Ćwiczenie nr 5
Wprowadzenie do sztucznej inteligencji

Autorzy
Maciej Prostak
Łukasz Jaremek

1 Wprowadzenie

Tematem zadania jest klasyfikacja za pomocą sztucznych sieci neuronowych. Zbudowana przez nas sieć zostanie poddana treningu na zbiorze danych MNIST. Sprawdzimy jak zmiany takich parametrów jak współczynnik nauki sieci, architektura oraz czas treningu wpłynie na ostateczny wynik sieci.

1.1 Technologia

Rozwiązanie zostało zaimplementowane w języku Python (3.9.0) z wykorzystaniem bibliotek numpy (1.12.4), idx2numpy(1.2.3), matplotlib(3.4.3). Dodatkowo dla poprawy czytelności kodu zostały dołączone biblioteki black (21.9b0), flake8 (4.0.1) oraz isort (5.9.3).

1.2 Zbiór danych

Jednym z najbardziej podstawowych zbiorów danych w celu potwierdzenia działania sieci neuronowej jest to zbiór MNIST. Zawiera on obrazy w wielkości 28x28 pikseli przedstawiające ręcznie napisane cyfry od 0 do 9. Łącznie obrazów w naszym zbiorze jest 60 000. Dla każdego obrazu mamy wynik ostatecznej klasy.

Dodatkowo, przed trenowaniem na zbiorze MNIST w celu potwierdzenia działania naszej sieci wygenerowaliśmy zbiór treningowy XOR. Dane treningowe można bardzo łatwo wygenerować: xor(0, 0) = xor(1, 1) = 0 oraz xor(1, 0) = xor(0, 1) = 1.

1.3 Podział obowiązków

Praca odbywała się najczęściej w trybie pair-programming. Używaliśmy narzędzia live share w edytorze Visual Studio Code, żeby móc pisać jednocześnie w projekcie. Budowa klas potrzebnej do sieci, algorytm feed-forward i backpropagation zostały napisane w ten sposób.

Obowiązki Macieja:

- analiza danych,
- implementacja mini-batch,
- rysowanie wykresów,
- rozdział pierwszy raportu,
- dłuższe trenowanie.

Obowiązki Łukasza:

- wczytywanie zbioru danych z plików,
- wyliczanie macierzy pomyłek,
- obliczenie wyniku sieci: accuracy, precision, recall,
- analiza wyników sieci dla różnych parametrów.

1.4 Analiza danych

Przykładowe dane wejściowe z opisaną klasą można zobaczyć na rysunku Fig. 1. Dla każdej klasy został wybrany losowy obraz wejściowy i przedstawiony jako obraz w wielkości 28x28 pikseli.

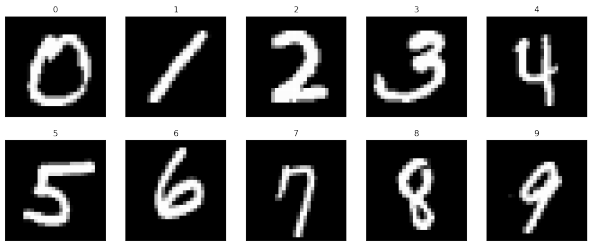

Fig. 1 Przykładowe dane ze zbioru MNIST.

Za pomocą histogramu sprawdziliśmy częstotliwość występowania każdej klasy w całym zbiorze (Fig. 2). Każdej klasy jest mniej więcej po równo (około 6 tysięcy przykładów na każdą klasę).

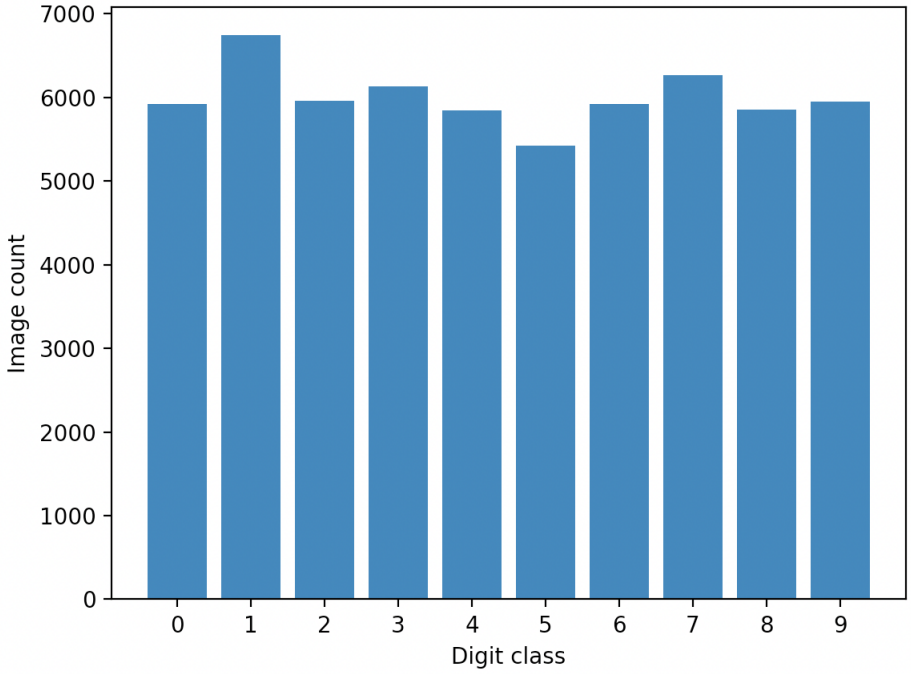

Fig. 2 Analiza częstotliwości występowania klas w zbiorze MNIST.

1.5 Inicjalizacja wartości wag oraz biases

Przed każdym treningiem w momencie utworzenia nowego obiektu naszej sieci neuronowej, przydzielamy wagom losowe wartości znajdujące się w przedziale od -1 do 1. Biases zostają również przydzielone losowo, ale od wartości -0.2 do 0.2.

1.6 Przykładowy wygląd sieci

Przykład architektury sieci w której mamy wejście (784 neurony), warstwę ukrytą (50 neuronów) oraz warstwa wyjściowa (10 neuronów) można zobaczyć na rysunku Fig. 3. Jako, że obrazek 28x28 pikseli musimy wprowadzić jako wektor, obrazy przed treningiem odpowiednio będą "spłaszczane". W naszej implementacji sieci, wejście (w tym przypadku 784 neuronów) nie traktujemy jako oddzielną warstwę.

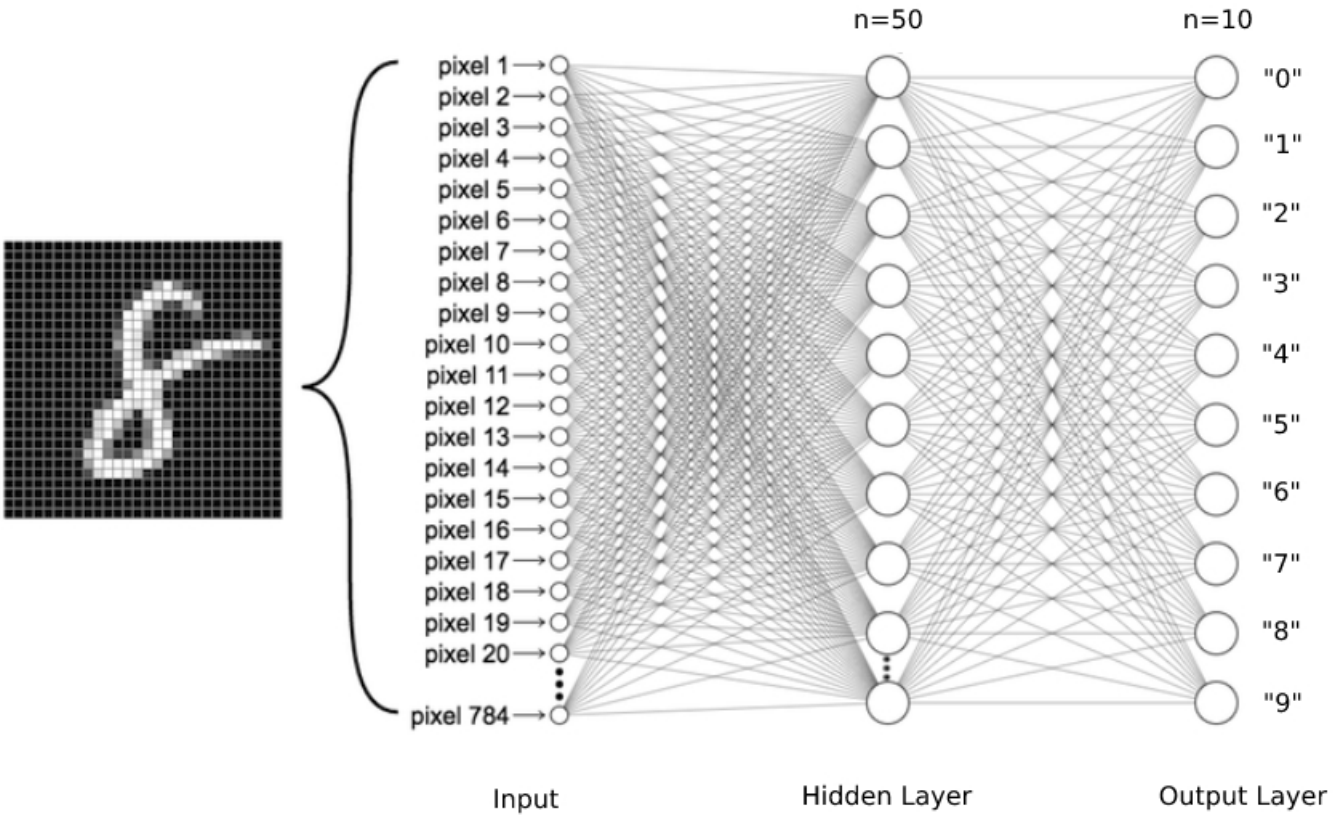

Fig. 3 Przykładowy wygląd sieci.

2 Wyniki

2.1 Współczynnik nauki sieci

Współczynnik nauki sieci (learning rate) mówi nam, jak szybko sieć ma się uczyć. Zbyt mały parametr sprawi, że tempo nauki będzie zbyt wolne, natomiast za duży doprowadzi do sytuacji, gdy nigdy nie znajdziemy najoptymalniejszego rozwiązania.

Na wykresach podanych poniżej widzimy jak parametr współczynnik sieci wpływa na wyniki modelu w kolejnych epokach. Dla małego "learning rate" wyniki bardzo wolno ulegają poprawie, ale wzrost jest jednak stabilniejszy niż przy większej wartości "learning rate".

Dla współczynnika równego 10 accuracy jest dość wysokie, jednak inne parametry takich jak recall oraz precision ulegają dużym zmianom. Sugeruje to, że wartość jest jednak trochę za duża.

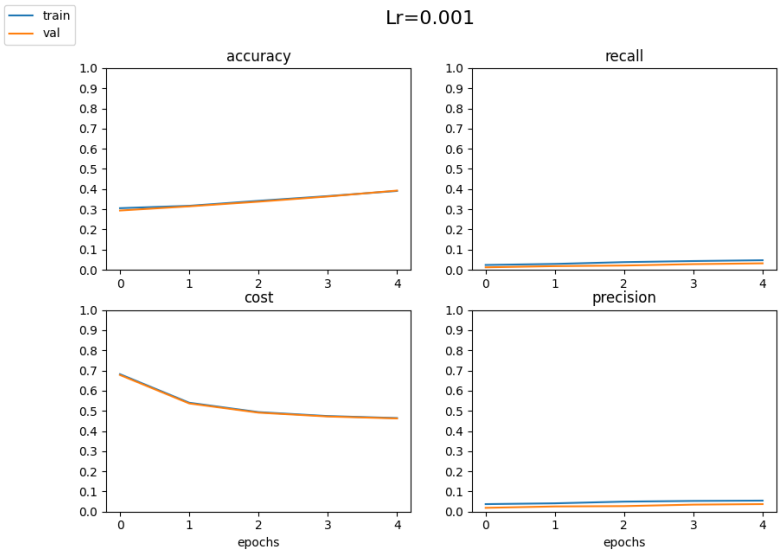

Fig. 4 Nauka sieci ze współczynnikiem nauki = 0.001

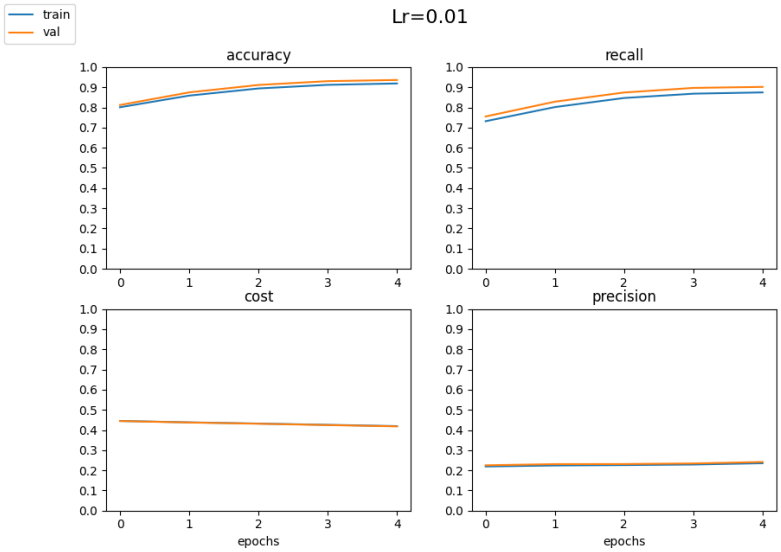

Fig. 5 Nauka sieci ze współczynnikiem nauki = 0.01

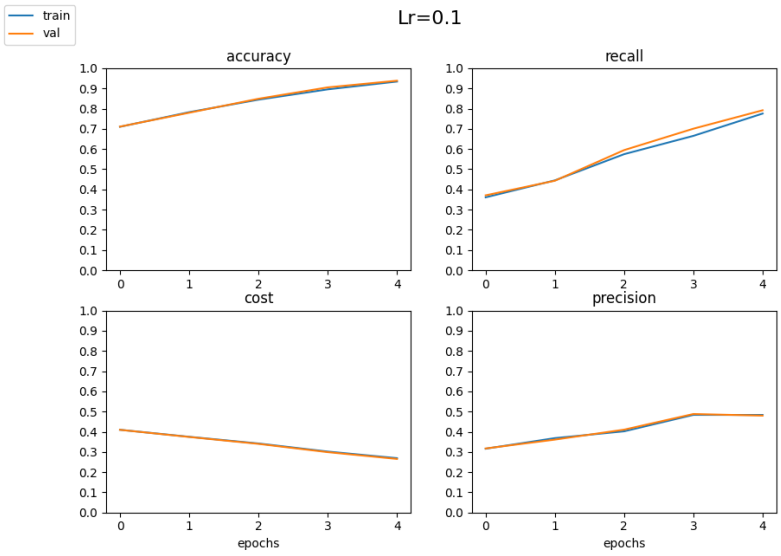

Fig. 6 Nauka sieci ze współczynnikiem nauki = 0.1

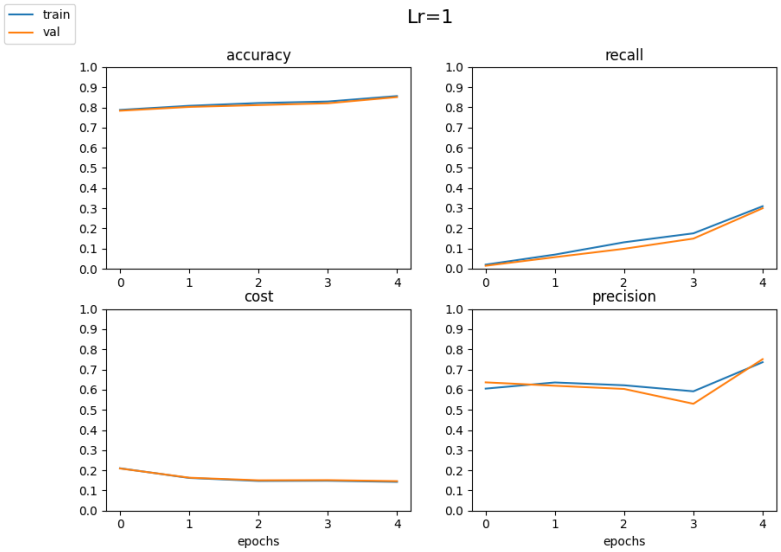

Fig. 7 Nauka sieci ze współczynnikiem nauki = 1

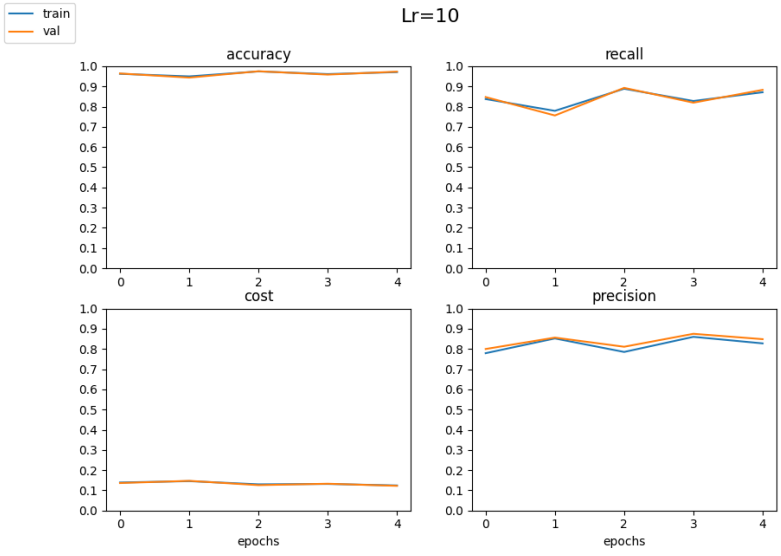

Fig. 8 Nauka sieci ze współczynnikiem nauki = 10

2.2 Liczba neuronów ukrytych

Można zauważyć jaki wpływ ma liczba neuronów w ukrytych warstwach. Dla uproszczenia sprawdzimy wyniki dla jednej ukrytej warstwy, której ustawimy różne liczby neuronów. Zbyt mała liczba doprowadza do sytuacji, gdy sieć się szybko "poddaje" i przestaje uczyć.

Dla małej liczby neuronów (w naszych przykładach n=1 oraz n=2) widzimy, że sieć nie uczy się problemu i wyniki są losowe. Dopiero przy większej ilości neuronów sieć zaczyna pokazywać obiecujące wyniki.

2.2.1 n=1

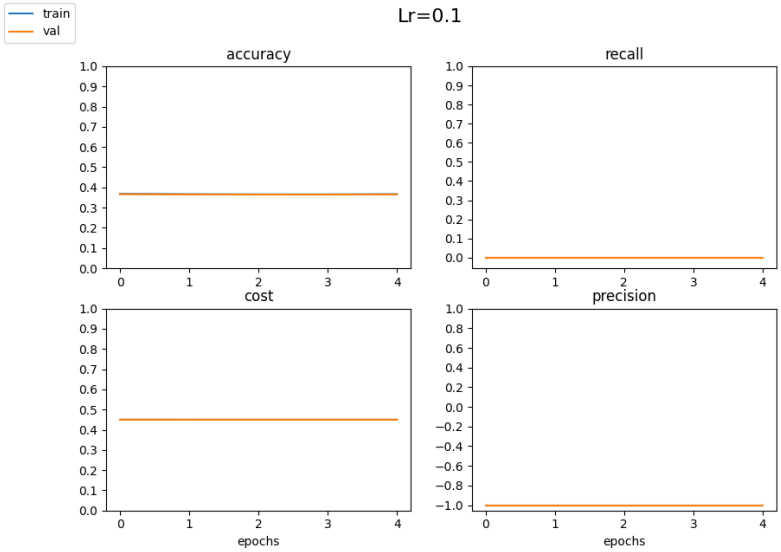

Fig. 9 Nauka sieci z liczbą neuronów = 1

2.2.2 n=2

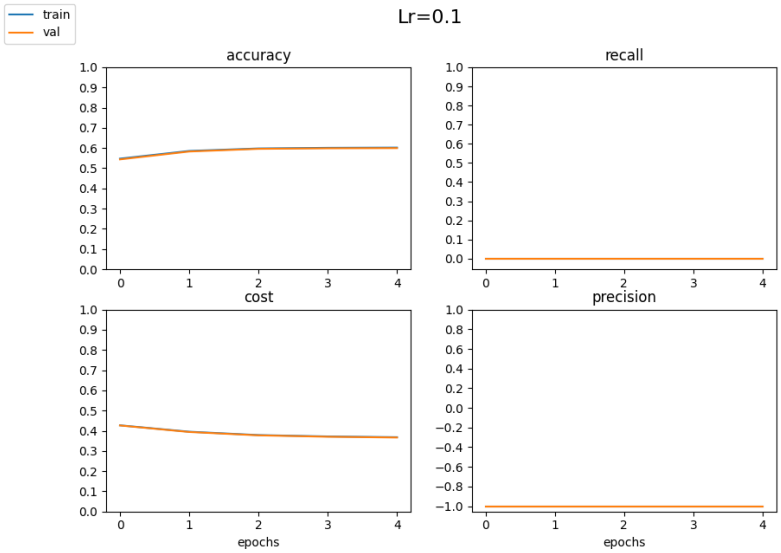

Fig. 10 Nauka sieci z liczbą neuronów = 2

2.2.3 n=5

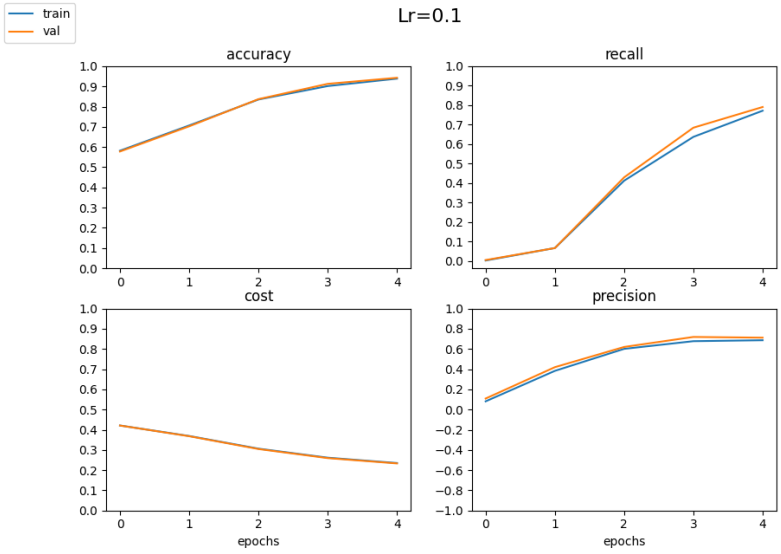

Fig. 11 Nauka sieci z liczbą neuronów = 5

2.2.4 n=10

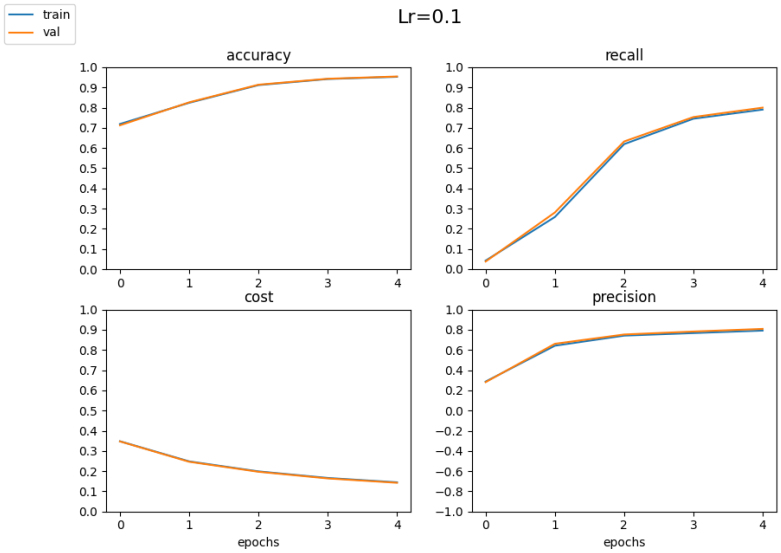

Fig. 12 Nauka sieci z liczbą neuronów = 10

2.2.5 n=50

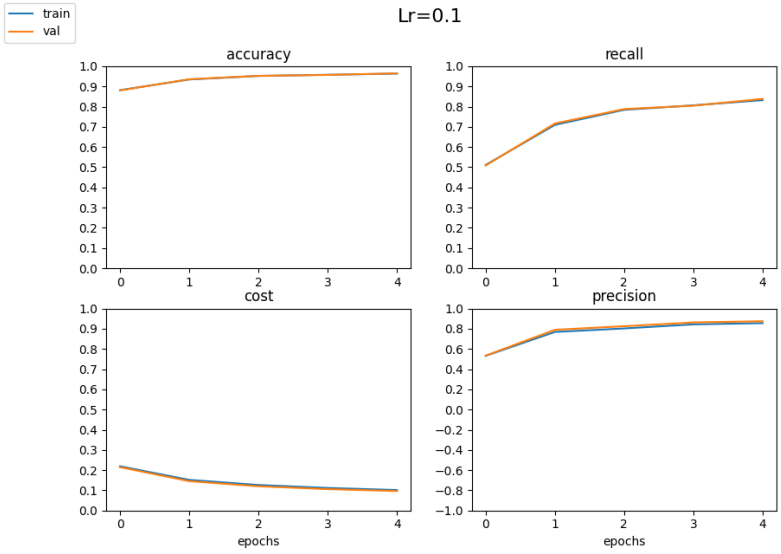

Fig. 13 Nauka sieci z liczbą neuronów = 50

2.2.6 n=100

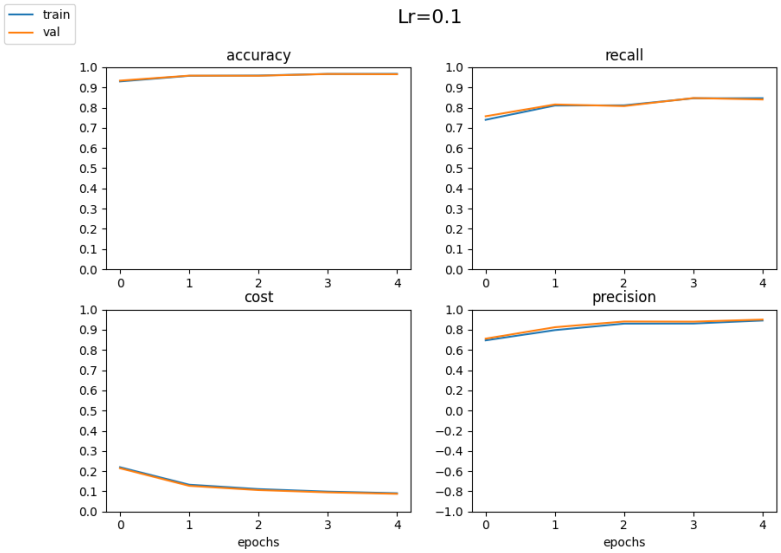

Fig. 14 Nauka sieci z liczbą neuronów = 100

2.3 Liczba ukrytych warstw

Odpowiednio będziemy zmieniać liczbę ukrytych warstw. Dla uproszczenia każda warstwa będzie posiadać po 10 neuronów. Tutaj można zauważyć ciekawe zjawisko; gdy liczba warstw ukrytych jest za duża, sieć się w ogóle nie uczy przy małej liczbie epok. Im więcej sieci tym więcej wymaganych epok, co wiąże się z dłuższym czasem wykonywania programu.

2.3.1 l=0

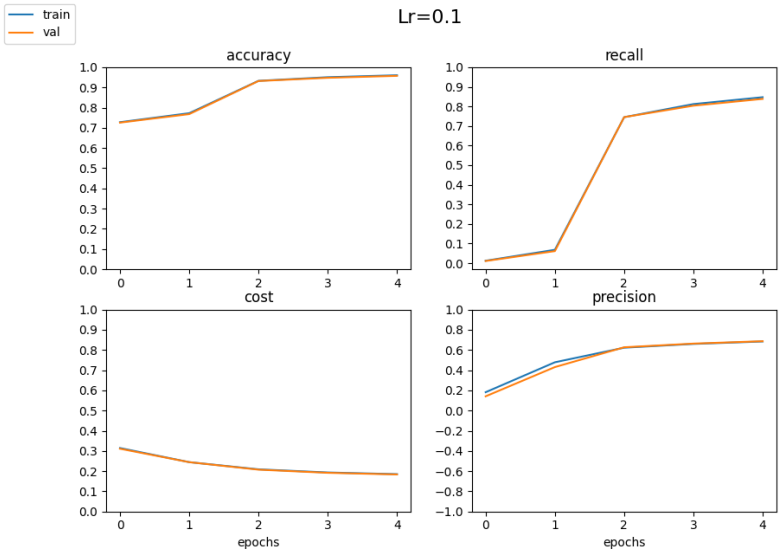

Fig. 15 Nauka sieci z liczbą ukrytych warstw = 0

2.3.2 l=1

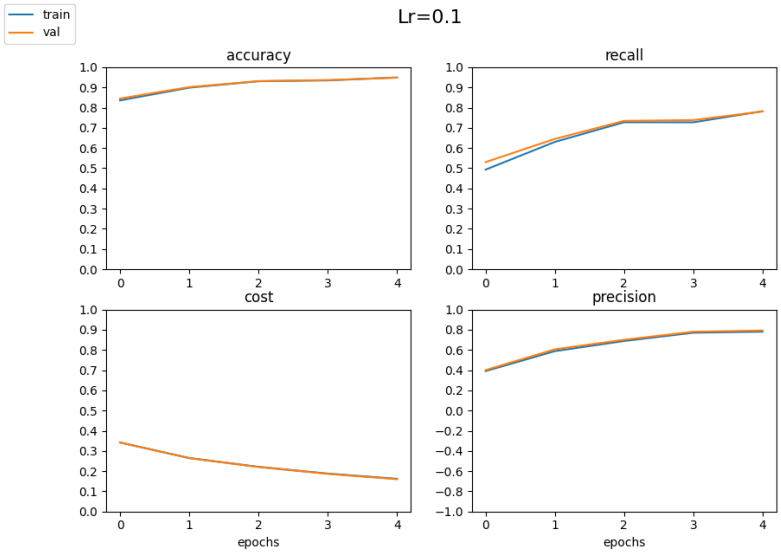

Fig. 16 Nauka sieci z liczbą ukrytych warstw = 1

2.3.3 l=2

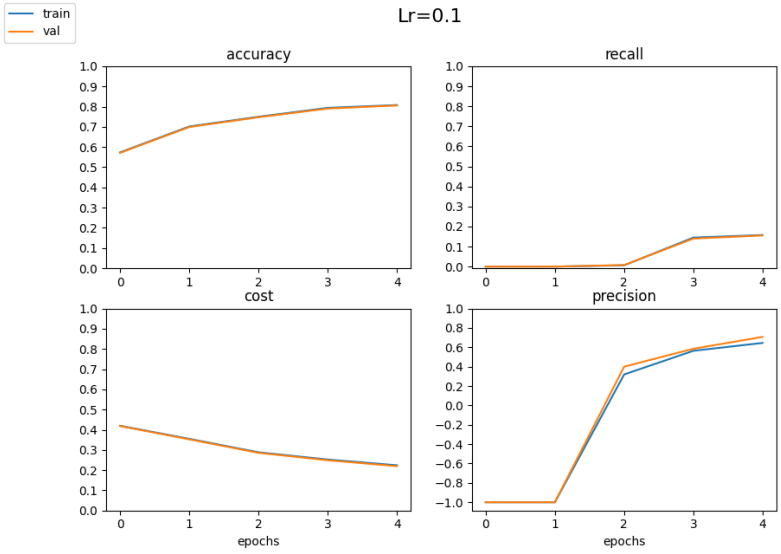

Fig. 17 Nauka sieci z liczbą ukrytych warstw = 2

2.3.4 l=5

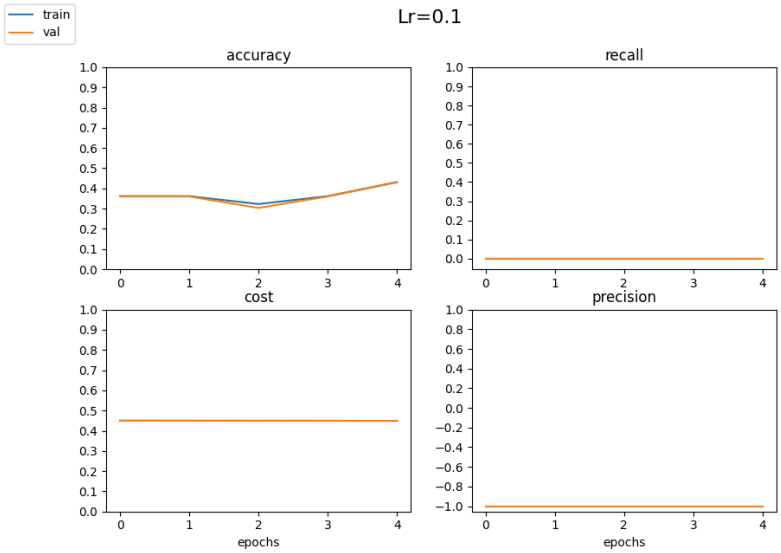

Fig. 18 Nauka sieci z liczbą ukrytych warstw = 5

2.3.5 l=10

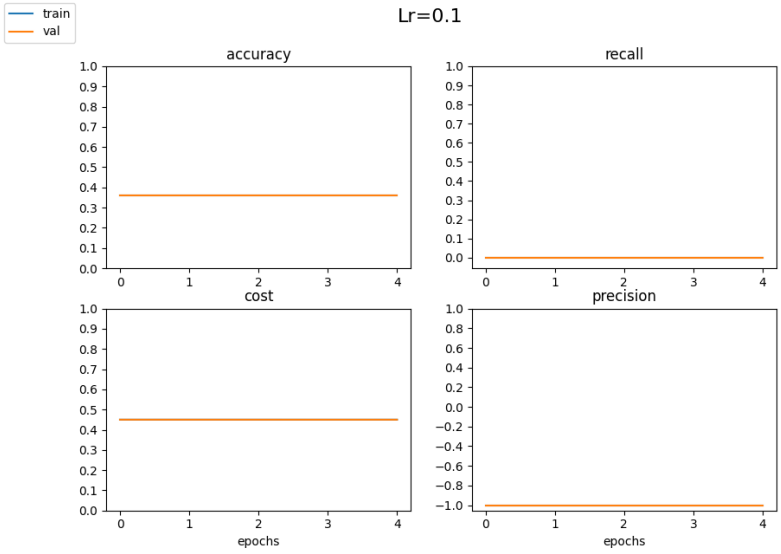

Fig. 19 Nauka sieci z liczbą ukrytych warstw = 10

2.4 Epoki

Sprawdźmy wpływ ilości epok na działanie sieci. Będziemy testować sieć z jedną warstwą ukrytą z trzydziestoma neuronami. Jak widać, większa liczba epok pozytywnie wpływa na efekt uczenia się sieci, lecz kosztuje nas więcej czasu.

2.4.1 l=2

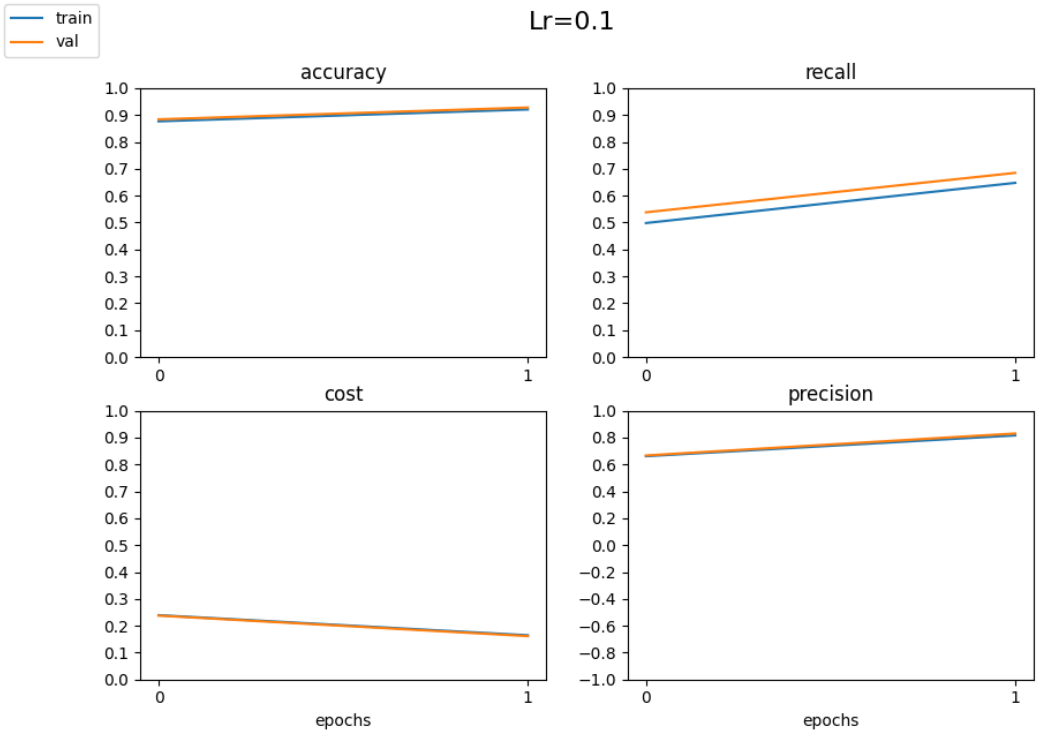

Fig. 20 Nauka sieci z liczbą epok = 2

2.4.2 l=3

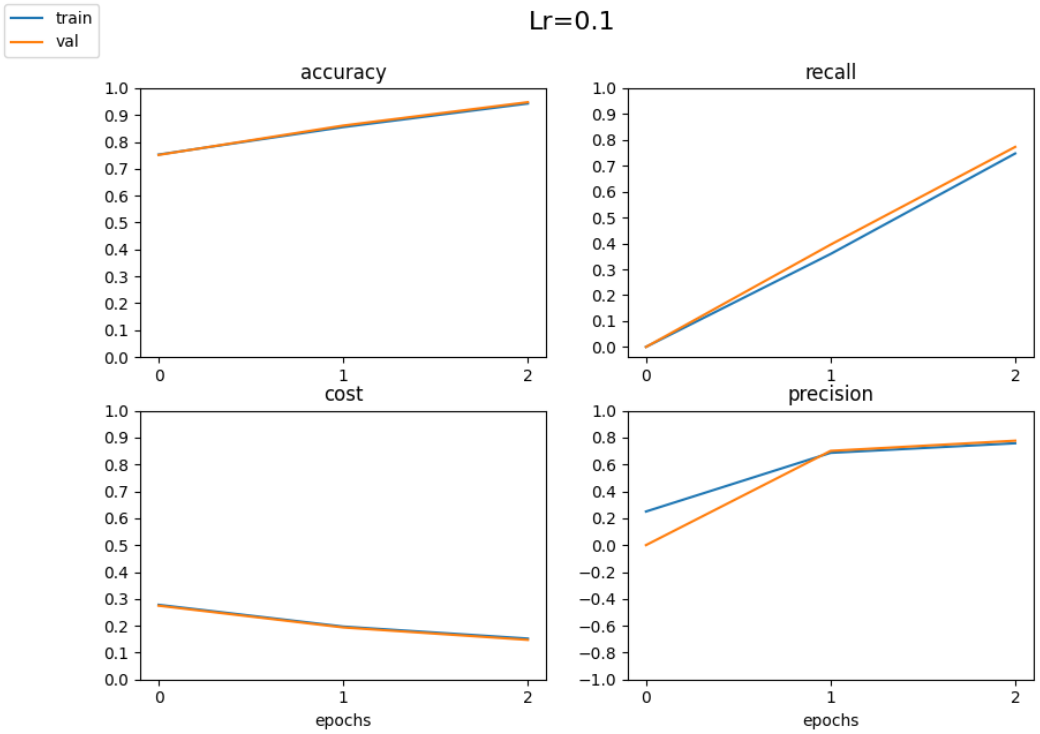

Fig. 21 Nauka sieci z liczbą epok = 3

2.4.3 l=5

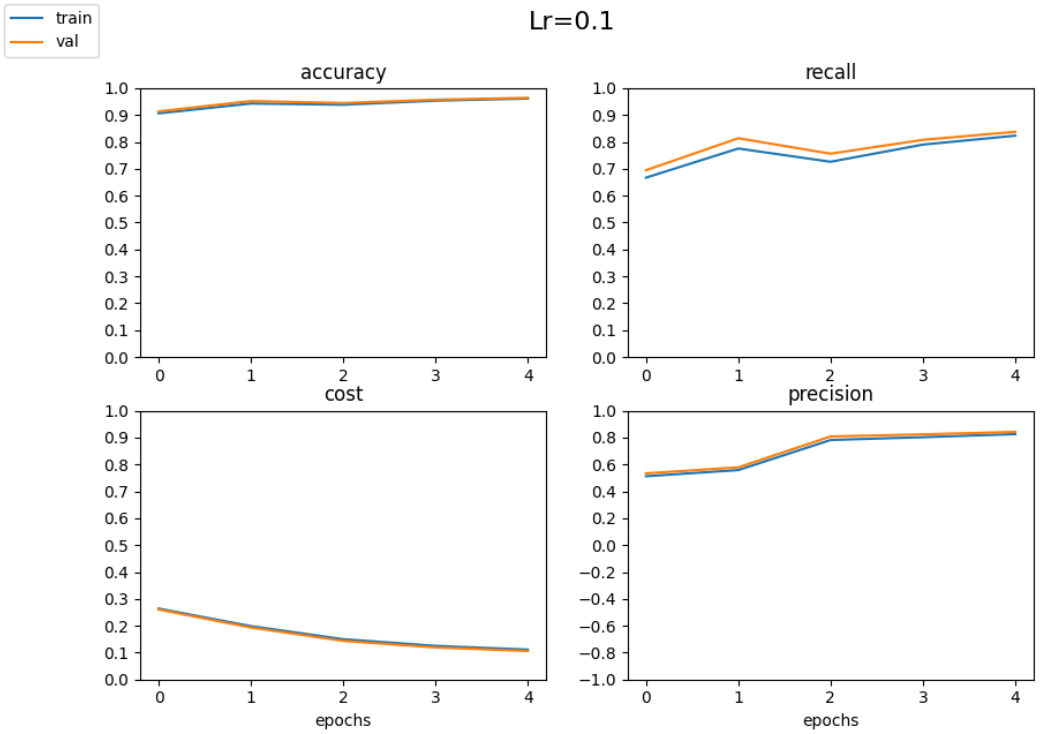

Fig. 22 Nauka sieci z liczbą epok = 5

2.4.4 l=10

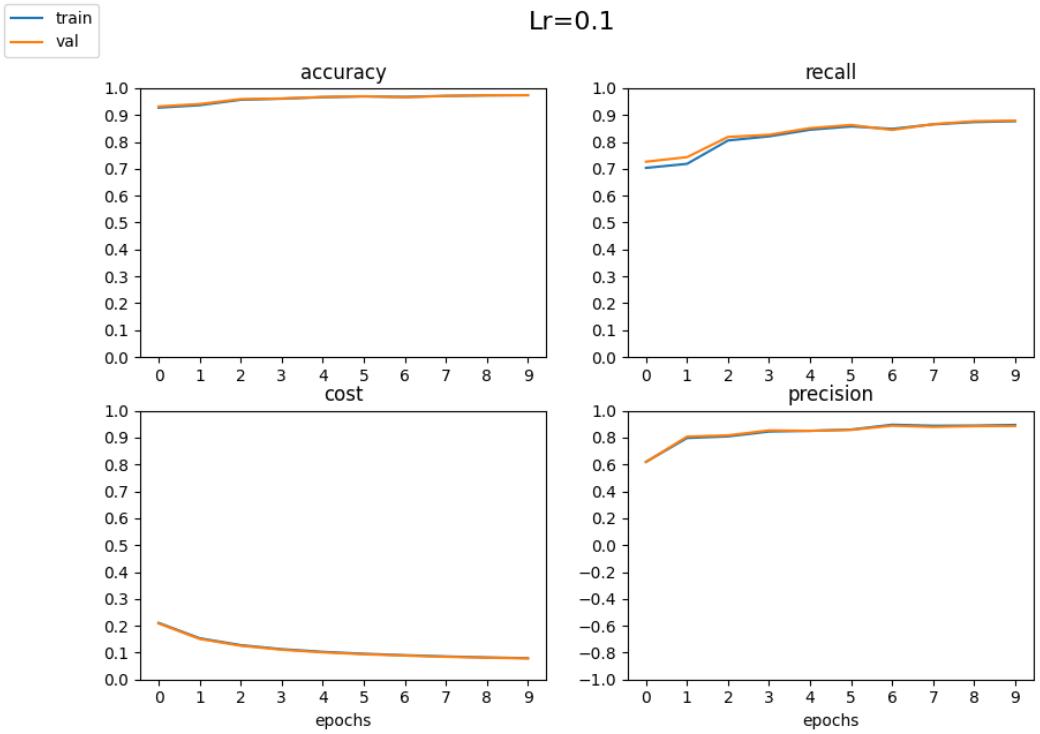

Fig. 23 Nauka sieci z liczbą epok = 10

2.5 Sortowanie danych treningowych i walidacyjnych

2.5.1 Z sortowaniem po klasie wynikowej

Po odpowiednim posortowaniu po klasie wynikowej widzimy na wykresie (Fig. 24), że accuracy jest dość niskie (około 10%), a precision bardzo wysokie (około 100%). Po sprawdzaniu wartości w confusion matrix zaobserwowaliśmy, że sieć zawsze wybiera jedną klasę (w tym przypadku zawsze klasę 10).

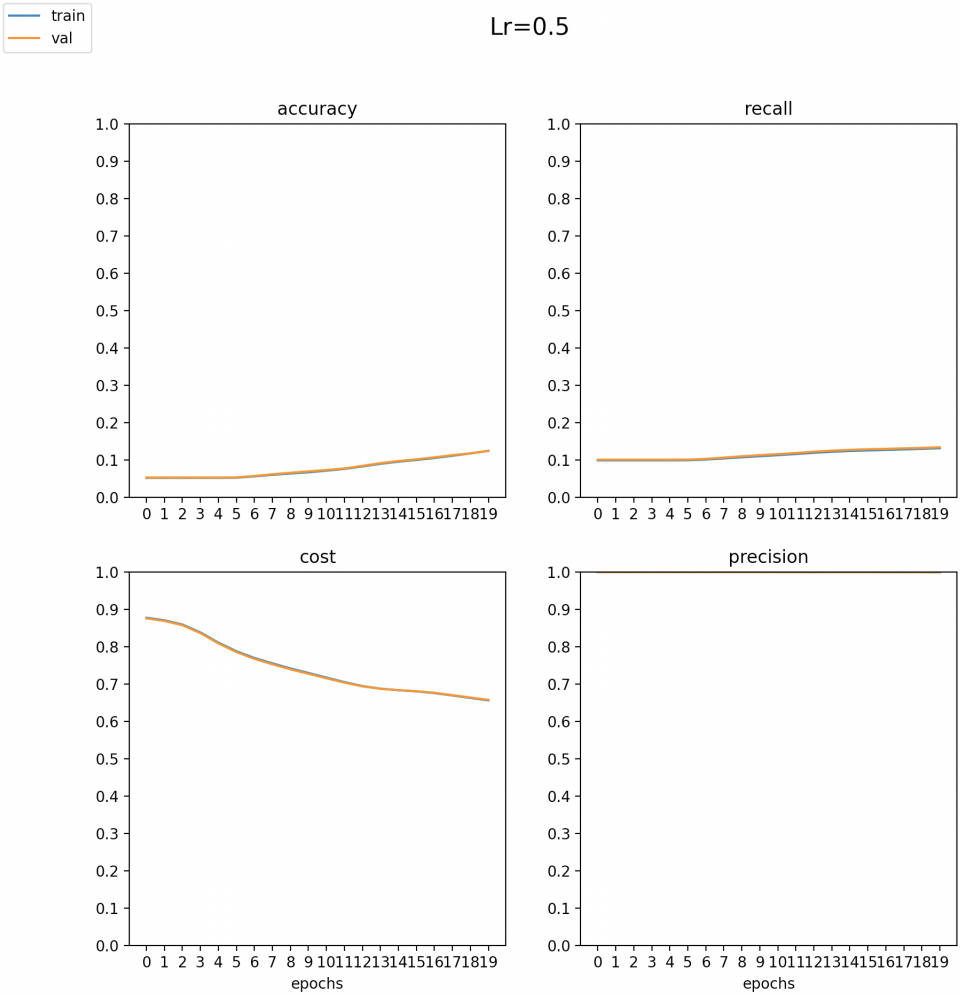

Fig. 24 Wyniki modelu z sortowaniem klasy.

2.5.2 Bez sortowania po klasie wynikowej

Ważne jest, żeby zadbać o to, żeby w trakcie nauki w mini-batchach były odpowiednio "różne" dane. W wszystkich testach zadbaliśmy o to, żeby dane były odpowiednio losowane.

2.6 Dłuższe trenowanie

W celu lepszego przetestowania naszej sieci neuronowej, trenowanie odbyło się na architekturze (764 warstwa wejściowa, 50 neuronów warstwa ukryta, 50 neuronów warstwa ukryta, 10 warstw wyjściowych) na 100 epokach. Wyniki trenowania i walidacji można zauważyć na wykresie poniżej. Jak na sieć działająca na obrazach bez mechanizmu konwolucji, sieć uzyskała całkiem dobre wyniki. Widać, jak w pewnym momencie nauka modelu poprawia tylko wyniki na danych treningowych, a wyniki na danych walidacyjnych utrzymują się na stałym poziomie. Trenowanie i walidacja wyniku naszej sieci na każdą epokę trwała około 15 sekund. Trenowanie trwało łącznie 25 minut.

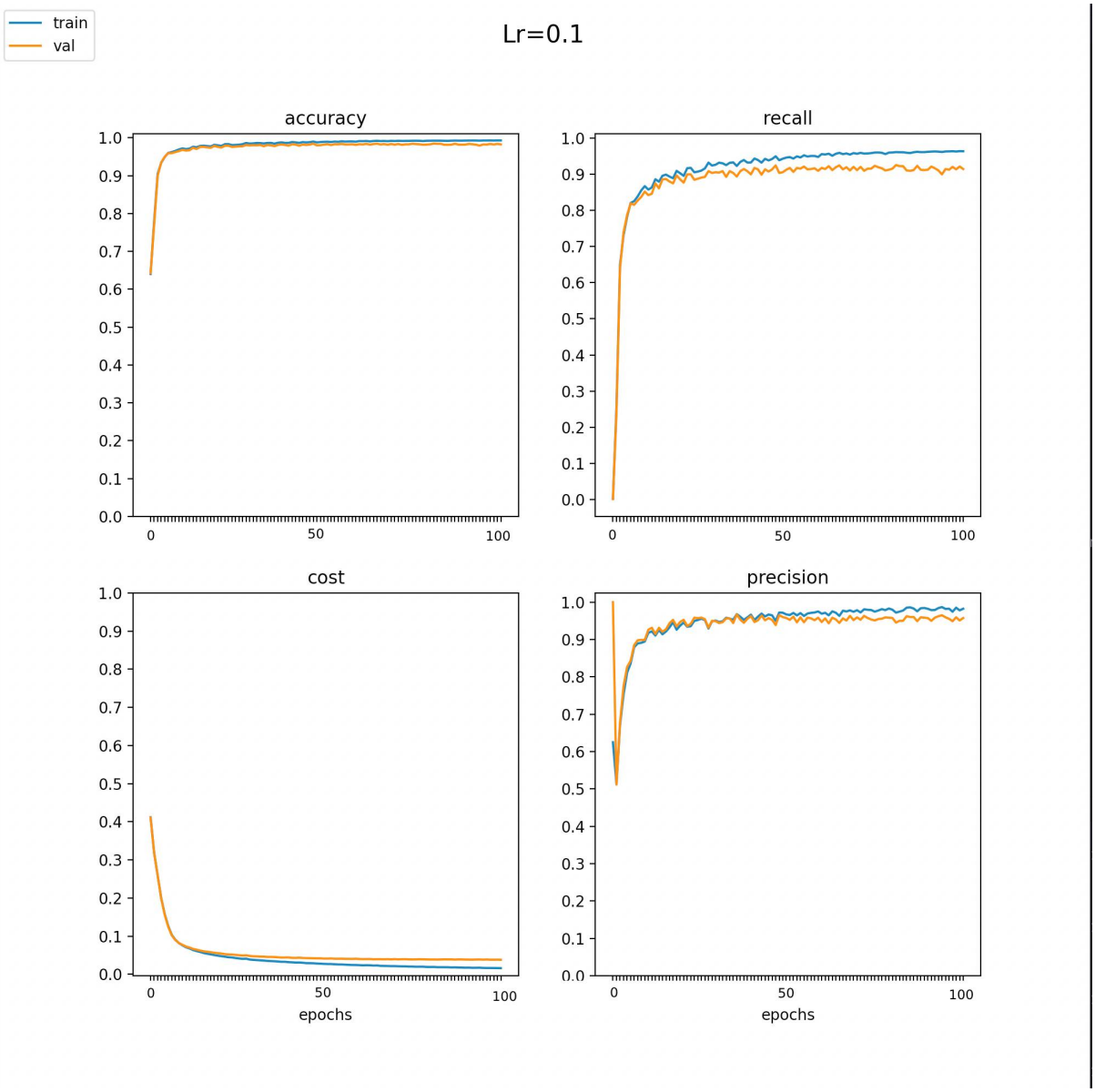

Fig. 25 Przykłady źle sklasyfikowanych cyfr.

2.7 Błędne predykcje

Poniżej zamieszczone zostały przykłady, w których nasz nauczony model się mylił. Widać, że niektóre przykłady jest ciężko nawet nam zinterpretować. Istnieje jednak trochę przykładów, w których dane są przejrzyste ale sieć źle sklasyfikowała przykład.

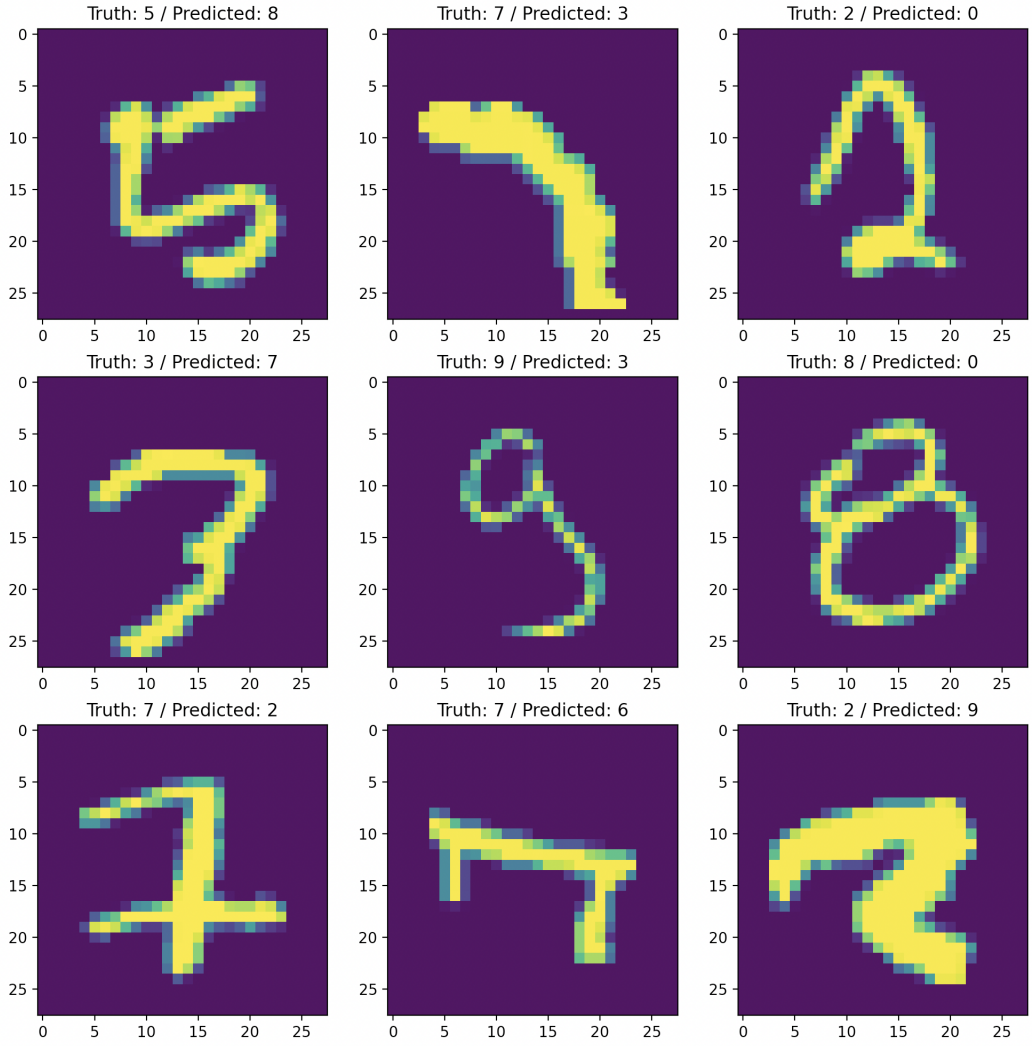

Fig. 26 Przykłady źle sklasyfikowanych cyfr.

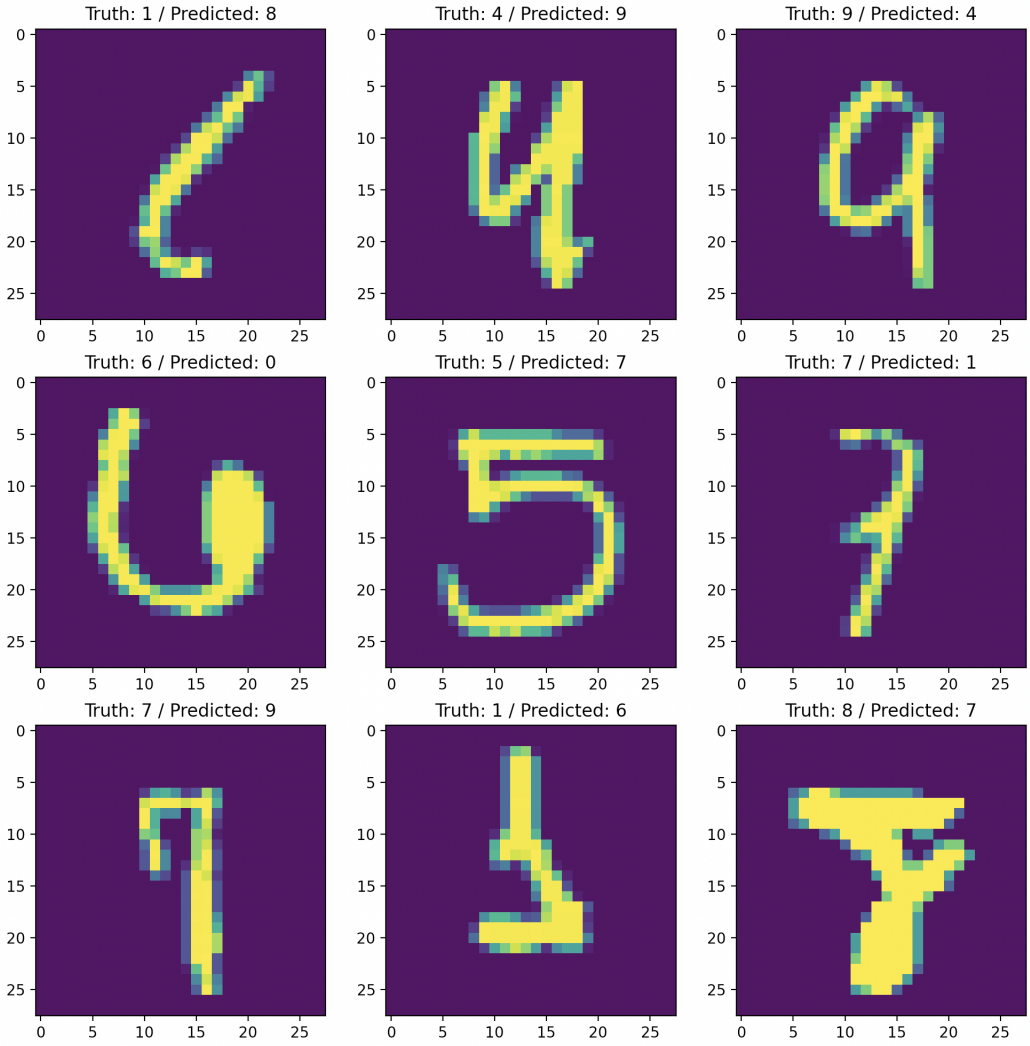

Fig. 27 Przykłady źle sklasyfikowanych cyfr.

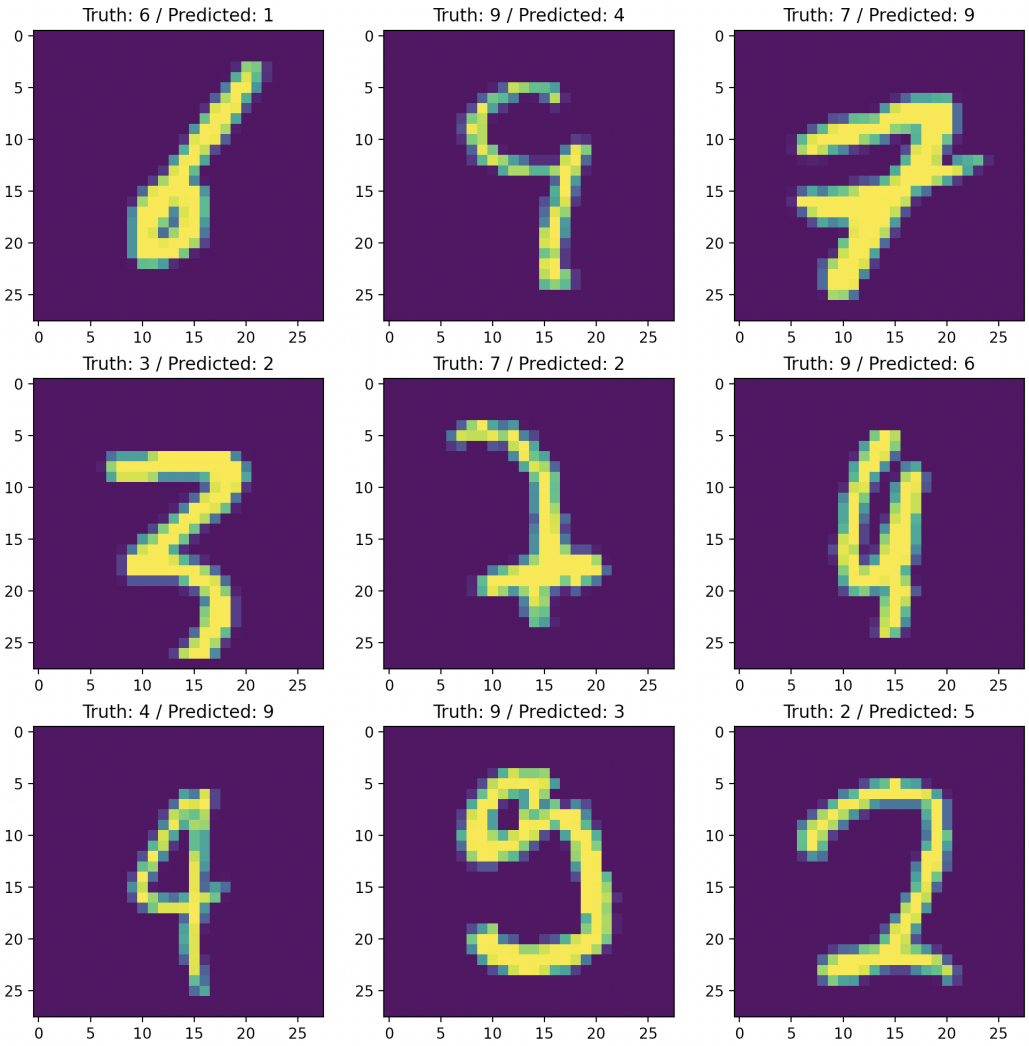

Fig. 28 Przykłady źle sklasyfikowanych cyfr.

3 Wnioski

Sieci neuronowe są powszechnie używane przy wielu rodzajach problemów. Przy względnie krótkim czasie pozwalają rozwiązać złożone problemy, lecz potrzebują dużej ilości danych przygotowanych w odpowiedni sposób oraz użytkownik trenujący sieć, musi skonfigurować wiele parametrów, co nie jest najłatwiejszym zadaniem dla niedoświadczonych osób.

W celu poprawienia skuteczności naszego modelu można by było skorzystać z mechanizmu konwolucji, która pozwala na wyciąganie odpowiednich cech obrazu. Inną czynnością, która mogłaby poprawić skuteczność jest odpowiednia augmentacja danych w trakcie nauki modelu. Pozwala ona na sztuczne stworzenie większego zbioru danych poprzez prostą manipulację zdjęć (na przykład: move, zoom).