia Gawenda, Marlena Kozieł
Kierunek studiów: Matematyka

Kraków, 2024

Spis treści

ROZDZIAŁ 1

Wstęp

Rozpoznawanie obrazów za pomocą sieci neuronowych to dynamicznie rozwijająca się dziedzina sztucznej inteligencji, która znalazła zastosowanie w wielu różnych obszarach, od medycyny po autonomiczne pojazdy. Jednym z przydatnych zastosowań jest automatyczna klasyfikacja ubrań, co może przynieść znaczące korzyści w branży mody, e-commerce oraz w systemach rekomendacyjnych.

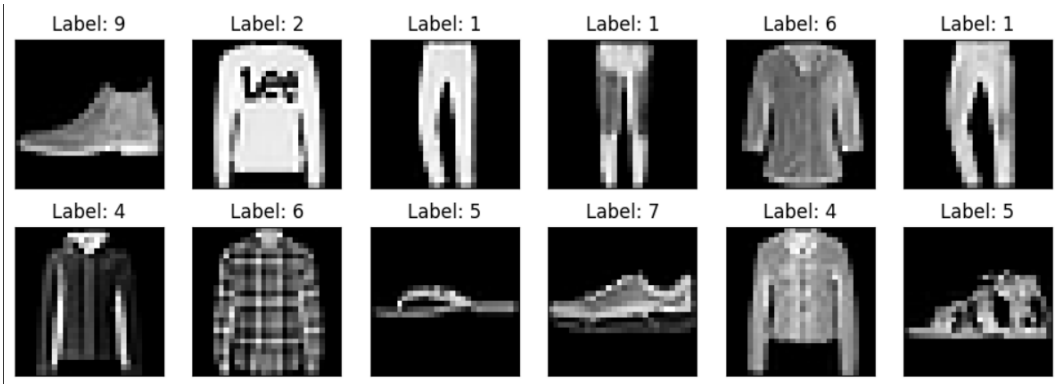
Nasz projekt skupia się na klasyfikacji ubrań ze względu na ich kategorie. W celu realizacji tego zadania wykorzystałyśmy konwolucyjne sieci neuronowe. CNN charakteryzują się zdolnością do automatycznego ekstraktowania cech z obrazów poprzez warstwy konwolucyjne, co czyni je idealnymi do analizy danych obrazowych.

W niniejszym projekcie przedstawiamy krok po kroku proces budowy modeli CNN do klasyfikacji ubrań, od przygotowania danych, przez budowę i trening modeli, aż po ich ewaluację i optymalizację.

Realizacja tego projektu stanowi znaczący krok w kierunku zrozumienia i zastosowania konwolucyjnych sieci neuronowych do złożonych zadań klasyfikacyjnych.

1.1 Opis Danych

Fashion MNIST to zbiór danych składający się z 70,000 obrazów (60,000 w zbiorze treningowym i 10,000 w zbiorze testowym) nie posiadający brakujących wartości. Dane są publicznie dostępne w repozytorium na GitHubie pod adresem: `https://github.com/zalandoresearch/fashion-mnist`. Zbiór danych zawiera 10 równomiernie rozłożonych klas ubrań: T-shirt/top, Spodnie, Sweter, Sukienka, Płaszcz, Sandały, Koszula, Buty sportowe, Torba, Botki. Każdy obrazek ma wymiary 28x28 pikseli i jest w skali szarości. Dane te zostały opracowane przez firmę Zalando w celu stworzenia bardziej złożonego zadania klasyfikacji obrazów niż klasyczny zbiór MNIST, który zawierał ręcznie pisane cyfry. Przykładowe obrazy z etykietami przedstawiłyśmy poniżej.



Rysunek 1.1: Przykładowe obrazki ze zbioru

ROZDZIAŁ 2

Przygotowanie Danych

Aby przygotować dane przeprowadziłyśmy proces, który obejmował następujące kroki:

- Wczytanie danych z biblioteki torchvision.
- Podział danych: Zbiór danych podzieliłyśmy na zbiory treningowy, walidacyjny i testowy.
- Wizualizacja danych, aby lepiej zrozumieć ich strukturę i charakterystykę.
- Augmentacja: Żeby zwiększyć różnorodność zbioru danych treningowych i tym samym poprawić ogólność naszego modelu i zapobiec przeuczeniu wykonałyśmy losową symetrię pionową na zbiorze uczącym.
- Użycie DataLoadera: Do efektywnego ładowania danych w małych partiach użyłyśmy klasy DataLoader z biblioteki PyTorch.

ROZDZIAŁ 3

Budowa Modelu

3.1 Wybór Architektury Sieci Neuronowej

Do zadania klasyfikacji obrazów wybrałyśmy konwolucyjną sieć neuronową (CNN), która jest dobrze dostosowana do przetwarzania danych obrazowych. Architektura sieci obejmuje warstwy konwolucyjne z max poolingiem i dropoutem, a następnie warstwy liniowe.

Konwolucyjne sieci neuronowe są jednymi z najpotężniejszych i najczęściej używanych narzędzi w dziedzinie wizji komputerowej. CNN są szczególnie skuteczne w analizie danych obrazowych, ponieważ potrafią automatycznie i hierarchicznie ekstraktować cechy z obrazów. Kluczowym elementem CNN są warstwy konwolucyjne, które stosują operację splotu (konwolucji) w celu wykrywania różnorodnych cech, takich jak krawędzie, tekstury i kształty na różnych poziomach abstrakcji.

Podstawowe komponenty CNN obejmują warstwy konwolucyjne, warstwy poolingowe oraz w pełni połączone warstwy. Warstwy konwolucyjne składają się z filtrów (jąder), które przesuwają się po obrazie, wykrywając różne wzorce. Warstwy poolingowe redukują rozmiar danych, zmniejszając liczbę parametrów i obciążenie obliczeniowe, co pomaga w zapobieganiu przeuczeniu (overfitting).

Kolejnym ważnym elementem CNN jest zastosowanie funkcji aktywacji, takich jak ReLU, które wprowadzają nieliniowość do modelu, pozwalając na modelowanie bardziej złożonych wzorców. Na końcu sieci znajdują się warstwy w pełni połączone, które klasyfikują złożone cechy wyekstrahowane przez poprzednie warstwy.

3.2 Implementacja Modelu

Modele zostały zaimplementowane przy użyciu PyTorch, popularnej biblioteki do uczenia maszynowego. Wszystkie treningi przeprowadziłyśmy z użyciem GPU w celu przyspieszenia uczenia.

3.2.1 Architektura 1

Pierwsza zaproponowana architektura składa się z dwóch bloków opisanych poniżej:

- warstwa konwolucyjna z 1 kanałem wejściowym i 64 wyjściowymi (za drugim razem 64 wejściowe i 128 wyjściowych), rozmiarem jądra ustawionym na 5, paddingiem ustawionym na 2 i funkcją aktywacji ReLU
- drop out z prawdopodobieństwem 0.5
- max pooling z rozmiarem jądra równym 2

Następnie dane z wielu kanałów ulegają spłaszczeniu do jednowymiarowego wektora o rozmiarze 6272. W dalszej części znajdują się warstwy gęsto połączone (MLP) o liczbie neuronów kolejno 1024 i 10, gdyż taka jest liczba rozpoznawanych klas. W warstwach liniowych ponownie używamy funkcji aktywacji ReLU, ale nie stosujemy na końcu funkcji softmax, gdyż jest ona wbudowana w używaną przez nas funkcję celu Cross Entropy Loss dla zapewnienia lepszej stabilności numerycznej.

Wykonałyśmy dwa treningi na dwóch wartościach współczynnika uczenia: 0.001 i 0.0005.

3.2.2 Architektura 2

Drugi model rozszerza poprzedni, dodając więcej warstw konwolucyjnych oraz zmianę drop out'u.

- Wykorzystuje cztery warstwy konwolucyjne z kolejno 32, 64, 128 i 256 kanałami wyjściowymi, rozmiarem jądra ustawionym na 5, paddingiem ustawionym na 2 i funkcją aktywacji ReLU
- drop out z prawdopodobieństwem 0.3
- max pooling z rozmiarem jądra równym 2.

Następnie wykonuje spłaszczenie i dodaje dwie warstwy liniowe z funkcją aktywacji ReLU.

Model został wytrenowany na 20 epokach z optymalizatorem AdamW i lr=0.001. Uzyskano wynik dokładności na poziomie około 98.6% na zbiorze testowym.Otrzymany wynik był dobry jednak spróbowałyśmy poeksperymentować ze zmianą learning rate'u. Dla lr= 0.0008 otrzymałyśmy wynik nieznacznie lepszy od poprzedniego: 98.7%

Architektura z większą ilością warstw konwolucyjnych okazała się bardziej skuteczna od poprzedniej, osiągając wynik zbliżony do 98.7% dokładności na zbiorze testowym.

3.2.3 Architektura 3

Trzeci model bazuje na architekturze modelu drugiego, ponieważ sprawdziła się ona lepiej niż ta z modelu pierwszego. Przeprowadziłyśmy zmianę funkcji aktywacji oraz zmodyfikowałyśmy warstwy liniowe.

- Architektura ta, tak jak poprzednio, wykorzystuje cztery warstwy konwolucyjne z kolejno 32, 64, 128 i 256 kanałami wyjściowymi, rozmiarem jądra ustawionym na 5, paddingiem ustawionym na 2, jednak z funkcją aktywacji LeakyReLU.
- Drop out oraz max polling pozostały bez zmian.
- Zmniejszyłyśmy liczby neuronów w pierwszej warstwie liniowej z 1024 do 512.

Model wytrenowałyśmy na 20 epokach z optymalizatorem AdamW i lr=0.001. Uzyskałyśmy wynik dokładności na poziomie około 98.1% na zbiorze testowym, co jest gorszym wynikiem niż w przypadku modelu 2. Spróbowałyśmy zmniejszyć learning rate. Dla lr= 0.0005 otrzymałyśmy zadowalający wynik równy 99.2%.

Przypuszczamy, że architektura 3 ze zmniejszonym learning ratem jest najlepszą spośród przez nas przetestowanych, ale mogłoby się okazać, że na zmniejszonym learnig racie model 2 byłby lepszy.

3.3 Dobór i Optymalizacja Hiperparametrów

Do treningu modelu użyłyśmy optymalizatora AdamW z wbudowaną regularyzacją L2. Ustawiłyśmy parametr weight decay na 0.05, by uniknąć przeuczenia. Hiperparametry takie jak learning rate i stopa dropout zostały dobrane na podstawie eksperymentów.

ROZDZIAŁ 4

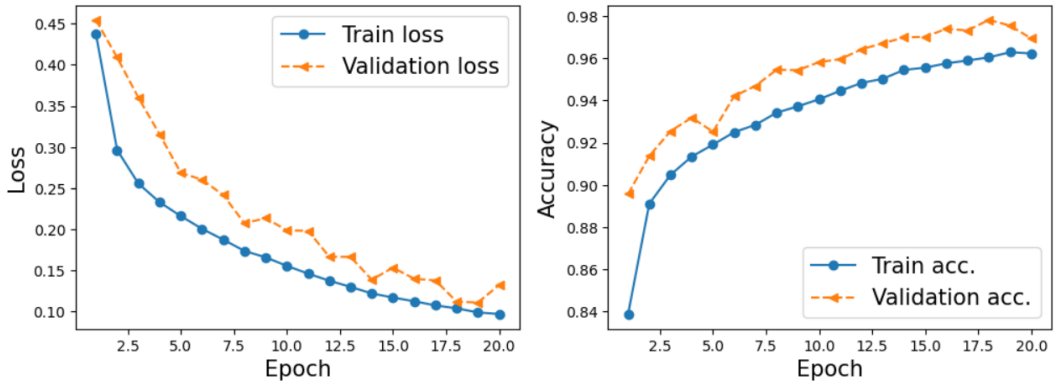
Ewaluacja

4.1 Ocena Modelu

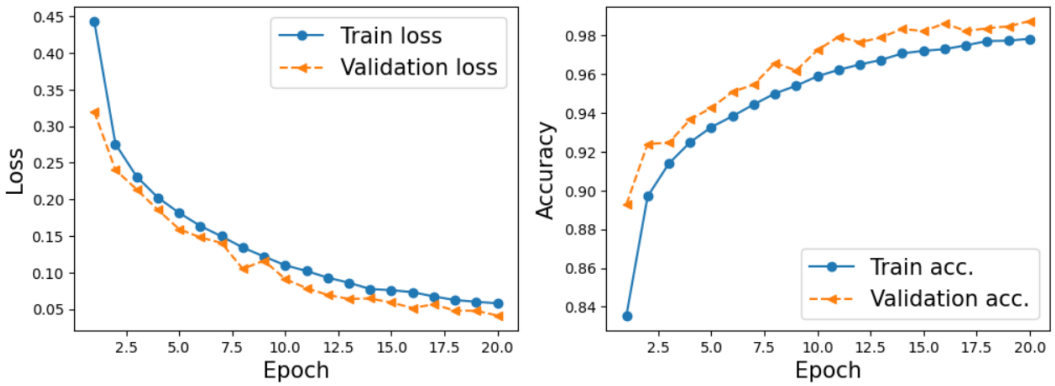
Modele oceniłyśmy na zbiorze walidacyjnym oraz testowym. Wykorzystałyśmy zaimplementowaną funkcję test_model do oceny jakości modeli. Wyniki treningu pokazały, że modele osiągają wysoką dokładność zarówno na zbiorze walidacyjnym, jak i testowym:

- Model 1, learning rate = 0.001, accuracy = 0.9629
- Model 1, learning rate = 0.0005, accuracy = 0.9694
- Model 2, learning rate = 0.001, accuracy = 0.9864
- Model 2, learning rate = 0.0008, accuracy = 0.9874
- Model 3, learning rate = 0.001, accuracy = 0.9815
- Model 3, learning rate = 0.0005, accuracy = 0.9922

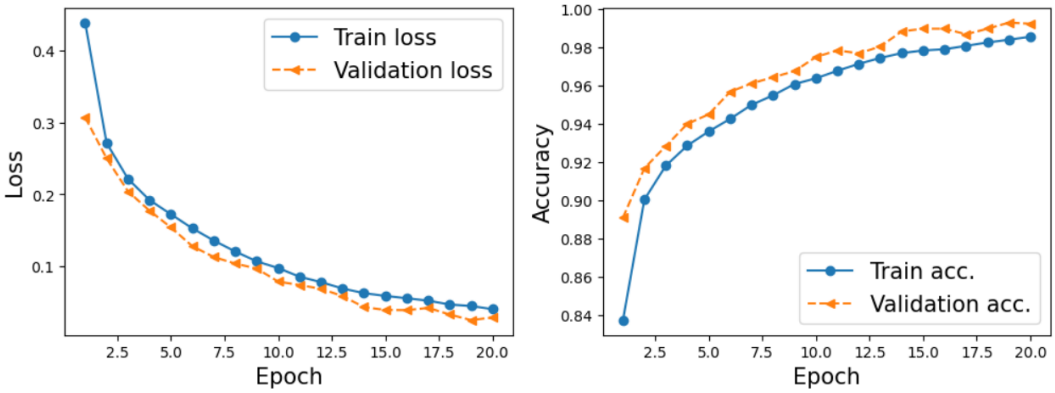
Poniżej przedstawiamy wykresy strat i dokładności dla zbiorów treningowych i walidacyjnych (zawężając się jedynie do wykresów każdego modelu z learning ratem dającym lepszy wynik):



Rysunek 4.1: Model 1, lr= 0.0005



Rysunek 4.2: Model 2, lr=0.0008



Rysunek 4.3: Model 3, lr=0.0005

ROZDZIAŁ 5

Podsumowanie

Każdy model oceniłyśmy na zbiorze walidacyjnym i testowym, mierząc dokładność klasyfikacji. Pokazałyśmy, że odpowiedni dobór architektury sieci oraz optymalizacja parametrów mogą istotnie poprawić wydajność modelu.

Model 2 osiągnął lepsze wyniki w porównaniu do modelu 1. Zaimplementowałyśmy model 3 w celu sprawdzenia, czy inna funkcja aktywacji oraz drobne zmiany w warstwie liniowej mogą poprawić wydajność modelu 2. Osiągnęłyśmy najlepsze rezultaty po wytrenowaniu modelu 3 ze współczynnikiem uczenia 0.0005. Jednak musimy pamiętać, że w porównaniu z modelem 2 wzięłyśmy nieco mniejszy learning rate, więc możemy przypuszczać, że poprzedni model na podobnym learning rate mógłby się zachowywać lepiej.

Zmiana learning rate'u była kluczowa dla poprawy wydajności modeli. Dodatkowo, augmentacja danych i regularyzacja poprzez drop out pomogły w uniknięciu przeuczenia. Wyniki wykazały stabilność i brak nadmiernego dopasowania.

Projekt wykazał, że automatyczne rozpoznawanie ubrań przy użyciu sieci konwolucyjnych jest realnym i skutecznym podejściem, które ma szerokie zastosowanie w różnych branżach.