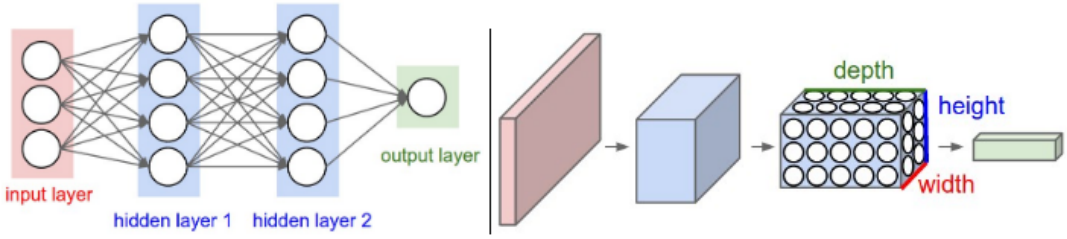
ĆWICZENIE 4
Sieć neuronowa z jedną warstwą konwolucyjną do rozpoznawania cyfr

1 Plan eksperymentów

Celem przeprowadzonych eksperymentów było zapoznanie się z operacją konwolucji oraz uczeniem sieci z **warstwą konwolucyjną** (*Convolution Layer*).

W ćwiczeniu tym skupiono się na badaniu wpływu **wielkości filtra** oraz wartości **paddingu**, czyli rozmiaru ramki złożonej z zer otaczającej obraz będący wzorcem treningowym, na otrzymywane wyniki klasyfikacji. Porównano również proces uczenia i otrzymaną skuteczności predykcji sieci MLP w stosunku do sieci z warstwą konwolucyjną.



Rysunek 1: Porównanie architektury sieci wielowarstwowej płaskiej *MLP* (po lewej) z architekturą sieci uczonej z warstwą konwolucyjną (po prawej)

Do wykonania eksperymentów rozszerzono implementowany w ramach poprzednich ćwiczeń laboratoryjnych program symulujący działanie sieci neuronowej. Dopisano fragment dotyczący warstwy konwolucyjnej, poolingu oraz warstwy odpowiadające za zmianę wymiaru podawanych danych.
Aktualizacja wag odbywała się po batchu, a po każdej epoce uczenia liczono i zapamiętywano wartość **funkcji kosztu** oraz **skuteczność** uczenia dla ciągu treningowego i walidacyjnego. Jako miarę skuteczności uczenia przyjęto liczbę poprawnych predykcji sieci na liczbę wszystkich wzorców z danego ciągu danych. Warunkiem stopu procedury uczącej była zadana z góry maksymalna liczba epok. Architektura sieci składała się z następujących warstw:

1. wejściowej (*Input*) podającej wzorce danych do sieci
2. zmieniającej wymiar wzorca danych (*Reshape*) ze spłaszczonej reprezentacji wektorowej na trójwymiarową macierz
3. konwolucji (*Conv2D*) z domyślnymi w eksperymentach hiperparametrami:
   - liczba filtrów - 8
   - krok przesunięcia (*stride*) - 1
   - metoda inicjalizacji wag - KaimingHe
   - funkcja aktywacji - ReLU

Wielkość filtra i padding był zmienny w zależności od eksperymentu.

4. maxpoolingu (*MaxPool2D*) redukującego liczbę parametrów modelu i uogólniającego wyniki pochodzące z poprzedzającej warstwy konwolucyjnej
   - wielkość filtra - 2x2
   - krok przesunięcia - 2
5. zmieniającej wzorzec z trójwymiarowej macierzy na spłaszczony wektor (*Flatten*)
6. ukrytej (*Dense*)
   - 100 w pełni połączonych neuronów
   - metoda inicjalizacji wag - KaimingHe
   - funkcja aktywacji - ReLU
7. klasyfikującej (*Dense*)
   - 10 w pełni połączonych neuronów
   - metoda inicjalizacji wag - KaimingHe
   - funkcja aktywacji - softmax

Stosowaną funkcją straty była entropia krzyżowa, która jest odpowiednia dla problemów klasyfikacji.

Domyślne parametry dla procedury uczenia:

1. liczba epok - 20
2. wielkość batcha - 64
3. optymalizator współczynnika uczenia - SGD (*Stochastic Gradient Descent*)

W każdym z eksperymentów zmieniano jeden lub kilka powyższych parametrów, reszta ustalonych wartości pozostawała niezmienna.

2 Opis aplikacji wykorzystywanej do badań

Aplikację symulującą działanie modelu sieci zaimplementowano w języku Python w środowisku PyCharm. Struktura projektu przedstawia się tak samo jak w poprzednim ćwiczeniu. Dodano jedynie kilka plików potrzebnych do pełnej implementacji architektury z warstwą konwolucyjną.

```
neural-net
├── experiments
│   ├── experiments_conv.py
│   ├── experiments_mlp.py
│   └── run_training.py
├── net
│   ├── layers
│   │   ├── activations.py
│   │   ├── conv.py
│   │   ├── dense.py
│   │   ├── flatten.py
│   │   ├── input.py
│   │   ├── layer.py
│   │   └── pooling.py
│   ├── model
│   │   ├── architectures.py
│   │   └── model.py
│   ├── batcher.py
│   ├── callbacks.py
│   ├── initializers.py
│   ├── losses.py
│   ├── metrics.py
│   ├── optimizers.py
│   └── loggers.py
├── utils.py
└── training.py
```

W pakiecie `layers` dodatkowo pojawiły się pliki:

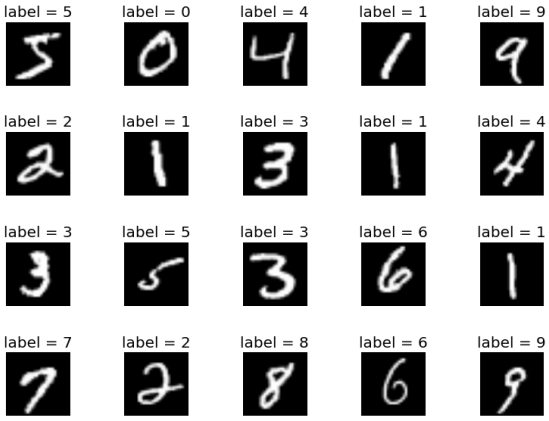
- `conv.py` - zawiera warstwę przeprowadzającą operację splotu (konwolucji). Metodę odpowiadającą za propagację wsteczną w tej warstwie zaimplementowano również z wykorzystaniem dwóch konwolucji do wyliczenia gradientów po wagach i macierzy wejściowej danych.
- `flatten.py` - zaimplementowano tu warstwy odpowiadające za zmianę wymiaru danych (z wektora dwuwymiarowego na macierz trójwymiarową oraz odwrotnie). Potrzebne do przejścia między warstwą wejściową a konwolucyjną oraz między warstwą konwolucyjną a w pełni połączoną.
- `pooling.py` - zawiera kod warstwy wykonującej sampling, która jest operacją splotu, z tym że wagi tutaj przyjmują domyślnie wartość 1. Dla wartości kroku 2 i wielkości filtra 2 zmniejsza pierwszy i drugi wymiar danych (wysokość i szerokość) trójwymiarowej macierzy o połowę.

3 Charakterystyka zbiorów danych użytych do badań

W ćwiczeniu użyto, tak jak poprzednio, zbioru danych MNIST składającego się z ręcznie pisanych cyfr od 0 do 9. Reprezentacją pojedynczej cyfry jest tu zdjęcie o wymiarach 28x28 pikseli przedstawionej jako znormalizowany spłaszczony wektor wymiaru 784x1 wraz z odpowiadającą mu etykietą klasy. Zbiór ten zawiera łącznie 70 tys. wzorców, które podzielono na:

- ciąg treningowy - 50 tys. wzorców
- ciąg walidacyjny - 10 tys. wzorców
- ciąg testowy - 10 tys. wzorców

Podział danych na trzy zbiory został wykonany raz i pozostawały one niezmienne co do zawartości w trakcie wszystkich eksperymentów w celu uniknięcia wpływu zmiany ciągów danych na uczenie sieci.



Rysunek 2: Wizualizacja danych MNIST przedstawiająca 10 klas cyfr

4 Wpływ wielkości filtra warstwy konwolucyjnej na wyniki uczenia sieci

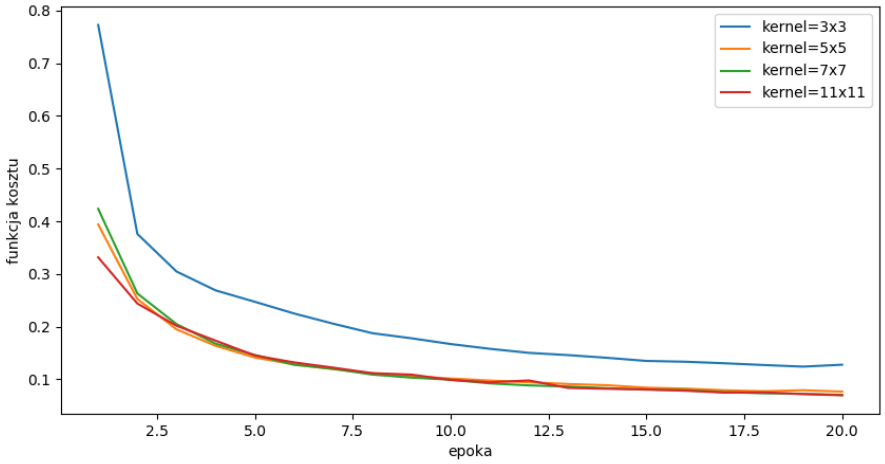
4.1 Eksperyment 1. Z zastosowaniem zerowej wartości paddingu

Założenia:

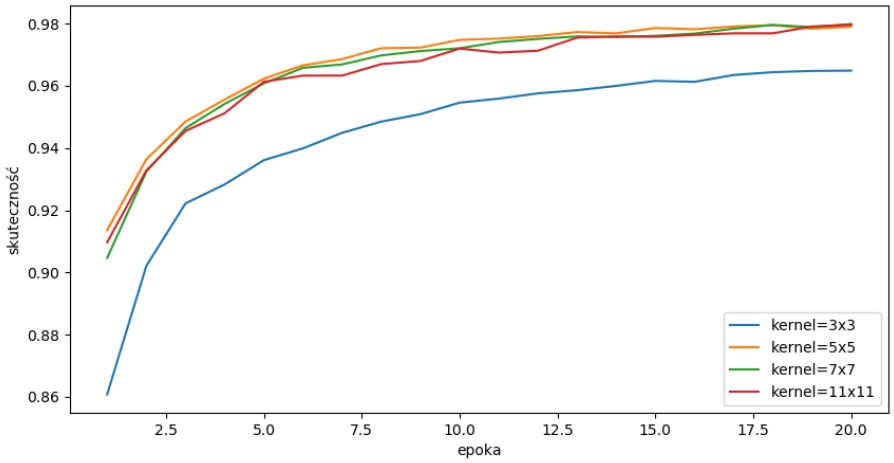
- stałe domyślne parametry modelu sieci konwolucyjnej oraz procedury uczącej wymienione w **pkt. 1**
- warunek stopu - 20 epok
- zmianom ulegała wielkość filtra warstwy konwolucyjnej z następującego zbioru {3x3, 5x5, 7x7, 11x11}
- padding - nie stosowano, więc wymiary wysokości i szerokości danych będących zdjęciami ulegał zmniejszeniu

Przebieg eksperymentu:
Zbadano wpływ wielkości filtra warstwy konwolucyjnej na wyniki klasyfikacji sieci neuronowej poprzez wykonanie procedury uczącej modelu raz dla każdej wartości wielkości filtra przy zachowaniu stałych wartości pozostałych parametrów uczenia. Zapamiętywano wartość funkcji kosztu oraz skuteczność dla ciągu walidacyjnego po każdej epoce uczenia w celach porównawczych uczenia sieci dla poszczególnych wielkości badanego parametru. Zmierzono również czas trwania uczenia w minutach dla każdego wywołania procedury uczącej.

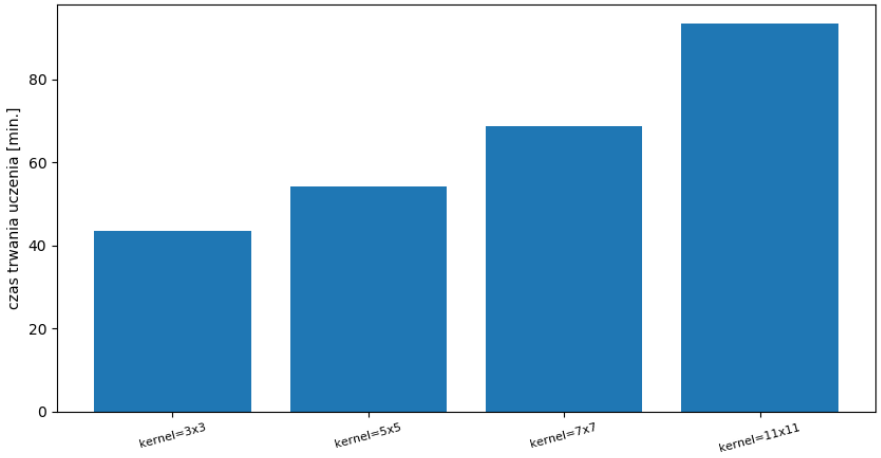
Wyniki:



Rysunek 3: Wartość funkcji straty na ciągu walidacyjnym po każdej epoce dla różnych wielkości filtra



Rysunek 4: Skuteczność na ciągu walidacyjnym po każdej epoce dla różnych wielkości filtra



Rysunek 5: Czas uczenia sieci z warstwą konwolucyjną dla różnych wielkości filtra

Tabela 1: Skuteczność na ciągu **testowym** uzyskana dla różnych wielkości filtra

ciąg testowy	
wielkość filtra	skuteczność
3x3	96.24%
5x5	97.78%
7x7	97.99%
11x11	97.95%

Komentarz:
Na podstawie wyników można stwierdzić, że użycie większych wartości wielkości filtra sprawdziło się lepiej niż najczęściej stosowana w sieciach konwolucyjnych wielkość 3x3. Filtry o wielkości 5x5 i większej uzyskały skuteczność na ciągu testowym równą ponad 97%, podczas gdy filtr 3x3 osiągnął skuteczność ok. 1 procent mniejszą. Nie jest to duża różnica, ale zauważalna dość wyraźnie na **Rysunku 4.**, gdzie jest przedstawiona skuteczność dla ciągu walidacyjnego - przez cały proces uczenia większe filtry dawały lepszą skuteczność niż najmniejsza zastosowana jego wielkość. Jest to spowodowane najpewniej charakterystyką danych. Cyfry są krzywymi liniami na obrazie, które są rzadko przestrzennie rozłożone. Większy filtr zobaczy więc lokalnie większą część fragmentu cyfry i najwidoczniej wtedy sieci łatwiej jest rozpoznać jej klasę.
Na podstawie wyników czasu uczenia sieci z warstwą konwolucyjną przedstawione na **Rysunku 5.** widać, że operacja splotu jest obciążająca dla CPU, na którym były wykonywane obliczenia macierzowe. Wymaga ona wiele operacji pośrednich i ciągłych zmian wymiarów przestrzennych macierzy, co powoduje, że proces liczenia w tej warstwie zajmuje dużo czasu (rzędu kilkudziesięciu minut, a nawet godzin). Im większy użyty filtr, tym czas uczenia sieci wydłużał się.

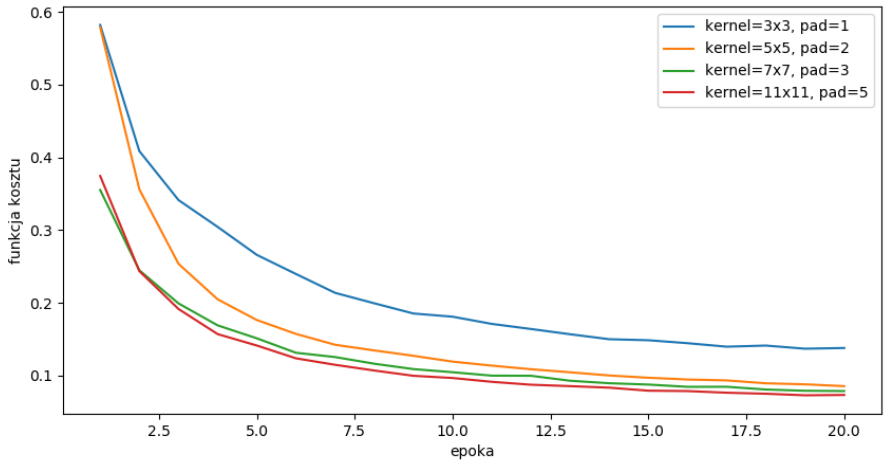
4.2 Eksperyment 2. Z dobraniem wartości paddingu, aby zachować początkowy wymiar wysokości i szerokości danych

Założenia:

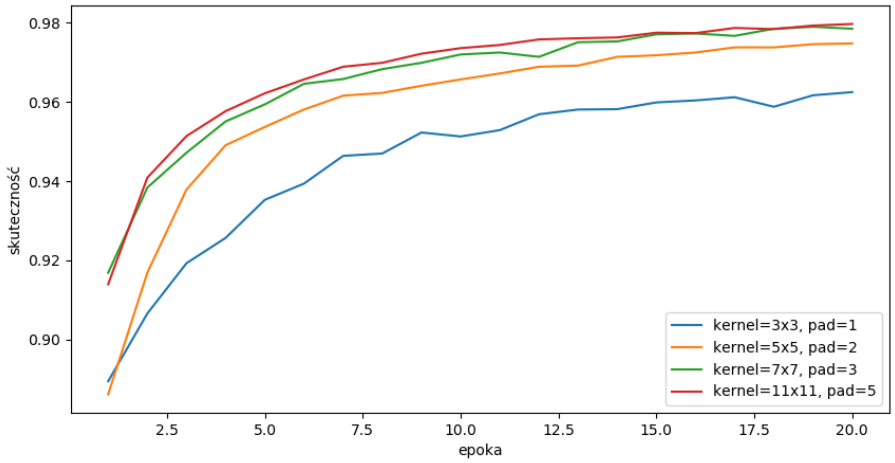
- stałe domyślne parametry modelu sieci konwolucyjnej oraz procedury uczącej wymienione w **pkt. 1**
- warunek stopu - 20 epok
- zmianom ulegała wielkość filtra warstwy konwolucyjnej z następującego zbioru {3x3, 5x5, 7x7, 11x11}
- padding - dobrano tak, aby wysokość i szerokość danych nie zmieniał się po wyjściu z warstwy konwolucji; dla wybranych wielkości filtra wartości przedstawiały się następująco: 3x3 - 1, 5x5 - 2, 7x7 - 3, 11x11 - 5

Przebieg eksperymentu:
Zbadano wpływ wielkości filtra warstwy konwolucyjnej poprzez uruchomienie procedury uczącej modelu sieci raz dla każdej wartości wielkości filtra, zachowując przy tym stałe pozostałe parametry uczenia. Dla każdej wielkości filtra dobrano wartość paddingu, czyli wielkość ramki złożonej z zer otaczającej wzorzec treningowy, odpowiednio tak, aby rozmiar przestrzenny danych wejściowych nie uległ zmianie. Zapamiętywano wartość funkcji kosztu oraz skuteczność dla ciągu walidacyjnego po każdej epoce uczenia, zmierzono też czas trwania uczenia w minutach dla każdego wywołania procedury uczącej.

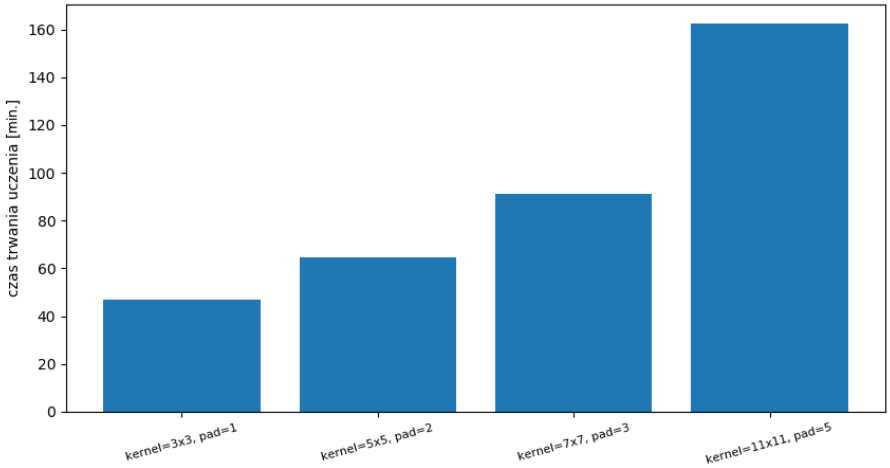
Wyniki:



Rysunek 6: Wartość funkcji straty na ciągu walidacyjnym po każdej epoce dla różnych wielkości filtra



Rysunek 7: Skuteczność na ciągu walidacyjnym po każdej epoce dla różnych wielkości filtra



Rysunek 8: Czas uczenia sieci z warstwą konwolucyjną dla różnych wielkości filtra

Tabela 2: Skuteczność na ciągu **testowym** uzyskana dla różnych wielkości filtra z zastosowaniem paddingu

ciąg testowy	
wielkość filtra	skuteczność
3x3	95.93%
5x5	97.69%
7x7	97.75%
11x11	98.01%

Komentarz:
Uzyskano wyniki podobne do poprzedniego eksperymentu - większe filtry okazały się lepsze niż użycie filtra o wielkości 3x3. Można jednak zwrócić uwagę na to, że linie przebiegu funkcji straty i skuteczności dla większych filtrów mniej się na siebie nakładają w porównaniu do **Rysunku 4.** i **5.** z poprzedniego eksperymentu. Najlepszą skuteczność na ciągu testowym równą 98% uzyskał największy zastosowany filtr 11x11.
W przypadku stosowania paddingu w celu zachowania wymiarów przestrzennych danych wejściowych nie tracimy na wyjściu informacji z brzegów wzorca będącego zdjęciem cyfry. Przy użytym dużym filtrze mogło to mieć większe znaczenie, bo redukcja wysokości i szerokości obrazu na wyjściu jest dla niego duża i dalsze warstwy tracą więcej informacji o krańcowych elementach podawanego wzorca. Biorąc jednak pod uwagę charakterystykę danych i to, że na krańcach obrazów nie ma linii prezentujących cyfrę, wyniki z tego eksperymentu dały porównywalną (ale nie lepszą) skuteczność dla poszczególnych użytych wielkości filtrów w stosunku do pierwszego eksperymentu.
Analizując wyniki czasu trwania uczenia sieci z zastosowanym paddingiem można powiedzieć, że trwa ono dłużej niż bez użycia ramki z zer. Jest to zrozumiałe, gdyż otoczenie macierzy wzorca wartościami zerowymi wzdłuż odpowiednich wymiarów wymaga dodatkowych operacji wykonywanych przez CPU. Wymiar podawanego dalej wzorca jest też większy, więc sieć w późniejszych swoich warstwach operuje na macierzach o większej wartości poszczególnych jej wymiarów.

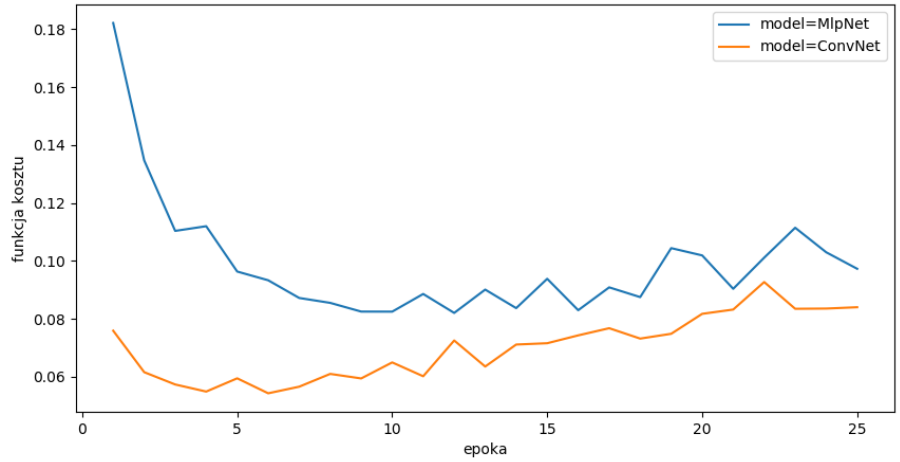
5 Eksperyment 3. Porównanie uczenia sieci z warstwą konwolucyjną do sieci MLP

Założenia:

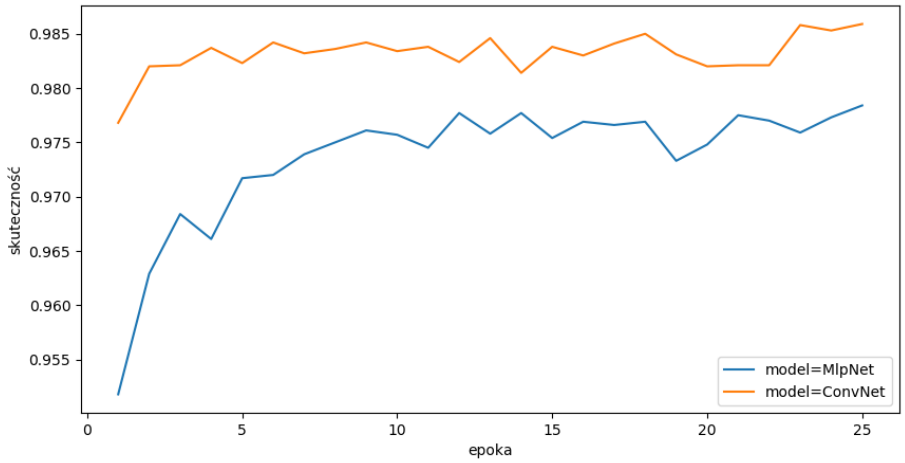
- stałe domyślne parametry modelu sieci konwolucyjnej oraz procedury uczącej wymienione w **pkt. 1**
- wielkość filtra - 7x7
- padding - 3 (wysokość i szerokość danych została zachowana na wyjściu warstwy konwolucyjnej)
- warunek stopu - 25 epok
- optymalizator współczynnika uczenia - Adam
- model sieci MLP składający się z warstwy wejściowej, ukrytej (100 neuronów) oraz wyjściowej (10 neuronów)

Przebieg eksperymentu:
W tym eksperymencie porównano przebieg uczenia oraz otrzymane wyniki klasyfikacji dwóch modeli sieci - MLP oraz sieci z jedna warstwą konwolucyjną. Dla każdego modelu uruchomiono procedurę uczącą raz aż do warunku zatrzymania, czyli osiągnięcia 25. epok uczenia. W trakcie trwania uczenia zapamiętywano wartości funkcji straty oraz skuteczność na ciągu walidacyjnym po każdej epoce uczenia, które następnie przedstawiono na wspólnych wykresach w celach porównawczych obu modeli. Zbadano także czas trwania uczenia danego modelu w minutach i zobrazowano wyniki na wykresie słupkowym.

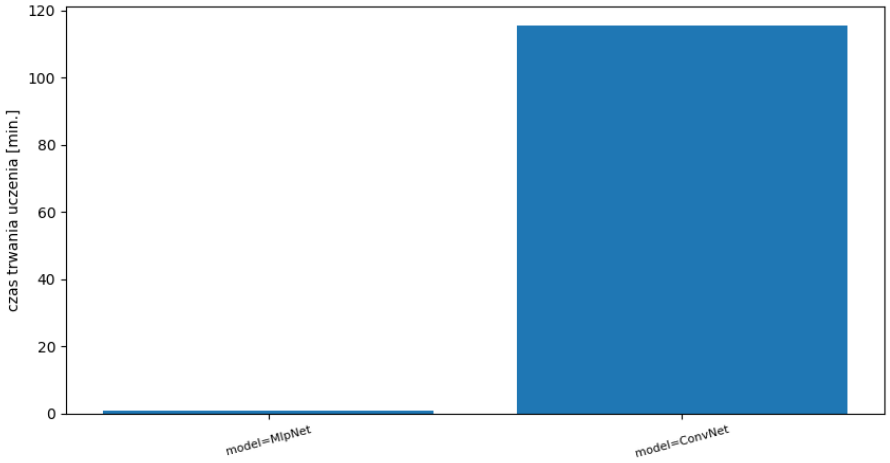
Wyniki:



Rysunek 9: Wartość funkcji kosztu na ciągu walidacyjnym liczona po każdej epoce dla różnych modeli sieci



Rysunek 10: Skuteczność na ciągu walidacyjnym liczona po każdej epoce dla różnych modeli sieci



Rysunek 11: Czas trwania uczenia dla różnych modeli sieci

Tabela 3: Skuteczność na ciągu **testowym** uzyskana dla modelu sieci MLP i sieci z warstwą konwolucyjną

ciąg testowy	
model	skuteczność
MlpNet	97.74%
ConvNet	98.55%

Komentarz:
Z analizy wyników można zobaczyć, że oba modele uzyskują wysoką skuteczność dla danych MNIST. Sieć z jedną warstwą konwolucyjną uzyskała ok. 1 procent większą skuteczność na ciągu testowym niż sieć MLP. Warstwa konwolucji dzięki lokalnym połączeniom neuronów pozwala na rozpoznawanie na tym obszarze pewnych konkretnych cech obrazu, co zwykle daje wyższą skuteczność predykcji klas niż w przypadku stosowania tylko pełnych połączeń (jak w MLP), jeśli nasze dane treningowe mają postać zdjęć. Użyte w zadaniu dane MNIST nie są skomplikowanym zbiorem danych, więc nie spodziewano się dużej różnicy w skuteczności i przebiegu uczenia obu modeli. Można jednak zwrócić uwagę na tempo spadku funkcji kosztu na **Rysunku 9**. Dla sieci *ConvNet* już po pierwszej epoce funkcja straty osiągnęła niską wartość, wokół której później oscylowała w dalszych epokach. Natomiast dla sieci *MlpNet* spadek funkcji kosztu w początkowych epokach zachodził trochę łagodniej, a ok. 5 epoki osiągnęła ona wartość, która potem nie ulegała dużym zmianom. Przebieg funkcji straty ma odzwierciedlenie w uzyskanej przez modele skuteczności na **Rysunku 10**. Sieć konwolucyjna błyskawicznie osiągnęła wysoką skuteczność, której poziom potem nieznacznie powoli wzrastał.
Patrząc na czas trwania uczenia obu modeli, sieć z warstwą konwolucyjną wymaga o wiele większych zasobów obliczeniowych niż sieć MLP. Uczenie podanej wcześniej architektury sieci konwolucyjnej zajęło prawie 2 godziny, podczas gdy sieć składająca się jedynie z warstw w pełni połączonych osiągnęła 25. epok uczenia zaledwie po paru minutach. Obliczenia wykonywane były niestety na CPU, a nie na GPU, która znacząco przyśpieszyłaby skomplikowane i liczne operacje macierzowe wykonywane w trakcie operacji splotu w warstwie konwolucyjnej.

6 Podsumowanie wyników badań eksperymentalnych

1. Dla prostych danych MNIST, gdzie cyfry są rzadko przestrzennie rozmieszczone na obrazie wzorca uczącego zastosowanie większych wielkości filtrów w warstwie konwolucyjnej pomaga w uzyskaniu minimalnie lepszej skuteczności niż dla filtra 3x3. Jednak zwykle praktykuje się używanie małych filtrów w procesie uczenia sieci konwolucyjnej.
2. Zastosowanie paddingu w celu zachowania przestrzennych wymiarów macierzy wyjściowej z warstwy konwolucyjnej jest częstym zabiegiem stosowanym w projektowaniu architektury sieci konwolucyjnych. Daje to zwykle lepsze wyniki, gdyż nie tracimy istotnych danych z krańcowych brzegów obrazu. Dla danych MNIST nie zauważono szczególnej poprawy skuteczności predykcji dla tego zabiegu ze względu na charakterystykę wzorców uczących.
3. Użycie sieci z warstwami konwolucji, gdy podawane dane są obrazami, poprawia jakość uczenia sieci i pozwala na uzyskanie wyższej skuteczności w porównaniu do sieci wykorzystującej jedynie warstwy w pełni połączone. Sieć konwolucyjna pozwala na zakodowanie pewnych właściwości w swojej architekturze (na mapie cech) oraz umożliwia współdzielenie parametrów, co zmniejsza znacząco ich liczbę.
4. Wykonywanie operacji splotu jest kosztowne obliczeniowo i wymaga wielu przekształceń na macierzach, przez co uczenie sieci z warstwą konwolucji trwa dłużej niż trenowanie sieci MLP.