WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ CYBERNETYKI

Sprawozdanie

Zaawansowane metody uczenia maszynowego

Sieci konwolucyjne

Autor:

Karol Baranowski

Prowadzący:

mgr inż. Przemysław Czuba

Spis treści

Zadanie

Implementacja sieci LaNet-5 2 . Zaprojektowana do klasyfikacji pisma odręcznego. Posiada dwie warstwy konwolucyjne, każda posiada warstwę subsampling (pooling).

1. Opis implementacji modelu LaNet

2. Analiza porównawcza modelu MLP z poprzednich zajęć oraz modelu LaNet:

    a. Jakie są główne różnice, który sprawuje się lepiej dla jakiej ilości danych

    b. Dla jakiej konfiguracji sieci MLP wyniki są zbliżone do CNN? Ile czasu trwa uczenie jednej oraz drugiej sieci aby osiągnąć podobne wyniki?

Zadanie I Opis implementacji modelu LaNet

```
"""
  Model LeNet:
  - opisać użyte funkcje oraz dodać komentarze nad każdą linijką
"""
class LeNet(nn.Module):
    # Uruchomienie metody inicjalizującej(konstruktora), wykonującej się przy powstaniu obiektu
    def __init__(self, output_dim):
        # Uruchomienie konstruktora __init__() obiektu narzędnego tzn. nn.Module po, którym dziedziczy
        # LeNet
        super().__init__()
        # Konwolucja map cech obiektu wejściowego z filtrem
        # in_channels - Liczba kanałów w obrazie wejściowym
        # out_channels - Liczba kanałów wyprodukowanych przez konwolucję
        # kernel_size - Rozmiar filtra konwolucji, określa długość i wysokość maski filtra
        self.conv1 = nn.Conv2d(in_channels=1,
                               out_channels=6,
                               kernel_size=5)
        # Druga konwolucja map cech obiektu wejściowego (tzn. wyjściowego z conv1) z filtrem
        self.conv2 = nn.Conv2d(in_channels=6,
                               out_channels=16,
                               kernel_size=5)
        # nn.Linear(x_size, y_size) jest funkcją realizującą transformację liniową do przychodzących
        # danych:
        # y = x * W ^ T + b. W parametrach przyjmuje wielkości każdej próbki wejścia i wyjścia czyli w
        # tym przypadku ilość neuronów na warstwie.
        # Są trzy warstwy (fc - fully connected - traktuje wejście jako jednowymiarową listę)
        # Do pierwszej warstwy przekazywane są 256 cechy wejściowe,120 to liczba cech wyjściowych
        self.fc_1 = nn.Linear(16 * 4 * 4, 120)
        # Do drugiej warstwy przekazywane są 120 cechy wejściowe (cechy wyjściowe fc_1),84 to liczba
        # cech
        # wyjściowych
        self.fc_2 = nn.Linear(120, 84)
        # W trzeciej warstwie 84 cechy wejściowe, output_dim to liczba zdefiniowanych klas wyjściowych
        # (10)
        self.fc_3 = nn.Linear(84, output_dim)


    # Propagacja do przodu.Definiuje obliczneia, operacje, funkcje, któ©e mają sie  wykonać
    # przy każdym wywołaniu.
    def forward(self, x):
        # Pierwsza konwolucja na warstwie wejściowej
        x = self.conv1(x)
        # Pooling po oknie 2x2. Przechodzenie po fragmencie i wybraniu wartości maksymalnej.
        x = F.max_pool2d(x, kernel_size=2)
        # Zastosowanie na tę warstwę funkcji aktywacji ReLu
        x = F.relu(x)
        # Druga konwolucja (konwolucje i transformacje zdefiniowane w __init__() - tu ich wywołanie)
        x = self.conv2(x)
        # Ponowny pooling po oknie 2x2. Przechodzenie po fragmencie i wybraniu wartości
        # maksymalnej.
        x = F.max_pool2d(x, kernel_size=2)
        # Zastosowanie na tę warstwę funkcji aktywacji ReLu
        x = F.relu(x)
        # Zmiana kształtu tensora aby otrzymać płaski w pełni połączoną warstwę. Gdy chcemy aby w x
        # była konkretna
        # liczba (x.shape[0]) wierszy, ale nie jesteśmy pewni ilości kolumn można wpisać -1. Dzięki temu
        # tensor
        # wyliczy odpowiednią liczbę kolumn dla zadanej liczby wierszy.
        x = x.view(x.shape[0], -1)
        h = x
        # Wyliczenie pierwszej w pełni połączonej warstwy
        x = self.fc_1(x)
        # Zastosowanie na tę warstwę funkcji aktywacji ReLu
        x = F.relu(x)
        # Wyliczenie drugiej w pełni połączonej warstwy
        x = self.fc_2(x)
        # Zastosowanie na tę warstwę funkcji aktywacji ReLu
```

4

```
        x = F.relu(x)
        # Zastosowanie na tę warstwę funkcji aktywacji ReLu
        x = self.fc_3(x)
        # Zwrócenie warstwy po zaaplikowaniu na niej powyższych przekształceń jako warstwę
        # wyjściową
        return x, h
```

Zadanie II Analiza porównawcza modelu MLP z poprzednich zajęć oraz modelu LaNet

a. Jakie są główne różnice, który sprawuje się lepiej dla jakiej ilości danych?

Główne różnice:

LaNet jest siecią CNN, więc w odróżnieniu ud modelu MLP z poprzednich zajęć, są zaimplementowane tutaj konwulacje czyli aplikowania wag z filtra w celu ekstrakcji lokalnych cech. W związku z tym w kodzie znajdują się dodatkowe definicje filtrów. Każdy neuron warstwy ukrytej wylicza wartość konwolucji oraz stosuje funkcję aktywacji (ReLU), gdzie w modelu MLP przeprowadzana była tylko funkcja aktywacji. Gdy pracuje się na sieci CNN to stosuje się również pooling, czyli zmniejszania obrazów zachowując ich właściwości, czego również nie było w MLP. Poprzez implementację tych mechanizmów, obliczenia są bardziej wymagające i trwają dłużej. Również dla MLP wystarczył obraz jako wektor 1D, natomiast do liczenia konwolucji potrzebna jest przestrzenność obrazu, stąd w kodzie są dodatkowe konwersję widoków tensorów poprzez użycie funkcji tensor.view (odpowiednik w numpy to reshape).

Który model sprawuje się lepiej dla jakiej ilości danych?

Próba 1: Domyślne wartości LeNet

Domyślne wartości w LeNet to 3 warstwy: 256 neuronów jako warstwa wejściowa

120 neuronów w pierwszej warstwie ukrytej, 84 w drugiej warstwie ukrytej. W MLP ustawiono takie same liczby neuronów poza warstwą wejściową, gdyż w MLP przyjmowane są obrazki 28x28 = 784, natomiast w LeNet 16x4x4=256. Funkcje aktywacji ustawiono jako ReLu.

Dla tak ustawionych sieci otrzymano wyniki:

MLP:

```
The model has 105,214 trainable parameters
Epoch:  1 | Epoch Time: 0m 22s
        Train Loss: 0.473 | Train Acc: 85.38%
         Val. Loss: 0.206 |  Val. Acc: 93.67%
Epoch:  2 | Epoch Time: 0m 18s
        Train Loss: 0.200 | Train Acc: 93.87%
         Val. Loss: 0.125 |  Val. Acc: 96.19%
Epoch:  3 | Epoch Time: 0m 18s
        Train Loss: 0.157 | Train Acc: 95.23%
         Val. Loss: 0.124 |  Val. Acc: 95.86%
Epoch:  4 | Epoch Time: 0m 20s
        Train Loss: 0.136 | Train Acc: 95.81%
         Val. Loss: 0.108 |  Val. Acc: 96.68%
Epoch:  5 | Epoch Time: 0m 21s
        Train Loss: 0.123 | Train Acc: 96.11%
         Val. Loss: 0.090 |  Val. Acc: 97.22%
Test Loss: 0.078 | Test Acc: 97.46%
```

LeNet:

```
The model has 44,426 trainable parameters
Epoch: 01 | Epoch Time: 0m 22s
        Train Loss: 0.416 | Train Acc: 86.82%
         Val. Loss: 0.136 |  Val. Acc: 95.86%
Epoch: 02 | Epoch Time: 0m 23s
        Train Loss: 0.145 | Train Acc: 95.47%
         Val. Loss: 0.097 |  Val. Acc: 97.29%
Epoch: 03 | Epoch Time: 0m 23s
        Train Loss: 0.104 | Train Acc: 96.77%
         Val. Loss: 0.068 |  Val. Acc: 97.87%
Epoch: 04 | Epoch Time: 0m 24s
        Train Loss: 0.085 | Train Acc: 97.31%
         Val. Loss: 0.064 |  Val. Acc: 98.09%
Epoch: 05 | Epoch Time: 0m 24s
        Train Loss: 0.072 | Train Acc: 97.73%
         Val. Loss: 0.058 |  Val. Acc: 98.27%
Test Loss: 0.041 | Test Acc: 98.74%
```

MLP ma ponad dwukrotną liczbę trenowalnych parametrów. To dlatego, że warstwa wejściowa zawiera 784 neurony, a nie 256 jak w LeNet. Mimo to, już od pierwszej epoki LeNet dokładność lepiej. Wpierwszej epoce MLP policzył ją na 93.67%, natomiast LeNet 95.86%. Dla danych testowych po pięciu epokach szkolenia, dokładność obu sieci wzrosła. MLP do 97.46%, LeNet do 98.74%. Różnica wynosi ~ 1.3%. Mimo, iż LeNet ma prawie 2.5 razy parametrów mniej to następne epoki trwają dłużej niż dla MLP. To przez wymagające operacji konwulacji i poolingu.

Próba 2: Mała liczba neuronów

Dla dwóch sieci ustawiono 2 warstwy ukryte po 10 i 5 neuronów.

MLP:

```
The model has 7,965 trainable parameters
Epoch:  1 | Epoch Time: 0m 21s
        Train Loss: 1.386 | Train Acc: 52.09%
         Val. Loss: 0.784 |  Val. Acc: 75.37%
Epoch:  2 | Epoch Time: 0m 22s
        Train Loss: 0.922 | Train Acc: 70.73%
         Val. Loss: 0.614 |  Val. Acc: 82.05%
Epoch:  3 | Epoch Time: 0m 24s
        Train Loss: 0.765 | Train Acc: 76.14%
         Val. Loss: 0.537 |  Val. Acc: 83.74%
Epoch:  4 | Epoch Time: 0m 25s
        Train Loss: 0.708 | Train Acc: 78.15%
         Val. Loss: 0.507 |  Val. Acc: 84.86%
Epoch:  5 | Epoch Time: 0m 21s
        Train Loss: 0.673 | Train Acc: 79.22%
         Val. Loss: 0.472 |  Val. Acc: 86.50%
Test Loss: 0.463 | Test Acc: 86.84%
```

LeNet:

```
The model has 5,257 trainable parameters
Epoch: 01 | Epoch Time: 0m 20s
        Train Loss: 0.909 | Train Acc: 69.51%
         Val. Loss: 0.286 |  Val. Acc: 91.79%
Epoch: 02 | Epoch Time: 0m 21s
        Train Loss: 0.321 | Train Acc: 90.47%
         Val. Loss: 0.203 |  Val. Acc: 94.35%
Epoch: 03 | Epoch Time: 0m 23s
        Train Loss: 0.242 | Train Acc: 92.91%
         Val. Loss: 0.150 |  Val. Acc: 95.55%
Epoch: 04 | Epoch Time: 0m 22s
        Train Loss: 0.203 | Train Acc: 94.12%
         Val. Loss: 0.129 |  Val. Acc: 96.22%
Epoch: 05 | Epoch Time: 0m 23s
        Train Loss: 0.175 | Train Acc: 94.92%
         Val. Loss: 0.104 |  Val. Acc: 96.98%
Test Loss: 0.085 | Test Acc: 97.41%
```

Wyraźnie widać, że MLP dla małych ilości neuronów w warstwach znacznie słabiej wylicza dokładność. Po pięciu epokach wynosi ona 86.84%, a dla LeNet 97.41%. Różnica wynosi prawie 11%. Nasuwa się również wniosek, że sieć LeNet wraz z ilością epok dla małych wartości neuronów i tak bardzo poprawnie rozpoznaje cyfry. Dla porównania z wynikiem z próby 1, dokładność różni się tylko o ~1.3%, mimo że neuronów w pierwszej warstwie ukrytej było o 110 mniej, a w drugiej o 79 mniej. Czasy obliczania zostały jednak podobne.

Próba 3: Duża liczba neuronów

Dla dwóch sieci ustawiono 2 warstwy ukryte po 500 i 250 neuronów.

MLP:

```
The model has 520,260 trainable parameters
Epoch:  1 | Epoch Time: 0m 19s
        Train Loss: 0.339 | Train Acc: 89.38%
         Val. Loss: 0.147 |  Val. Acc: 95.39%
Epoch:  2 | Epoch Time: 0m 26s
        Train Loss: 0.154 | Train Acc: 95.27%
         Val. Loss: 0.102 |  Val. Acc: 96.68%
Epoch:  3 | Epoch Time: 0m 26s
        Train Loss: 0.122 | Train Acc: 96.18%
         Val. Loss: 0.098 |  Val. Acc: 97.16%
Epoch:  4 | Epoch Time: 0m 23s
        Train Loss: 0.107 | Train Acc: 96.68%
         Val. Loss: 0.092 |  Val. Acc: 97.29%
Epoch:  5 | Epoch Time: 0m 28s
        Train Loss: 0.099 | Train Acc: 96.90%
         Val. Loss: 0.089 |  Val. Acc: 97.41%
Test Loss: 0.076 | Test Acc: 97.64%
```

LeNet:

```
The model has 258,832 trainable parameters
Epoch: 01 | Epoch Time: 0m 23s
        Train Loss: 0.326 | Train Acc: 89.65%
         Val. Loss: 0.119 |  Val. Acc: 96.32%
Epoch: 02 | Epoch Time: 0m 23s
        Train Loss: 0.104 | Train Acc: 96.75%
         Val. Loss: 0.071 |  Val. Acc: 97.98%
Epoch: 03 | Epoch Time: 0m 25s
        Train Loss: 0.080 | Train Acc: 97.47%
         Val. Loss: 0.055 |  Val. Acc: 98.27%
Epoch: 04 | Epoch Time: 0m 24s
        Train Loss: 0.069 | Train Acc: 97.83%
         Val. Loss: 0.058 |  Val. Acc: 98.28%
Epoch: 05 | Epoch Time: 0m 25s
        Train Loss: 0.058 | Train Acc: 98.18%
         Val. Loss: 0.047 |  Val. Acc: 98.68%
Test Loss: 0.033 | Test Acc: 98.93%
```

Dla tak zwiększonej liczby neuronów widać poprawę dla LeNet, które dochodzi do prawie 99% dokładności po 5 epokach. Dla MLP jest poprawa 0.26% od danych z próby 1. Sugeruje to, że liczba neuronów wpływa korzystnie dla sieci LeNet, natomiast wpływa bardzo nieznacznie na poprawę sieci MLP. Czas w obu sieciach rośnie o 2 sekundy na epokę w stosunku do czasu z próby 1.

Próba 2: Najlepsze wartości MLP

W poprzednim ćwiczeniu najlepszą wartość MLP osiągnięto dla konfiguracji z 3 warstwami ukrytymi odpowiednio: 500 neuronów, 100 neuronów, 50 neuronów. Dokładność wtedy wyniosła 98.4% po 10 epokach:

```
Epoch: 02 | Epoch Time: 0m 24s
        Train Loss: 0.189 | Train Acc: 94.13%
         Val. Loss: 0.135 |  Val. Acc: 95.64%
Test Loss: 0.109 | Test Acc: 96.40%
```

Ustawiono więc te parametry w dwóch sieciach w celu sprawdzenia czy MLP może lepiej policzyć dokładność niż LeNet.

MLP:

```
The model has 448,160 trainable parameters
Epoch:  1 | Epoch Time: 0m 25s
        Train Loss: 0.405 | Train Acc: 87.32%
         Val. Loss: 0.184 |  Val. Acc: 94.04%
Epoch:  2 | Epoch Time: 0m 30s
        Train Loss: 0.163 | Train Acc: 94.91%
         Val. Loss: 0.127 |  Val. Acc: 96.13%
Epoch:  3 | Epoch Time: 0m 28s
        Train Loss: 0.129 | Train Acc: 95.94%
         Val. Loss: 0.105 |  Val. Acc: 96.56%
Epoch:  4 | Epoch Time: 0m 29s
        Train Loss: 0.113 | Train Acc: 96.49%
         Val. Loss: 0.091 |  Val. Acc: 97.16%
Epoch:  5 | Epoch Time: 0m 29s
        Train Loss: 0.102 | Train Acc: 96.78%
         Val. Loss: 0.081 |  Val. Acc: 97.53%
Epoch:  6 | Epoch Time: 0m 30s
        Train Loss: 0.094 | Train Acc: 97.08%
         Val. Loss: 0.078 |  Val. Acc: 97.77%
Epoch:  7 | Epoch Time: 0m 31s
        Train Loss: 0.084 | Train Acc: 97.29%
         Val. Loss: 0.073 |  Val. Acc: 97.67%
Epoch:  8 | Epoch Time: 0m 26s
        Train Loss: 0.084 | Train Acc: 97.41%
         Val. Loss: 0.064 |  Val. Acc: 98.02%
Epoch:  9 | Epoch Time: 0m 30s
        Train Loss: 0.077 | Train Acc: 97.57%
         Val. Loss: 0.070 |  Val. Acc: 97.88%
Epoch: 10 | Epoch Time: 0m 29s
        Train Loss: 0.076 | Train Acc: 97.61%
         Val. Loss: 0.059 |  Val. Acc: 98.18%
Test Loss: 0.056 | Test Acc: 98.26%
```

LeNet:

```
The model has 186,732 trainable parameters
Epoch: 01 | Epoch Time: 0m 27s
        Train Loss: 0.427 | Train Acc: 86.16%
         Val. Loss: 0.128 |  Val. Acc: 95.96%
Epoch: 02 | Epoch Time: 0m 28s
        Train Loss: 0.129 | Train Acc: 95.93%
         Val. Loss: 0.095 |  Val. Acc: 97.06%
Epoch: 03 | Epoch Time: 0m 28s
        Train Loss: 0.093 | Train Acc: 97.03%
         Val. Loss: 0.071 |  Val. Acc: 97.89%
Epoch: 04 | Epoch Time: 0m 28s
        Train Loss: 0.076 | Train Acc: 97.65%
         Val. Loss: 0.058 |  Val. Acc: 98.40%
Epoch: 05 | Epoch Time: 0m 27s
        Train Loss: 0.068 | Train Acc: 97.92%
         Val. Loss: 0.049 |  Val. Acc: 98.80%
Epoch: 06 | Epoch Time: 0m 27s
        Train Loss: 0.059 | Train Acc: 98.22%
         Val. Loss: 0.047 |  Val. Acc: 98.76%
Epoch: 07 | Epoch Time: 0m 27s
        Train Loss: 0.055 | Train Acc: 98.33%
         Val. Loss: 0.044 |  Val. Acc: 98.93%
Epoch: 08 | Epoch Time: 0m 27s
        Train Loss: 0.052 | Train Acc: 98.46%
         Val. Loss: 0.041 |  Val. Acc: 98.91%
Epoch: 09 | Epoch Time: 0m 29s
        Train Loss: 0.047 | Train Acc: 98.58%
         Val. Loss: 0.044 |  Val. Acc: 98.74%
Epoch: 10 | Epoch Time: 0m 27s
        Train Loss: 0.044 | Train Acc: 98.66%
         Val. Loss: 0.044 |  Val. Acc: 98.78%
Test Loss: 0.033 | Test Acc: 98.92%
```

W typ przypadku MLP osiągnęło po 10 epokach dokładność 98.26 %, natomiast LeNet 98.92%. Widać, że zwiększenie liczby warstw nie poprawia dokładności LeNet, a wręcz ją pogarsza (wynik jest taki sam jak dla dwóch warstw z 500 i 250 neuronami po pięciu epokach).

b. Dla jakiej konfiguracji sieci MLP wyniki są zbliżone do CNN? Ile czasu trwa uczenie jednej oraz drugiej sieci aby osiągnąć podobne wyniki?

Wyniki zbliżone są najbardziej dla prób 1 i 4 z powyższego podpunktu czyli dla domyślnych wartości LeNet oraz najlepszych wartości MLP wyznaczonych eksperymentalnie na poprzednich laboratoriach.

Dla próby pierwszej wynik jest zbliżony po 5 epokach uczenia MLP (97.22%) oraz 2 epokach CNN (97.29).

Natomiast najbardziej zbliżony wynik po takiej samej ilości iteracji (10 epok) wyznaczono dla parametrów próby 4, czyli 3 warstwy ukryte: 500, 100 i 50 neuronów. Dokładność MLP wyniosła 98.26 %, natomiast LeNet 98.92%, więc różnica to 0.66% (w najlepszym osiągniętym wyniku na poprzednim laboratorium 98.4% różnica to 0.46%). Sieć MLP daje wynik zbliżone dla większej ilości warstw niż dwie. Uczenie w tym przypadku trwa podobną ilość czasu (ok. 30 sekund na epokę), natomiast MLP ma ponad 2 razy więcej trenowalnych parametrów. Znaczy to, że uczenie LeNet jest wolniejsze, ale przez to, że wejściowe obrazy mają 256 neuronów, a nie 784 wykonuje się w tym przypadku podobną ilość czasu. Jeżeli podać by jej na wejściu 784 neurony jak w MLP z pewnością nauka byłaby dłuższa.

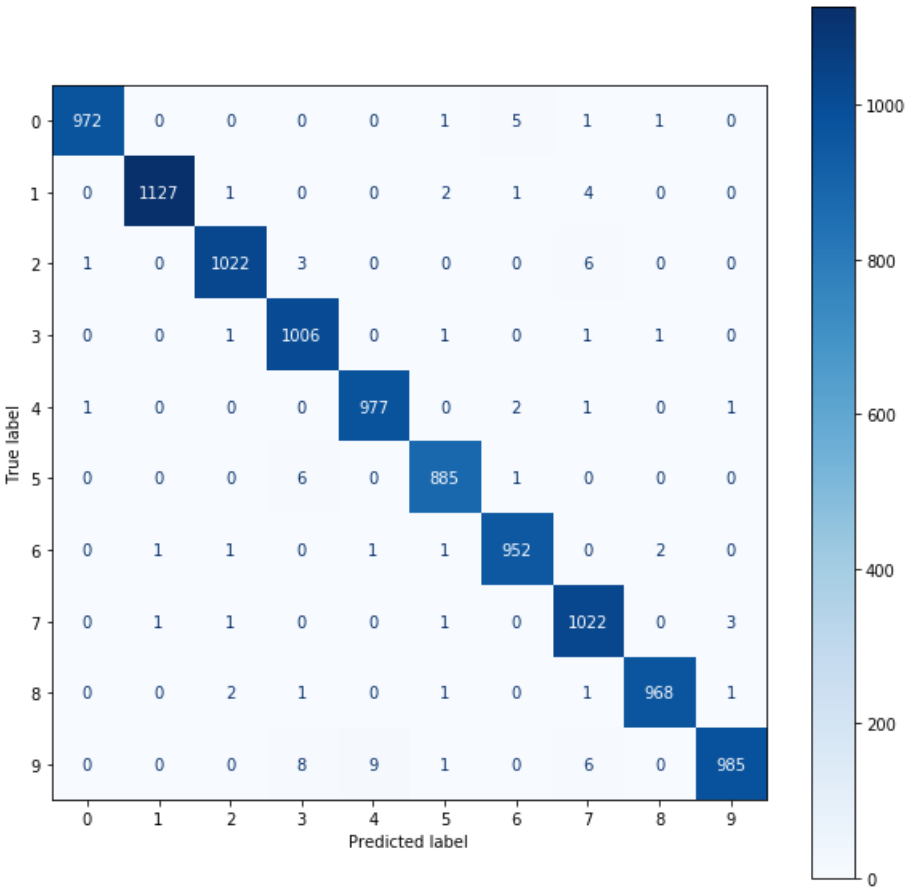
Wnioski

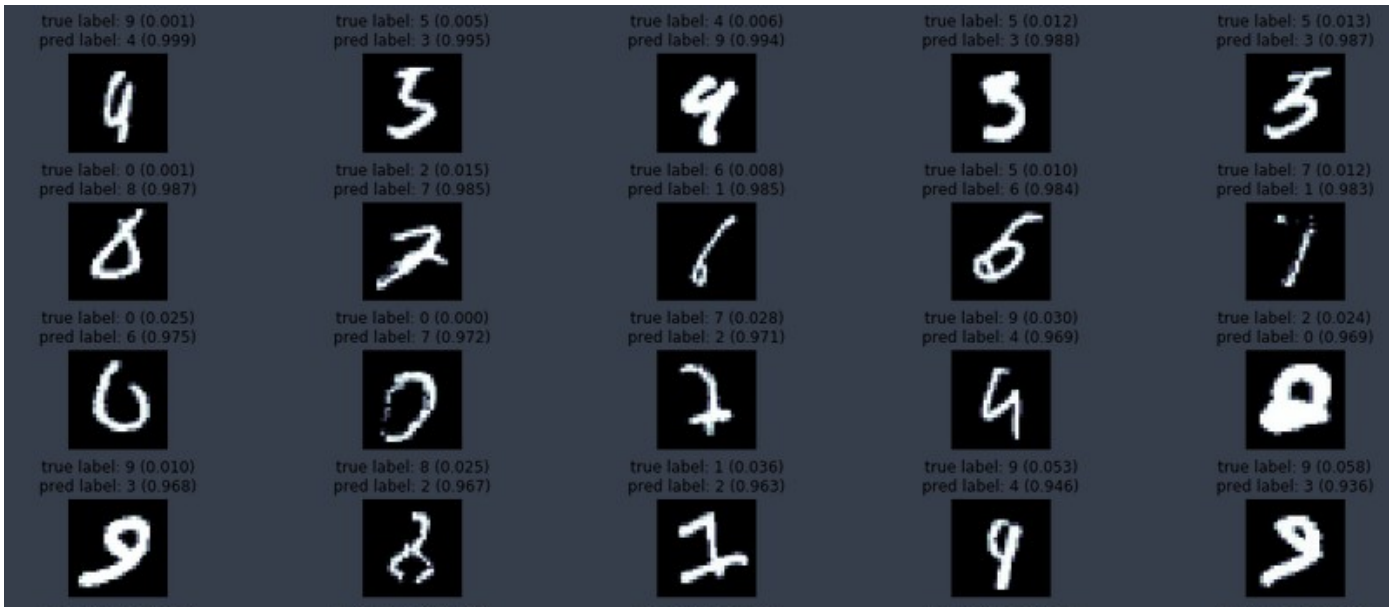
Cel zadania został osiągnięty. Wszystkie polecenia udało się wykonać. Kod programu dokładnie przeanalizowano i zapoznano się z budową konwolucyjnych sieci neuronowych złożonych z wielu warstw.

Zaobserwowano, że sieci te liczą dokładność rozpoznawania obrazków lepiej niż zwykłe sieci wielowarstwowe. Szczególnie dla sieci dwuwarstwowej z 500 i 250 neuronami. Dokładność po 20 iteracjach wyniosła 99.16%.

```
Epoch: 19 | Epoch Time: 0m 28s
        Train Loss: 0.029 | Train Acc: 99.06%
         Val. Loss: 0.029 |  Val. Acc: 99.25%
Epoch: 20 | Epoch Time: 0m 28s
        Train Loss: 0.029 | Train Acc: 99.11%
         Val. Loss: 0.040 |  Val. Acc: 99.04%
Test Loss: 0.025 | Test Acc: 99.16%
```

Wynik ten można uznać za bardzo dokładny. Dodawanie epok prawdopodobnie jeszcze by go zwiększyło, natomiast znacznie wydłużyłoby czas. Sieć pomyliła najwięcej razy 4 z 9 (9 razy) i 3 z 9 (8 razy).





Na powyższym rysunku widać, że mylone cyfry to skrajne wartości mogące stanowić ciężki wybór nawet dla człowieka (np. 2 w trzecim rzędzie, ostatniej kolumnie) . Natomiast widać, że mylone są 4, w których górna przerwa jest bardzo mała i dlatego sieć identyfikuje je jako 9. Również przekręcona 9 (ostatni rząd, pierwsza kolumna) została sklasyfikowana jako 8. Być może losowanie transformacji obrotu dla tensora z szerszego kąta wyeliminowało by ten błąd i jeszcze zwiększyło dokładność.