POLITECHNIKA WARSZAWSKA
WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

DIGIT RECOGNIZER
Raport z projektu

Autorzy: **Katarzyna Gołębiewska**
Mateusz Bieńkowski

Prowadzący: **mgr Dominika Basaj**
Przedmiot: **Metody Sztucznej Inteligencji 2**
Kierunek studiów: **informatyka**

Warszawa, dnia 04.06.2018

Spis treści

1 Specyfikacja **2**
1.1 Opis problemu 2
1.2 Cel projektu i wybrane algorytmy 2
1.3 Technologia 2

2 Implementacja, rezultaty i wnioski **3**
2.1 CNN 3
2.1.1 Opis algorytmu 3
2.1.2 Implementacja 3
2.1.3 Wyniki i wnioski 4
2.2 SVM 8
2.2.1 Opis algorytmu 8
2.2.2 Implementacja 8
2.2.3 Wyniki i wnioski 8
2.3 Porównanie metod 11

1 Specyfikacja

1.1 Opis problemu

Tematem projektu jest klasyfikacja ręcznie pisanych cyfr ze zbioru MNIST. Zbiór ten składa się z 70 000 obrazów o wymiarach 28x28 pikseli i wartościach z przedziału [0-255] (kolorystyka czarno-biała). Każda z cyfr zapisana jest w postaci wektora, którego elementy odpowiadają kolejnym pikselom. Dodatkowo, dla każdego wektora dostarczona jest również etykieta będąca cyfrą, którą reprezentuje obrazek.

Rysunek 1: Przykładowe dane ze zbioru MNIST

1.2 Cel projektu i wybrane algorytmy

Cel projektu to analiza działania i porównanie skuteczności klasyfikacji w zależności od doboru algorytmu i jego parametrów. Algorytmy poddawane badaniom w niniejszym projekcie:

- CNN - konwolucyjne sieci neuronowe;
- SVM - maszyna wektoryów nośnych.

1.3 Technologia

Projekt został zaimplementowany w języku Python, przy użyciu biliotek takich jak:

- Keras (backend - TensorFlow) - implementacja CNN;
- Skicit-learn - implementacja SVM.

2 Implementacja, rezultaty i wnioski

2.1 CNN

2.1.1 Opis algorytmu

Konwolucyjne sieci neuronowe stanowią pewną podklasę sieci neuronowych powszechnie stosowanych m.in. przy problemach związanych z klasyfikacją danych obrazowych.

Jedną z podstawowych operacji stanowi operacja *konwolucji*, która polega na przefiltrowaniu sygnału poruszając się po nim w ramach mniejszego okna, tak aby uchwycić cechy z sąsiedztwa. Z konwolucją związany jest filtr - macierz. Przesuwając ją po obrazie mnożymy jej wartości przez wartości odpowiadających im pixeli, a następnie wszystko sumujemy. Filtry pozwalają na wydobycie różnych cech obrazu, np. krawędzi, charakterystycznych kształtów itd. Wagi macierzy filtrów są aktualizowane przez sieć podczas procesu uczenia - sieć dąży do tego, by błąd wyrażony funkcją straty był jak najmniejszy.

Operacja *maxpooling* z kolei odpowiada za zmniejszenie wymiarowości oraz w pewnym sensie za dostarczenie „wyabstrahowanej formy" danych.

Dropout polega na „ucinaniu" części połączeń (losowo). Ma to na celu zapobieganiu zjawisku przeuczenia.

2.1.2 Implementacja

Zbiór 70 000 obrazów wraz z etykietami został podzielony na 3 podzbiory:

- treningowy, na podstawie którego sieć uczy się rozpoznawać cyfry - 48 000 obrazów;
- walidacyjny - 12 000 obrazów;
- testowy - 10 000 obrazów.

W sposób eksperymentalny skonstruowane zostały 4 „szkielety" modeli sieci. Znajdują się one w pliku *NetworkModels.py* i są parametryzowane wielkością filtrów, okna max pooling czy też wielkością dropout. Za funkcję aktywacji (według której obliczana jest wartość wyjścia neuronów) została przyjęta funkcja *ReLU* mająca postać:

$$f(x) = max(0, x).$$

Wyjątkiem są ostatnie warstwy każdego z modeli, w których rolę funkcji aktywacji odgrywa $softmax$. Jest ona przystosowana do problemów klasyfikacyjnych, w których wykorzystywana jest reprezentacja zmiennej wyjściowej typu jeden-z-N. W połączeniu z funkcją błędu pozwala szacować przy pomocy prawdopodobieństwa przynależności badanego obiektu do poszczególnych klas.

Parametryzacja szkieletów modeli pozwoliła na wykonanie obliczeń dla różnych kombinacji parametrów. W ten sposób uzyskano **38 różnych modeli sieci konwolucyjnych**. Dla każdego z nich (w celu wstępnej oceny jakości) uruchomiono proces uczenia przez 1 epokę. Jako funkcję straty przyjęto *categorical crossentropy*. Natomiast jako optymalizator posłużyła metoda *AdaDelta*.

Wartości *accuracy* (dokładność), *loss* (określa błąd) dla poszczególnych zbiorów (treningowego, walidacyjnego i testowego) oraz czas trwania obliczeń dla wszystkich modeli sieci zostały zapisane w pliku *CNN_ALL.csv*.

Na podstawie tych danych byliśmy w stanie wybrać **6 najlepiej prosperujących modeli** i to je poddaliśmy dalszym badaniom. Mianowicie dla każdego z nich, ponownie

zostały uruchomione obliczenia, lecz tym razem każdy model był trenowany przez **30 epok**. Przewidywaliśmy, iż wraz ze wzrostem numeru epoki, do pewnego momentu, dokładność na zbiorze testowym będzie rosła, a błąd malał. Natomiast dla epok późniejszych prognozowaliśmy stabilizację tych wartości. Wyniki omawianych eksperymentów oraz czas ich trwania znajdują się w pliku *CNN_BEST.csv*.

2.1.3 Wyniki i wnioski

Poniższa tabela zawiera wyniki uczenia sieci odpowiadającej każdemu z 38 modeli przez jedną epokę.

ID	model	test_acc	test_loss	train_acc	train_loss	val_acc	val_loss
0	model1, kernel: (2,2), pool: (2, 2), dropout: 0.1	[0.9734]	[0.0861]	[0.9175]	[0.2665]	[0.9748]	[0.0928]
1	model1, kernel: (2,2), pool: (2, 2), dropout: 0.2	[0.973]	[0.0894]	[0.9186]	[0.2703]	[0.9725]	[0.0917]
2	model1, kernel: (2,2), pool: (2, 2), dropout: 0.3	[0.9732]	[0.0848]	[0.9186]	[0.2623]	[0.9729]	[0.0945]
3	model1, kernel: (2,2), pool: (2, 2), dropout: 0.4	[0.974]	[0.0816]	[0.9108]	[0.2878]	[0.9745]	[0.0887]
4	model1, kernel: (2,2), pool: (2, 2), dropout: 0.5	[0.971]	[0.0969]	[0.9057]	[0.3076]	[0.969]	[0.1055]
5	model1, kernel: (7,7), pool: (2, 2), dropout: 0.1	[0.9844]	[0.0462]	[0.9351]	[0.2093]	[0.9842]	[0.0528]
6	model1, kernel: (7,7), pool: (2, 2), dropout: 0.2	[0.9804]	[0.0602]	[0.9359]	[0.203]	[0.9797]	[0.0669]
7	model1, kernel: (7,7), pool: (2, 2), dropout: 0.3	[0.9761]	[0.0718]	[0.9326]	[0.2196]	[0.9762]	[0.0759]
8	model1, kernel: (7,7), pool: (2, 2), dropout: 0.4	[0.9832]	[0.0475]	[0.9292]	[0.2234]	[0.9812]	[0.0599]
9	model1, kernel: (7,7), pool: (2, 2), dropout: 0.5	[0.9837]	[0.0517]	[0.9286]	[0.2295]	[0.9807]	[0.0641]
10	model1, kernel: (12,12), pool: (2, 2), dropout: 0.1	[0.8918]	[0.3427]	[0.6801]	[0.9657]	[0.8908]	[0.3439]
11	model1, kernel: (12,12), pool: (2, 2), dropout: 0.2	[0.8626]	[0.4187]	[0.6694]	[1.0069]	[0.8618]	[0.4173]
12	model1, kernel: (12,12), pool: (2, 2), dropout: 0.3	[0.6866]	[0.8727]	[0.5015]	[1.4249]	[0.6952]	[0.8581]
13	model1, kernel: (12,12), pool: (2, 2), dropout: 0.4	[0.7534]	[0.9042]	[0.4716]	[1.5381]	[0.7566]	[0.8989]
14	model1, kernel: (12,12), pool: (2, 2), dropout: 0.5	[0.6714]	[1.147]	[0.4227]	[1.6343]	[0.6769]	[1.152]
15	model2, kernel: (2,2), pool: (2, 2)	[0.9543]	[0.155]	[0.8885]	[0.3714]	[0.9532]	[0.1628]
16	model2, kernel: (7,7), pool: (2, 2)	[0.9763]	[0.078]	[0.9269]	[0.2405]	[0.9751]	[0.0885]
17	model2, kernel: (12,12), pool: (2, 2)	[0.9758]	[0.078]	[0.9242]	[0.2503]	[0.9713]	[0.0949]
18	model2, kernel: (17,17), pool: (2, 2)	[0.9676]	[0.1065]	[0.9099]	[0.2985]	[0.9615]	[0.1196]
19	model3, kernel: (2,2)	[0.9608]	[0.1432]	[0.8961]	[0.3437]	[0.9588]	[0.1505]
20	model3, kernel: (7,7)	[0.9753]	[0.0793]	[0.9345]	[0.2114]	[0.9721]	[0.0931]
21	model3, kernel: (12,12)	[0.9738]	[0.0884]	[0.9298]	[0.2406]	[0.9702]	[0.0996]
22	model3, kernel: (17,17)	[0.9666]	[0.1089]	[0.9157]	[0.2727]	[0.9667]	[0.1127]
23	model4, kernel1: (2,2), kernel2:(2, 2)	[0.9721]	[0.0963]	[0.9135]	[0.2903]	[0.9709]	[0.1076]
24	model4, kernel1: (2,2), kernel2:(7, 7)	[0.9797]	[0.0652]	[0.9404]	[0.2005]	[0.9777]	[0.0734]
25	model4, kernel1: (2,2), kernel2:(12, 12)	[0.9786]	[0.0689]	[0.9395]	[0.198]	[0.9762]	[0.0753]
26	model4, kernel1: (2,2), kernel2:(17, 17)	[0.9698]	[0.1026]	[0.9241]	[0.2482]	[0.9687]	[0.1078]
27	model4, kernel1: (7,7), kernel2:(2, 2)	[0.9741]	[0.0801]	[0.937]	[0.2121]	[0.9718]	[0.0904]
28	model4, kernel1: (7,7), kernel2:(7, 7)	[0.9806]	[0.0604]	[0.9366]	[0.2097]	[0.9785]	[0.0699]
29	model4, kernel1: (7,7), kernel2:(12, 12)	[0.9626]	[0.1113]	[0.8859]	[0.4117]	[0.9628]	[0.1261]
30	model4, kernel1: (7,7), kernel2:(17, 17)	[0.8635]	[0.4611]	[0.7927]	[0.6933]	[0.8612]	[0.4532]
31	model4, kernel1: (12,12), kernel2:(2, 2)	[0.9797]	[0.0679]	[0.9347]	[0.2173]	[0.976]	[0.0795]
32	model4, kernel1: (12,12), kernel2:(7, 7)	[0.8662]	[0.4663]	[0.8704]	[0.449]	[0.8691]	[0.4635]
33	model4, kernel1: (12,12), kernel2:(12, 12)	[0.8991]	[0.3379]	[0.7694]	[0.7372]	[0.8999]	[0.3275]
34	model4, kernel1: (12,12), kernel2:(17, 17)	[0.2153]	[2.0082]	[0.1928]	[2.13]	[0.2135]	[1.9951]
35	model4, kernel1: (17,17), kernel2:(2, 2)	[0.9609]	[0.1212]	[0.9162]	[0.2777]	[0.9608]	[0.1239]
36	model4, kernel1: (17,17), kernel2:(7, 7)	[0.9679]	[0.1059]	[0.9005]	[0.3199]	[0.9644]	[0.1143]
37	model4, kernel1: (17,17), kernel2:(12, 12)	[0.2139]	[2.0341]	[0.1778]	[2.1933]	[0.2142]	[2.0412]

Rysunek 2: Wyniki działania algorytmu dla poszczególnych modeli sieci

Analizując wyniki, które zawiera powyższa tabela, można zauważyć jak istotny wpływ na dokładność klasyfikacji ma budowa sieci - sposób konfiguracji warstw, ich ilość oraz parametry.

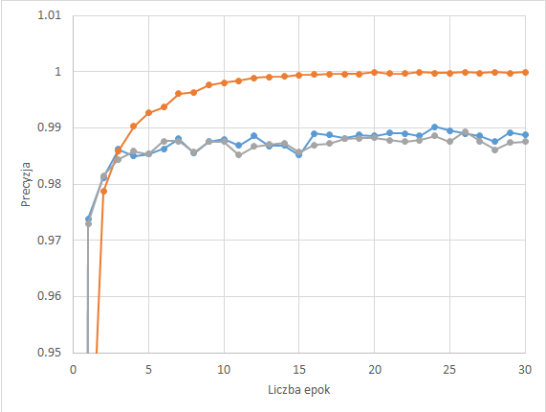
Weźmy pod lupę modele o ID równych 5 i 37. Pierwszy z nich na zbiorze testowym osiąga dokładność ok. 98% tymczasem drugi - ok. 21%. Przewagę modelu 5. potwierdza także wartość funkcji straty, która jest o 2 rzędy wielkości mniejsza od wartości odpowiadającej modelowi 37.

Zwróćmy uwagę na zależność dokładności od parametru *dropout*, który zapobiega przeuczeniu. W przypadku testowanych w tym projekcie modeli i danych, duże wartości tego parametru działają na niekorzyść skuteczności algorytmu. Wraz ze zwiększającym się *dropoutem* dokładności na wszystkich zbiorach mają tendencję malejącą, tymczasem funkcja straty - rosnącą. Może to wynikać z natury rozwiązywanego problemu (który należy do grupy prostych) i ilości danych dostarczonych do trenowania. Zbyt dużo „odciętych" połączeń powoduje utratę cennych informacji.

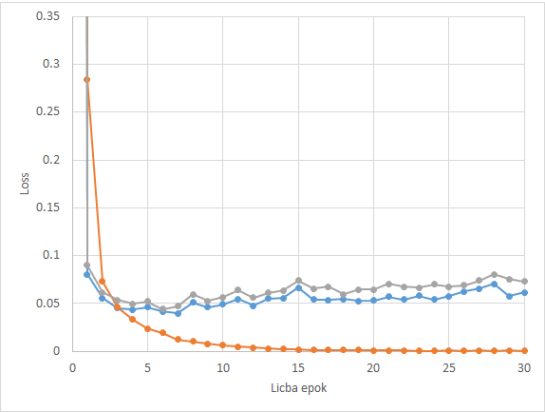
Nie bez znaczenia są też rozmiary zastosowanych filtrów. Zbyt małe nie pozwalają sieci na uchwycenie charakterystycznych dla danej cyfry kształtów, wzorców. Natomiast zbyt duże powodują utratę szczegółów znajdujących się na obrazie.

Na podstawie wyżej zaprezentowanych wyników, do dalszego rozważania, spośród grupy najlepszych (pod względem osiąganej dokładności na zbiorze testowym) w sposób losowy wybrano 6 modeli odpowiadających wierszom o numerach ID: 1, 5, 6, 27, 28 oraz 31. Na tabeli zostały one wyróżnione różowym tłem. Każdy z nich był nastepnie ponownie trenowany, tym razem przez 30 epok.

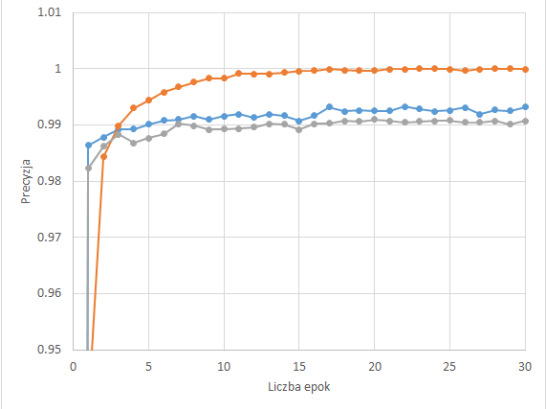
Na poniższych wykresach zobrazowano wyniki eksperymentu dla wybranych 6 modeli. Niebieską linią oznaczone są wartości dotyczące zbioru *testowego*, pomarańczową - zbioru *treningowego*, natomiast siwą - zbioru *walidacyjnego*. Są to wykresy odpowiednio dokładności (po lewej) i wartości funkcji straty (po prawej) na poszczególnych zbiorach w zależności od numeru epoki.



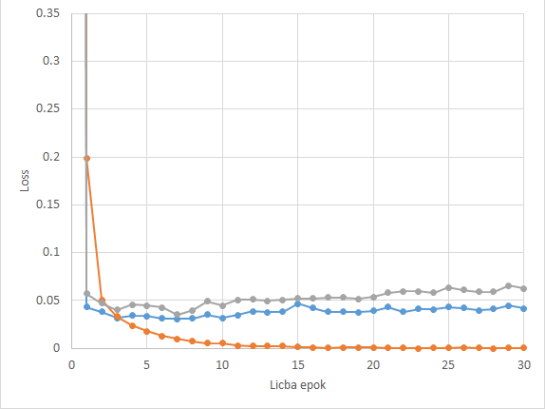
(a) Precyzja dla modelu o ID=1



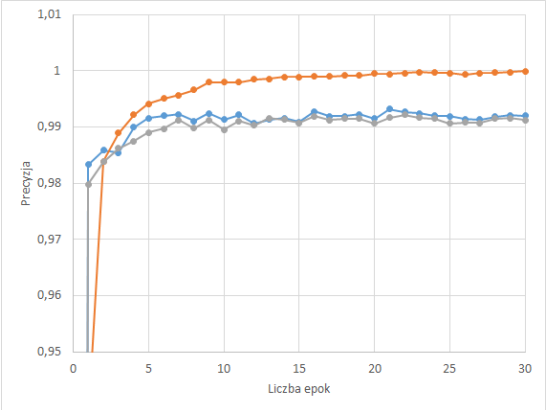
(b) Loss dla modelu o ID=1



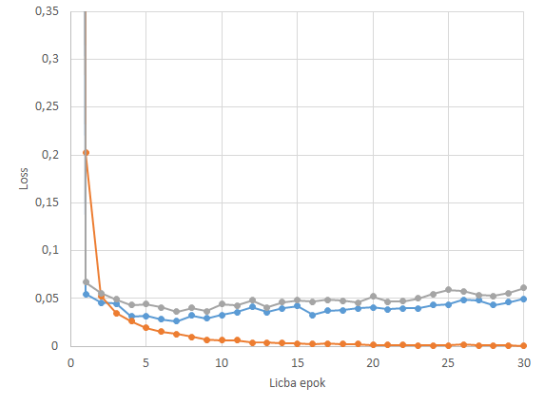
(c) Precyzja dla modelu o ID=5



(d) Loss dla modelu o ID=5

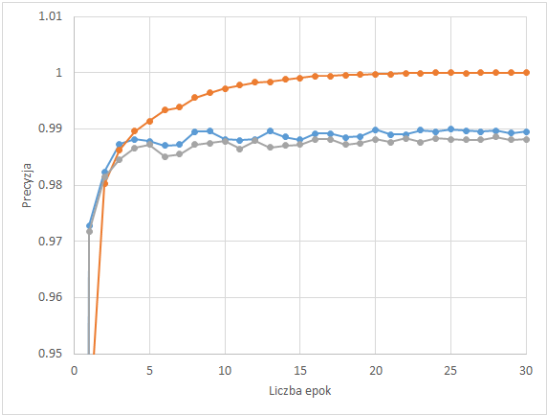


(e) Precyzja dla modelu o ID=6

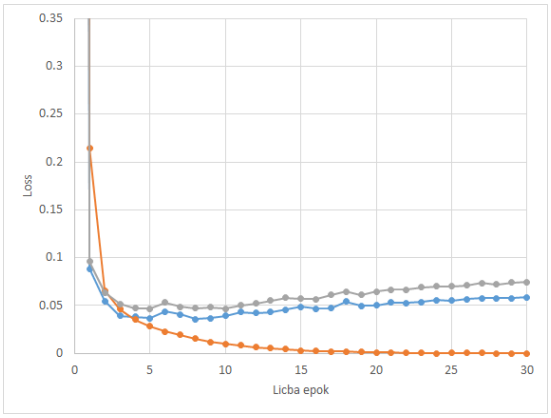


(f) Loss dla modelu o ID=6

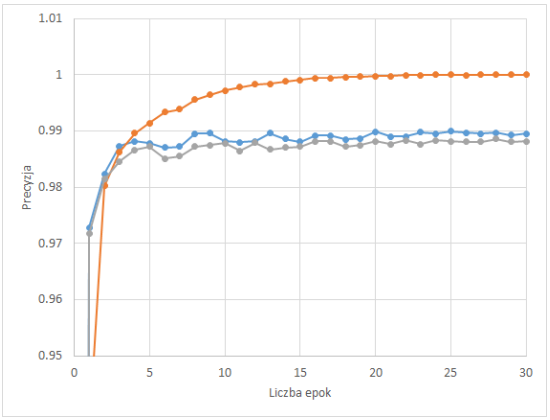
Rysunek 3



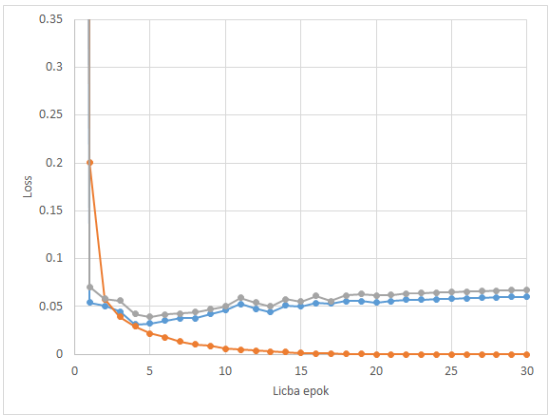
(a) Precyzja dla modelu o ID=27



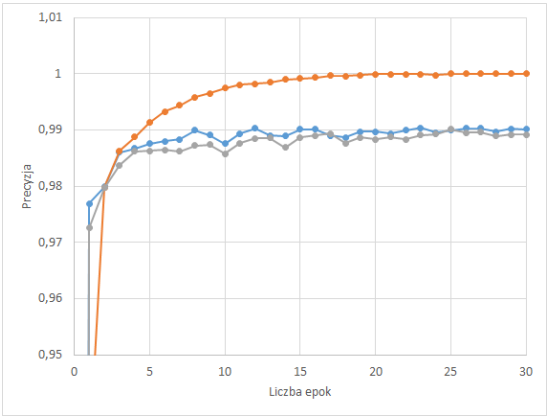
(b) Loss dla modelu oID=27



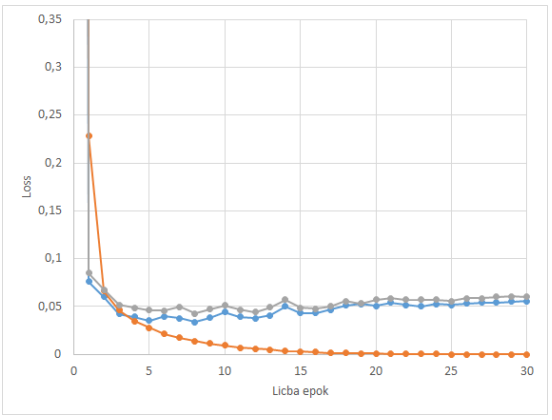
(c) Precyzja dla modelu o ID=28



(d) Loss dla modelu o ID=28



(e) Precyzja dla modelu o ID=31



(f) Loss dla modelu o ID=31

Rysunek 4

Wszystkie zaprezentowane powyżej wykresy posiadają wiele wspólnych cech. Przede wszystkim wzrostowi numeru epoki towarzyszy coraz większa dokładność. Co istotne, dotyczy to nie tylko zbioru treningowego, ale przede wszystkim testowego, co świadczy o tym, iż sieć poprawia swoje zdolności klasyfikacji danych, które są dla niej obce. Dla pierwszych 5 epok wzrost dokładności i spadek wartości funkcji straty jest gwałtowny, natomiast w kolejnych epokach stabilizuje się i oscyluje wokół pewnej granicy. Zjawisko to jest zgodne z naszymi przewidywaniami.

Możemy również zaobserwować, iż wszystkie rozważane tu modele sieci uzyskują precyzję w granicach 98%-99% juz po pierwszych 3-4 epokach. Jednakże po 5 epokach na

wykresach prezentujących wartość *loss* obserwujemy nieznaczny wzrost i oscylację tej wartości. Dotyczy to zbioru walidacyjnego i testowego. W zbiorze treningowym natomiast zjawisko to nie występuje. Może to świadczyć o powolnym „przeuczaniu się" sieci.

Model o ID równym 6 charakteryzuje się najmiejszym wzrostem *loss* na zbiorze wlidacyjnym i testowym przy jednoczesnym wysokim wskaźniku precyzji utrzymującym się powyżej 99%. Dlatego też uznaliśmy ten model za najlepszy z rozpatrywanych. Warto go użyć do rozwiązywania rozważanego problemu.

2.2 SVM

2.2.1 Opis algorytmu

Metoda maszyny wektorów nośnych jest (podobnie jak CNN) algorytmem opierającym się na uczeniu nadzorowanym. Intuicyjnie polega ona na wyznaczneniu hiperpłaszczyzny oddzielającej elementy zbioru należące do różnych klas. "Margineszozdzielający klasy powinien być możliwie jak największy.

- Parametr *c* - od jego wartości zależy kształt hiperpłaszczyzny oddzielającej kategorie. Czym mniejsza wartość, tym hiperpłaszczyna jest "gładsza", co jest pożądane w przypadku danych zaszumionych. Natomiast zwiększanie wartości *c* powoduje, efekt odwrotny - granica jest coraz bardziej dostosowana do danych treningowych.
- Parametr *gamma* - zależy od niego zasięg wpływu poszczególnych przykładów ze zbioru treningowego. Czym większa wartość *gamma*, tym mniejszy zasięg.

2.2.2 Implementacja

Zbiór danych został podzielony na 2 podzbiory:

- Zbiór treningowy - 60 000 obrazów;
- Zbiór testowy - 10 000 obrazów

Następnie uruchomiono algorytm dla różnych kombinacji parametrów *c* i *gamma* (łącznie 60 uruchomień). Wartości tych parametrów, wyniki dokładności klasyfikacji zbioru testowego oraz czas trwania obliczeń zostały zapisane w pliku *SVM_ALL.csv*.

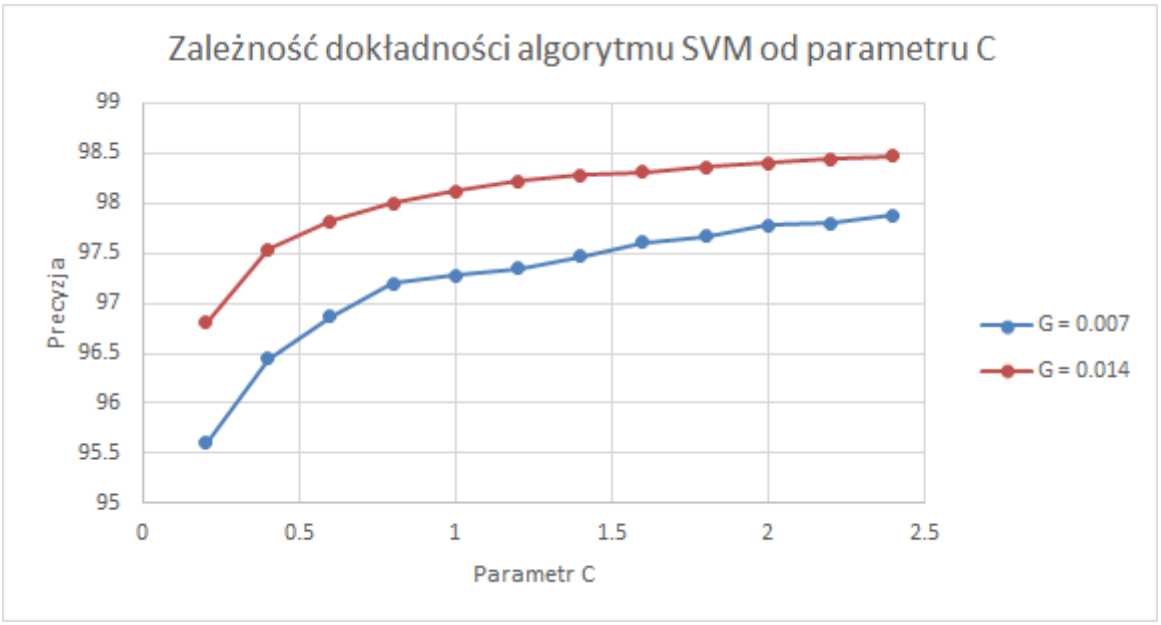
2.2.3 Wyniki i wnioski

W poniższej tabeli zostały przedstawione wyniki uyskane przy pomocy algorytmu SVM dla różnych parametrów *c* oraz *gamma(G)*. *Accuracy* oznacza dokładność wyrażoną w procentach. Podano również czas działania algorytmu (wyrażono go w sekundach).

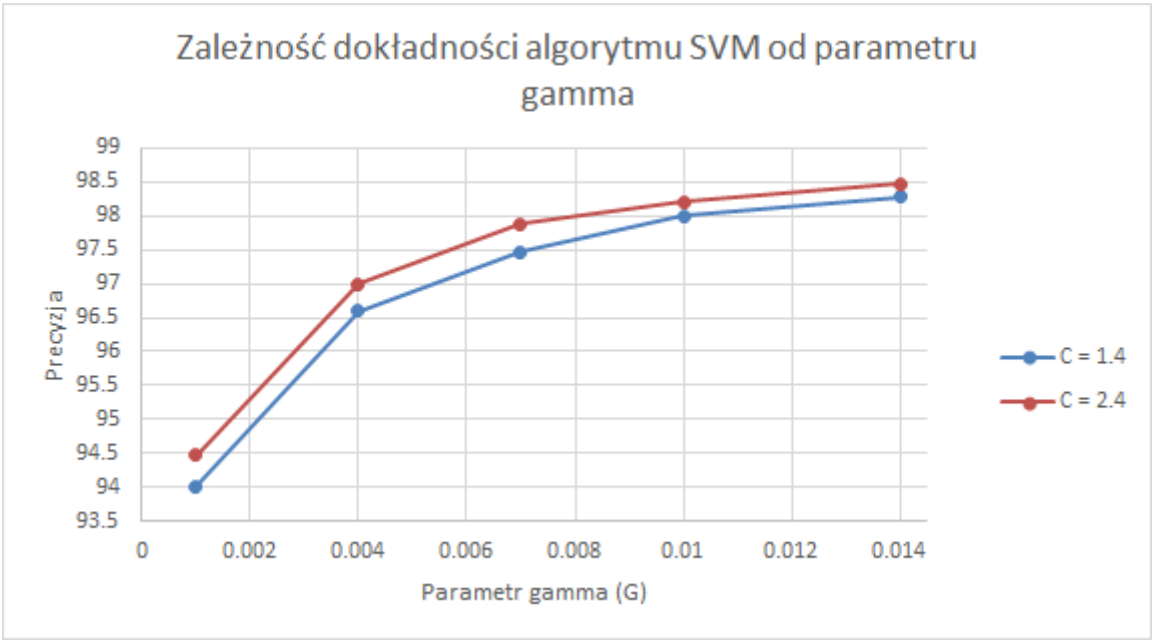
id	Accuracy	C	G	Time
0	98.4702	2.4	0.014	399.09
1	98.4402	2.2	0.014	399.92
2	98.4002	2	0.014	403.81
3	98.3602	1.8	0.014	420.13
4	98.3102	1.6	0.014	469.92
5	98.2802	1.4	0.014	428.06
6	98.2202	1.2	0.014	419.8
7	98.2002	2.4	0.01	367.78
8	98.1402	2.2	0.01	371.12
9	98.1202	1	0.014	434.91
10	98.0902	2	0.01	391.78
11	98.0802	1.8	0.01	397.74
12	98.0402	1.6	0.01	399.26
13	98.0002	0.8	0.014	452.9
14	98.0002	1.4	0.01	409.15
15	97.8802	2.4	0.007	363.8
16	97.8602	1.2	0.01	406.36
17	97.8202	0.6	0.014	490.02
18	97.8002	2.2	0.007	370.22
19	97.7802	2	0.007	385.56
20	97.7502	1	0.01	424.71
21	97.6702	1.8	0.007	404.6
22	97.6502	0.8	0.01	448.86
23	97.6102	1.6	0.007	405.71
24	97.5402	0.4	0.014	547.16
25	97.4703	1.4	0.007	419.03
26	97.4403	0.6	0.01	490.35
27	97.3503	1.2	0.007	420.19
28	97.2803	1	0.007	442.77
29	97.2003	0.8	0.007	472.52
30	97.0703	0.4	0.01	557.53
31	97.0203	2.2	0.004	409.15
32	96.9903	2	0.004	419.42
33	96.9903	2.4	0.004	400.9
34	96.9003	1.8	0.004	435.08
35	96.8703	0.6	0.007	515.98
36	96.8103	0.2	0.014	729.68
37	96.7503	1.6	0.004	460.83
38	96.5903	1.4	0.004	475.94
39	96.4504	0.4	0.007	594
40	96.4204	1.2	0.004	482.67
41	96.3504	0.2	0.01	779.82
42	96.2204	1	0.004	508.81
43	96.0304	0.8	0.004	544.31
44	95.7604	0.6	0.004	599.25
45	95.6004	0.2	0.007	936.95
46	95.2005	0.4	0.004	694.7
47	94.4706	2.4	0.001	593.03
48	94.4406	0.2	0.004	1131.65
49	94.3906	2.2	0.001	611.41
50	94.2506	2	0.001	648.36
51	94.2306	1.8	0.001	664.21
52	94.1306	1.6	0.001	696.26
53	94.0106	1.4	0.001	733.32
54	93.9506	1.2	0.001	744.02
55	93.7806	1	0.001	790.87
56	93.5406	0.8	0.001	859.15
57	93.2207	0.6	0.001	954.64
58	92.8807	0.4	0.001	1123.78
59	91.9808	0.2	0.001	1756.78

Rysunek 5: Wyniki algorytmu SVM dla różnych parametrów

Z powyższej tabeli możemy odczytać, iż maksymalny uzyskany przez nas wynik dla algorytmów SVM wynosi około 98%.



Rysunek 6: Zależność precyzji od parametru C dla *gamma=0.007 oraz gamma=0.014*



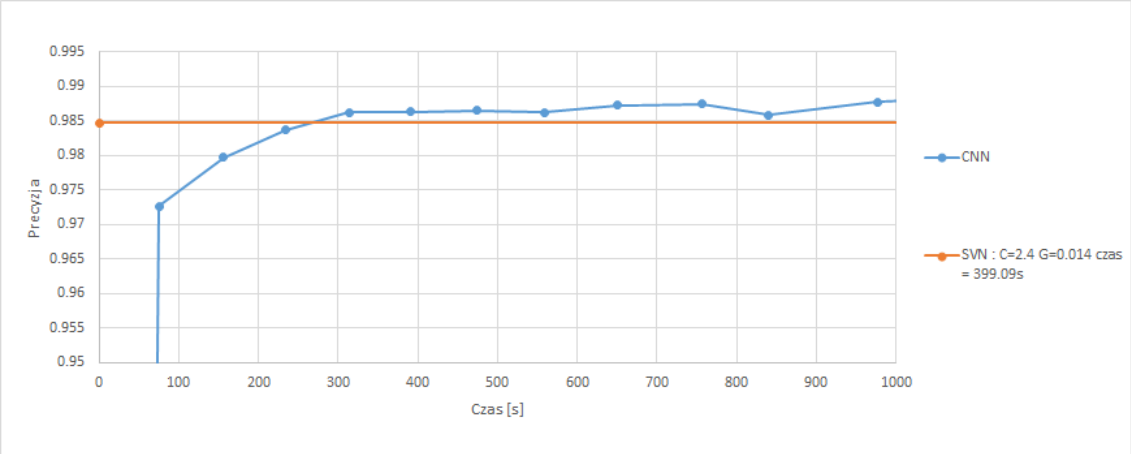
Rysunek 7: Zależność precyzji od parametru *gamma* dla *C=1.4 oraz C=2.4*

Powyższe wykresy (*Rysunek 6 oraz Rysunek 7*) pokazują, że precyzja rośnie wzraz ze wzrostem parametru C jak i parametru *gamma*. Być może dalsze zwiększanie parametrów C lub *gamma* poprawiłoby nieco uzyskaną dokładność klasyfikacji tej metody. Przypomnijmy, że parametr c odpowiada za dostosowanie „kształtu" hiperpłaszczyzny do danych treningowych. Zatem istnieni ryzyko, iż od pewnego momentu algorytm zacząłby wykazywać oznaki przeuczenia opierając swoje decyzje na danych będących szumami.

2.3 Porównanie metod

Podsumowując, zarówno CNN, jak i SVM uzyskują dokładność na poziomie co najmniej 98%. Nie można więc jedynie na tej podstawie wnioskować o wyższości jednej z tych metod nad drugą. Weźmy więc pod uwagę czas ich działania.

Poniższy rysunek przedstawia wykres zależności dokładności na zbiorze testowym od czasu uczenia modelu sieci CNN o ID 31 oraz linię wskazującą najlepszy uzyskany wynik za pomocą metody SVM. Wynik ten został uzyskany w czasie 399s.



Rysunek 8

Z powyższego wykresu możemy odczytać, że CNN osiąga wynik lepszy od SVM po czasie około 315s. A zatem jest o około 84s szybszy.

Można więc wnioskować, iż odpowiednio przygotowane konwolucyjne sieci neuronowe uzyskują lepsze wyniki dla rozpatrywanego problemu.