Rzeszów, 22.12.2021

SZTUCZNA INTELIGENCJA

# RAPORT Z LABORATORIÓW

"Jednokierunkowa wielowarstwowa sieć neuronowa typu MLP"

"Algorytm Support Vector Machines (SVM)"

Piotr Krawiec L1
Semestr: 5 2021/2022
Kierunek: III/FS0-DI
Numer indeksu: 164165
Prowadzący: Maciej Kusy

# Spis treści

# 1 Dane

Dane wykorzystane podczas pracy pochodzą ze zbioru OptDigits5. Składa się on z 62 cech i jednej listy klas, do których należy dana lista cech. Problem, który prezentują dane, jest problemem wieloklasowym, zatem wymagać on będzie dostosowania kodu prezentowanego na zajęciach.

# 2 Jednokierunkowa wielowarstwowa sieć neuronowa typu MLP

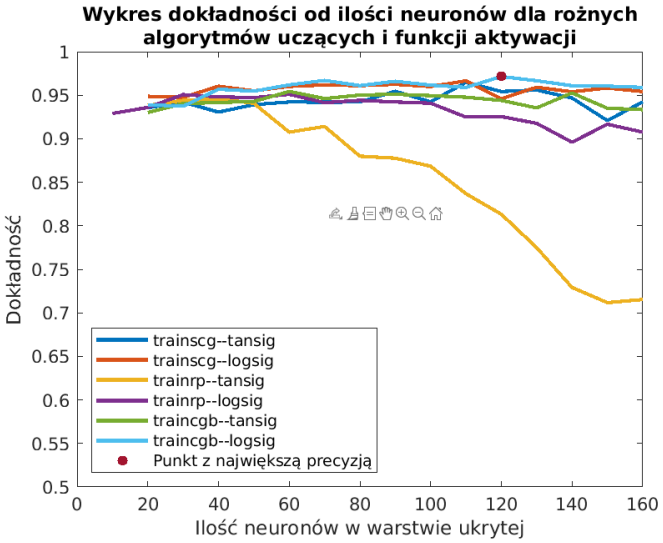
## 2.1 Sieć z jedną warstwą ukrytą

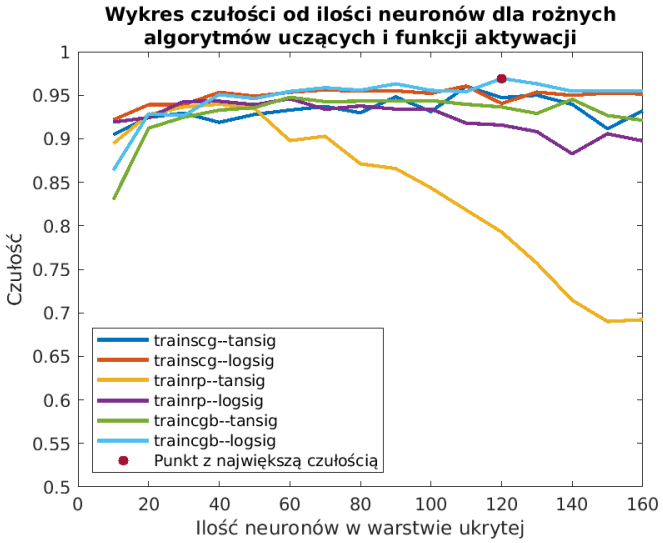
Ze względu na ilość danych określiłem przedział neuronów, które będę przeszukiwał na 10 do 190 neuronów. Trenowanie sieci odbyło się z wykorzystaniem GPU, jednak nie wszystkie funkcje trenowania sieci oraz aktywacji neuronów są dostępne. Zatem do trenowania wykorzystałem następujące funkcje: *trainscg*, *trainro* i *traincgb* oraz następujące funkcje aktywacji: *tansig* i *logsig*. Wykorzystanie GPU pozwoliło skrócić czas trenowania do około 10 minut.

Poniżej znajduje się tabela, która zestawia informacje o dokładności, czułości sieci i odchyleniach tych wartości dla każdej z sieci. Najlepszy wynik osiągnęła sieć **traincgb-logsig-120** z dokładnością **0.971** i czułością **0.969**.

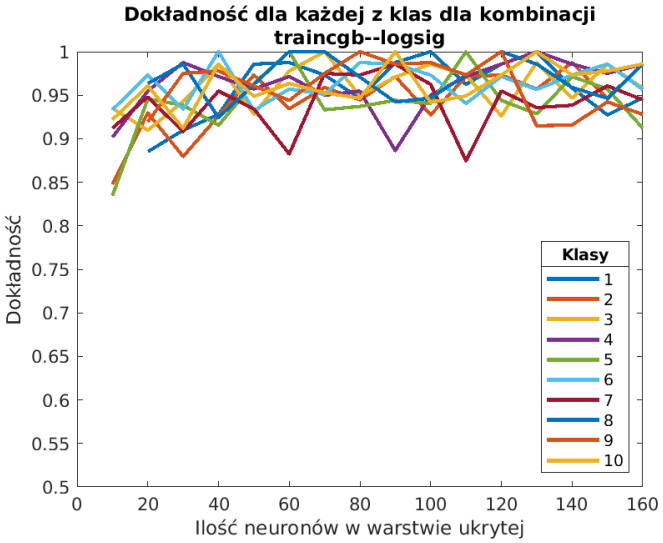
| Network | acc | std(acc) | recall | std(recall) |
|---|---|---|---|---|
| traincgb-logsig-120 | 0.971 | 0.0488 | 0.969 | 0.0551 |
| traincgb-logsig-130 | 0.967 | 0.0582 | 0.963 | 0.0652 |
| traincgb-logsig-90 | 0.966 | 0.0572 | 0.963 | 0.0648 |
| trainscg-logsig-110 | 0.966 | 0.0656 | 0.96 | 0.073 |
| trainscg-tansig-110 | 0.965 | 0.0611 | 0.96 | 0.0683 |
| traincgb-logsig-70 | 0.967 | 0.0679 | 0.959 | 0.0754 |
| trainscg-logsig-70 | 0.962 | 0.0702 | 0.956 | 0.0775 |
| traincgb-logsig-80 | 0.961 | 0.0609 | 0.956 | 0.0782 |
| traincgb-logsig-100 | 0.962 | 0.0608 | 0.955 | 0.0817 |
| trainscg-logsig-90 | 0.962 | 0.0667 | 0.955 | 0.0746 |
| trainscg-logsig-80 | 0.961 | 0.0681 | 0.955 | 0.0807 |
| traincgb-logsig-140 | 0.961 | 0.0687 | 0.955 | 0.0781 |
| traincgb-logsig-150 | 0.96 | 0.0682 | 0.955 | 0.0897 |
| traincgb-logsig-160 | 0.959 | 0.0711 | 0.955 | 0.0804 |
| traincgb-logsig-60 | 0.962 | 0.0613 | 0.954 | 0.0654 |
| traincgb-logsig-110 | 0.959 | 0.0661 | 0.954 | 0.0794 |
| trainscg-logsig-130 | 0.959 | 0.0661 | 0.954 | 0.0793 |
| trainscg-logsig-40 | 0.96 | 0.07 | 0.953 | 0.0861 |
| trainscg-logsig-60 | 0.96 | 0.0669 | 0.953 | 0.0822 |
| trainscg-logsig-150 | 0.958 | 0.0722 | 0.953 | 0.0822 |

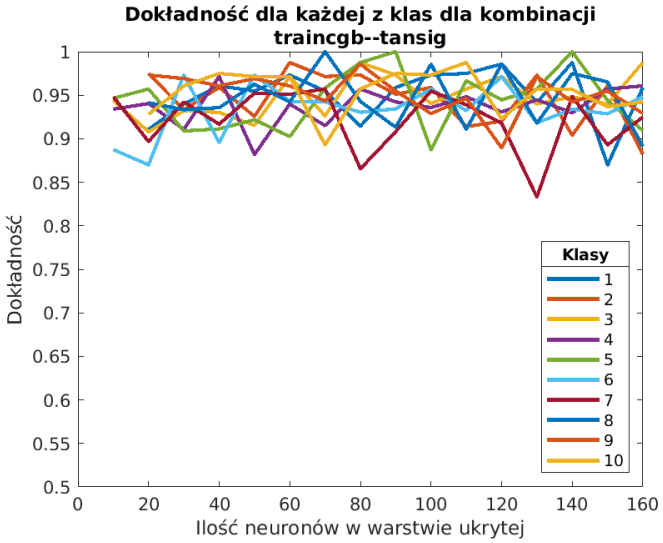
Tabela 1: Podsumowanie najlepszych sieci z jedną warstwą

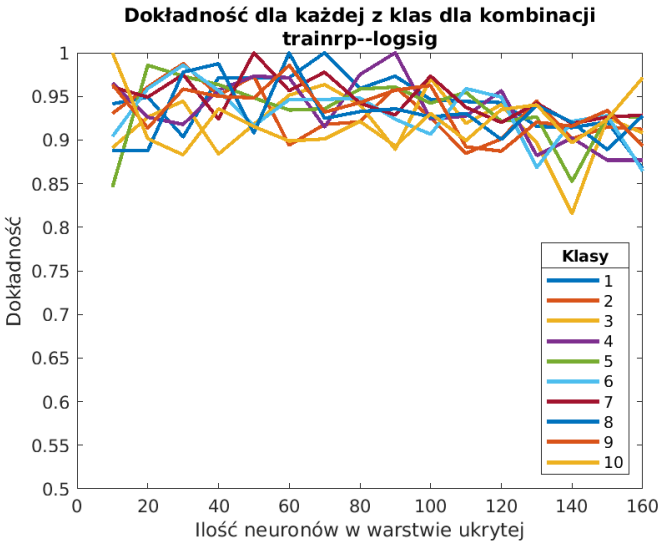


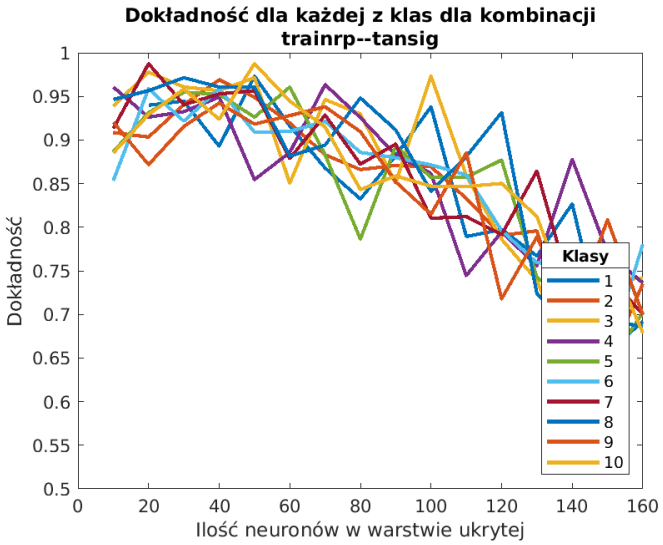



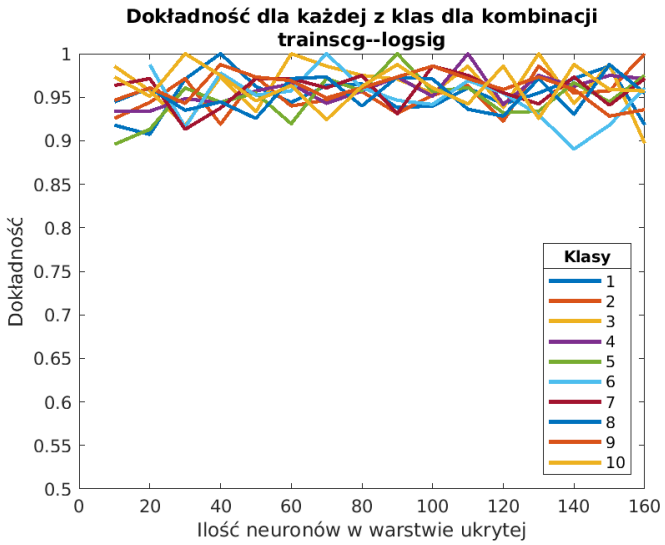

### 2.1.1 Wykresy precyzji dla każdej z klas

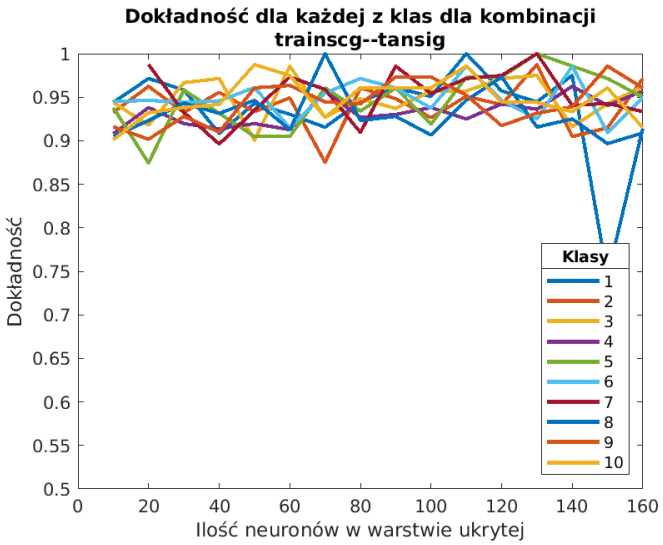


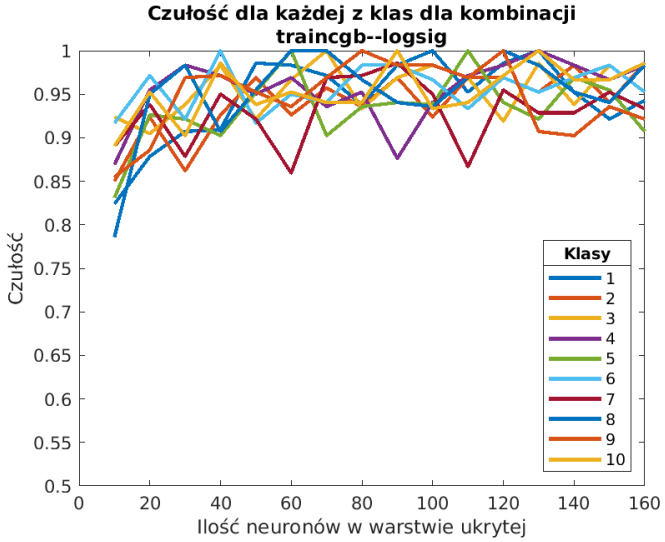



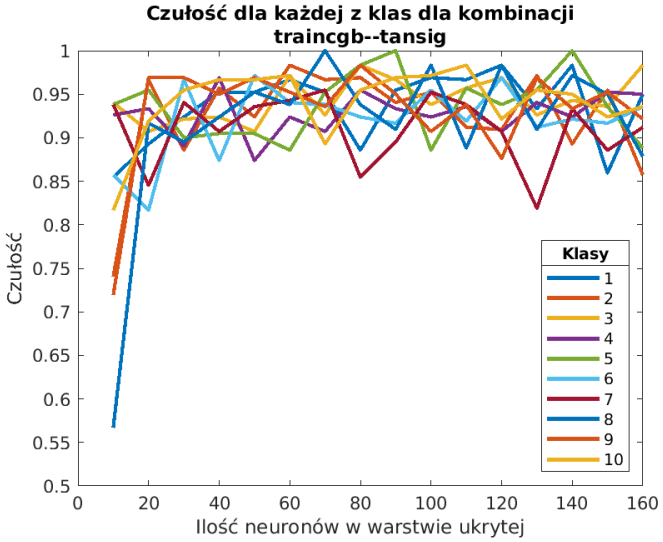



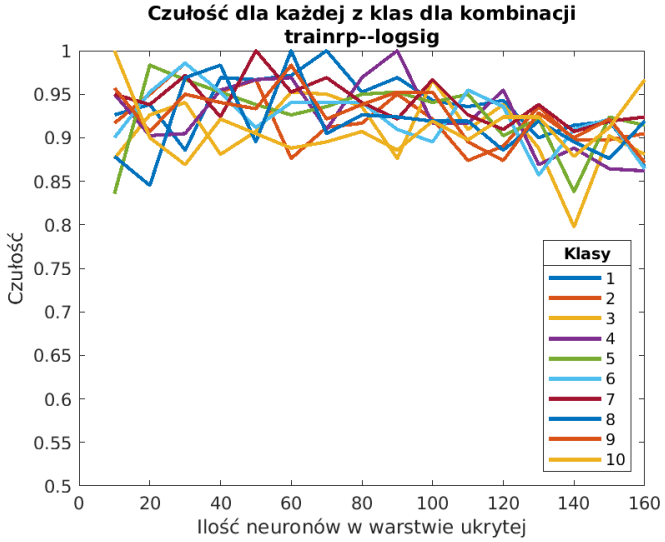


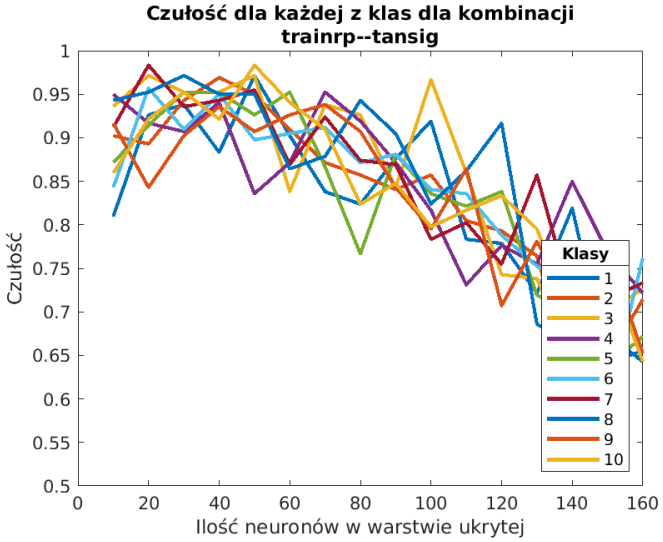



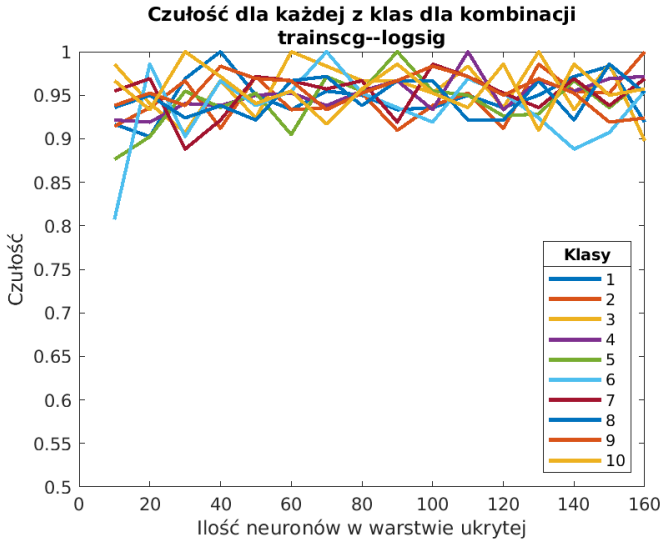

### 2.1.2 Wykresy czułości dla każdej z klas

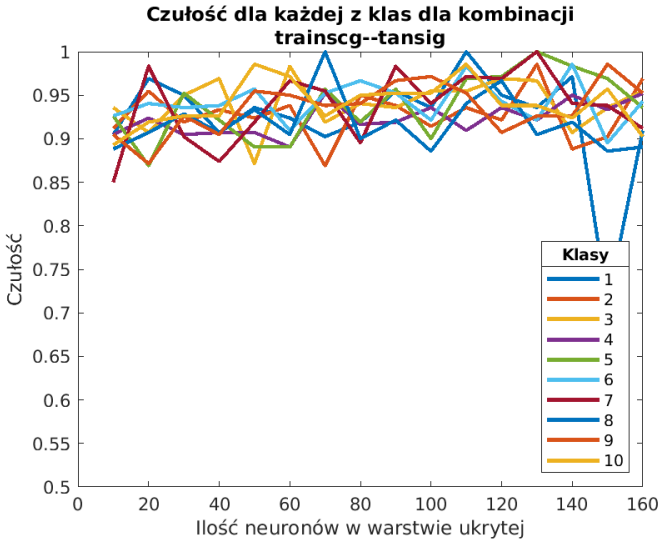





## 2.2 Sieć z dwiema warstwami ukrytymi

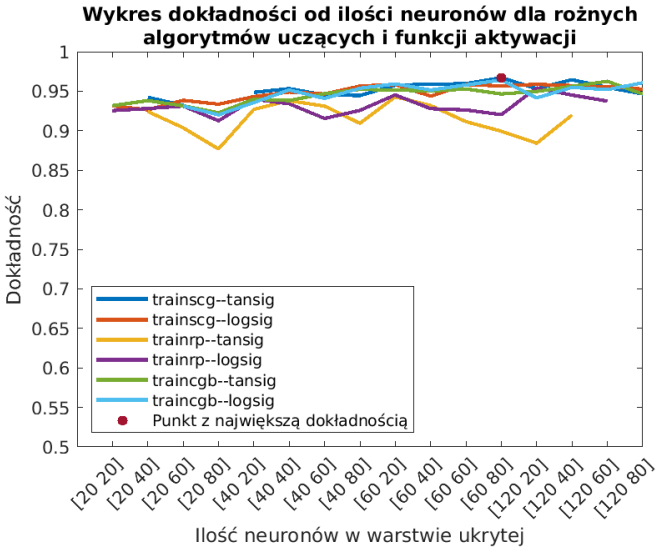
Sieci te także były trenowane na GPU, na tych samych funkcjach trenowania i aktywacji co sieć z jedną ukrytą warstwą. Natomiast ilość neuronów:

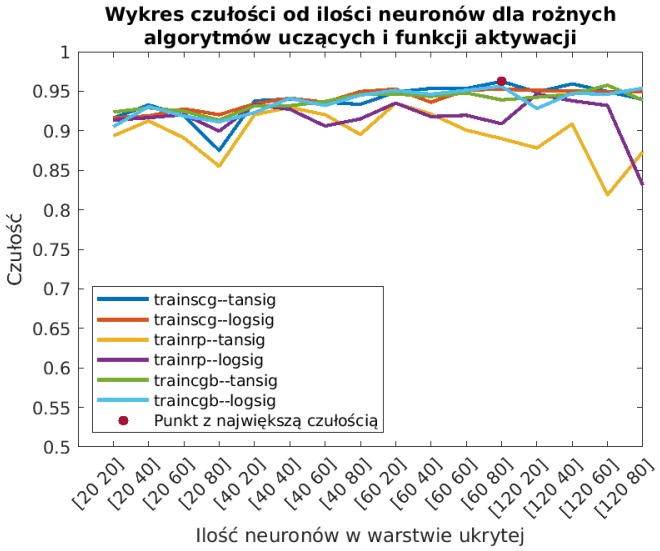
1. W pierwszej warstwie wynosi $20, 40, 60, 120$
2. A w drugiej $20, 40, 60, 80$

Czas trenowania wszystkich sieci wyniósł około 20 minut. Najlepsze wyniki osiągnęła sieć **trainscg-tansig-[60 80]** z dokładnością **0.967** i czułością **0.962**.

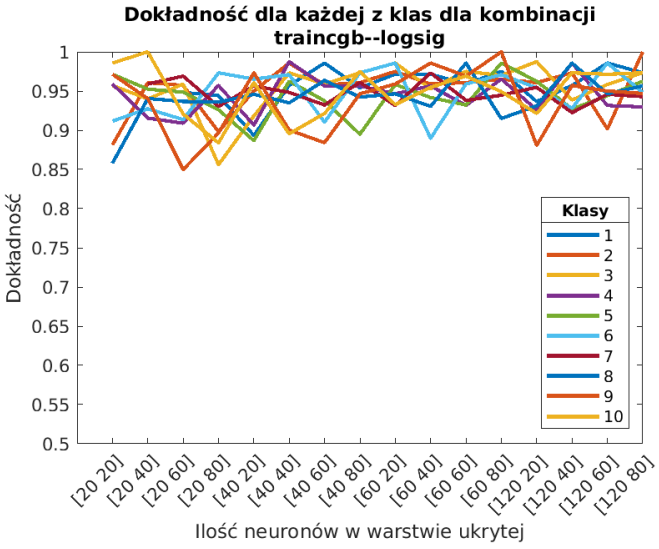
| Network | acc | std(acc) | recall | std(recall) |
|---|---|---|---|---|
| trainscg-tansig-[60 80] | 0.967 | 0.0616 | 0.962 | 0.0718 |
| trainscg-tansig-[120 40] | 0.964 | 0.0615 | 0.959 | 0.0801 |
| traincgb-tansig-[120 60] | 0.963 | 0.064 | 0.958 | 0.0749 |
| traincgb-logsig-[60 80] | 0.964 | 0.0686 | 0.956 | 0.0735 |
| traincgb-logsig-[120 80] | 0.961 | 0.0671 | 0.954 | 0.0742 |
| trainscg-tansig-[60 40] | 0.959 | 0.0683 | 0.954 | 0.0762 |
| trainscg-tansig-[60 60] | 0.959 | 0.0733 | 0.953 | 0.0793 |
| trainscg-logsig-[60 20] | 0.958 | 0.0716 | 0.953 | 0.0792 |
| trainscg-logsig-[60 80] | 0.957 | 0.067 | 0.953 | 0.0796 |
| trainscg-logsig-[120 20] | 0.959 | 0.0789 | 0.951 | 0.084 |
| traincgb-logsig-[60 20] | 0.959 | 0.0737 | 0.951 | 0.0808 |
| trainscg-logsig-[60 60] | 0.958 | 0.0774 | 0.951 | 0.0879 |
| traincgb-logsig-[60 60] | 0.958 | 0.0705 | 0.951 | 0.0839 |
| trainscg-logsig-[120 40] | 0.957 | 0.0734 | 0.95 | 0.0776 |
| trainscg-logsig-[40 80] | 0.956 | 0.0736 | 0.95 | 0.0962 |
| trainscg-logsig-[120 60] | 0.956 | 0.0692 | 0.95 | 0.073 |
| trainscg-tansig-[120 60] | 0.955 | 0.0801 | 0.95 | 0.0893 |
| trainscg-tansig-[60 20] | 0.958 | 0.0666 | 0.949 | 0.0885 |
| trainscg-logsig-[120 80] | 0.953 | 0.0706 | 0.949 | 0.0845 |
| trainscg-tansig-[120 20] | 0.953 | 0.0674 | 0.949 | 0.0731 |

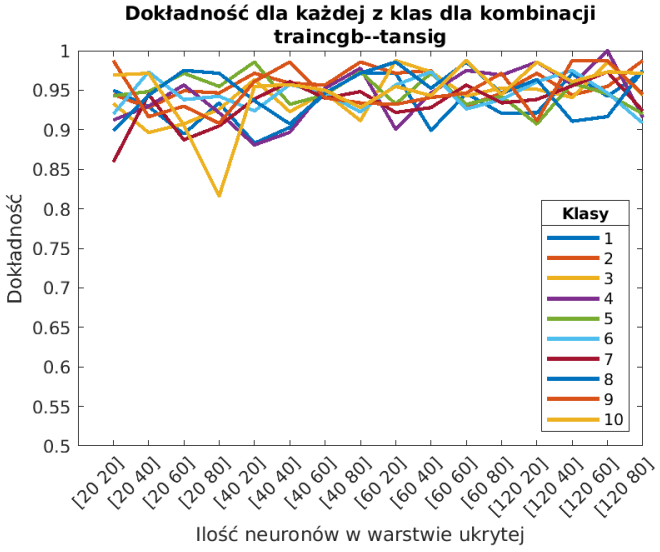
Tabela 2: Podsumowanie najlepszych sieci z dwiema warstwami

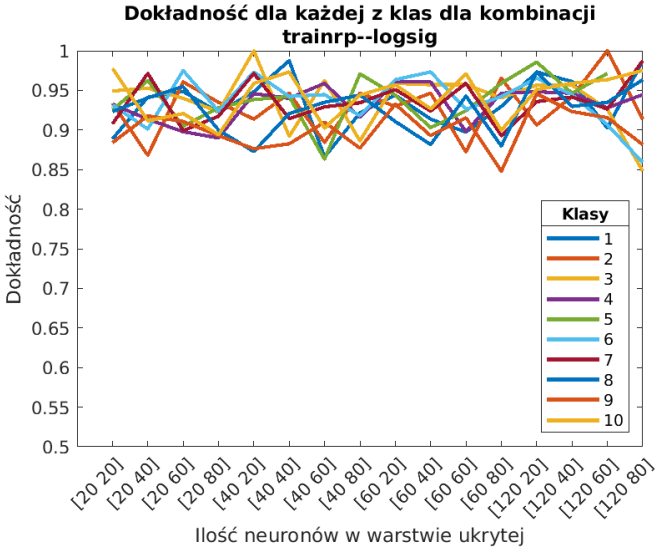


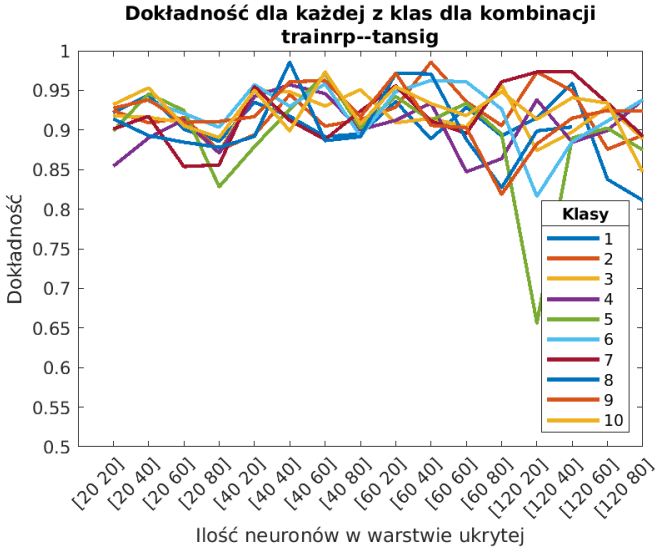



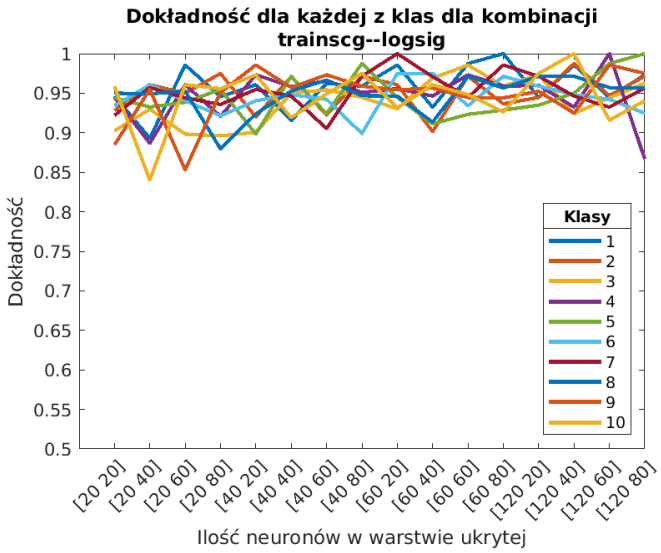

### 2.2.1 Wykresy precyzji dla każdej z klas

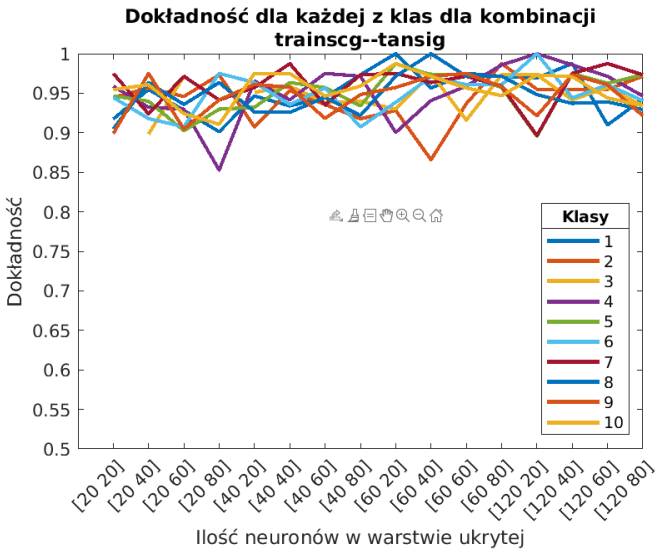


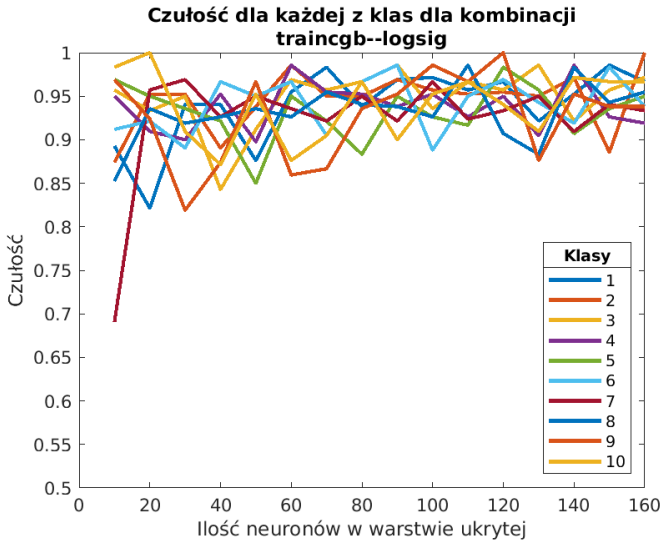



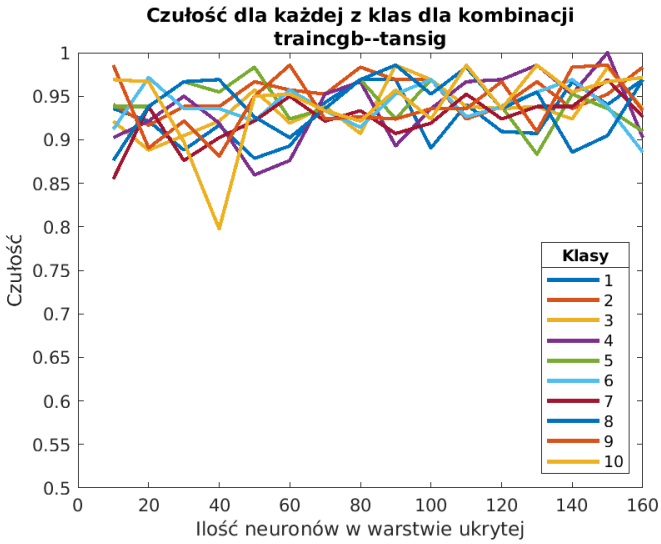



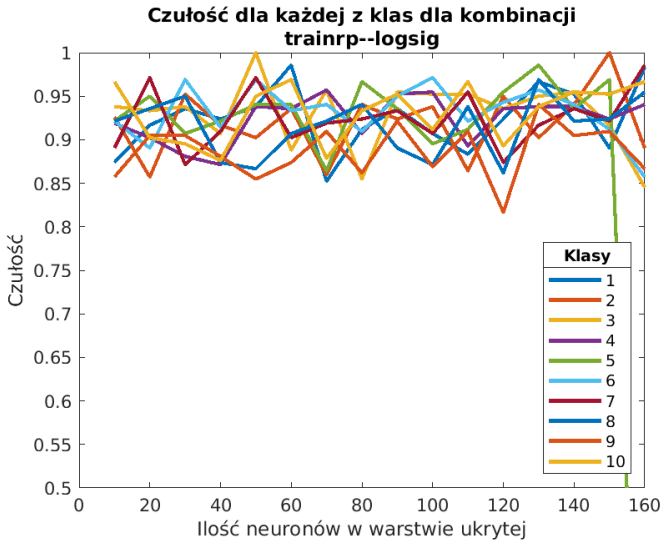


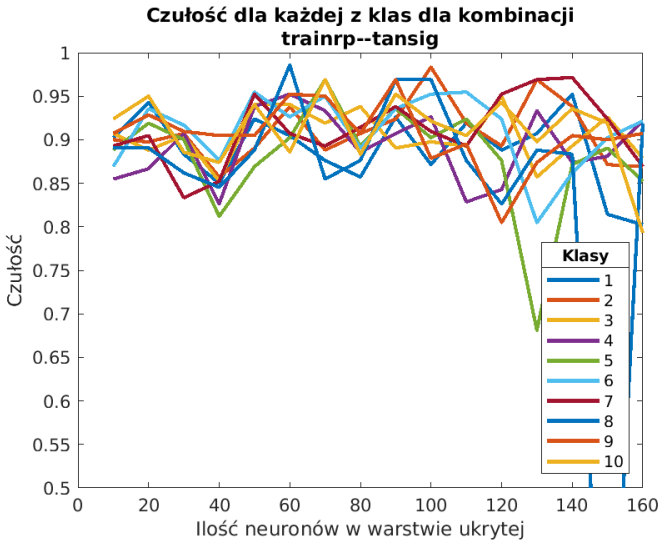



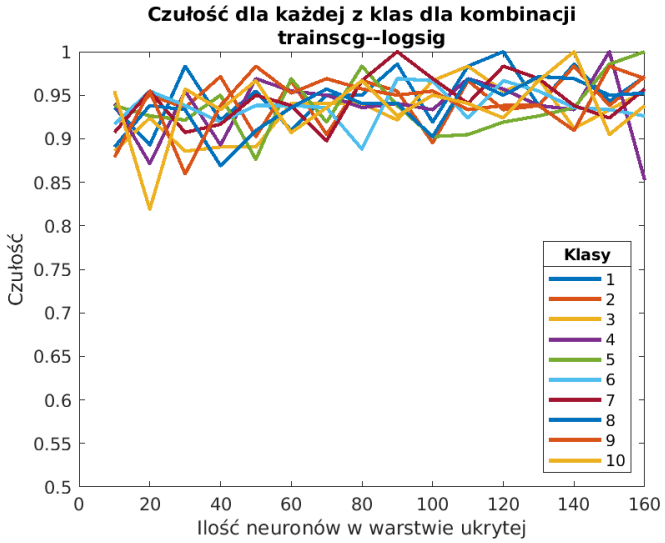

### 2.2.2 Wykresy czułości dla każdej z klas

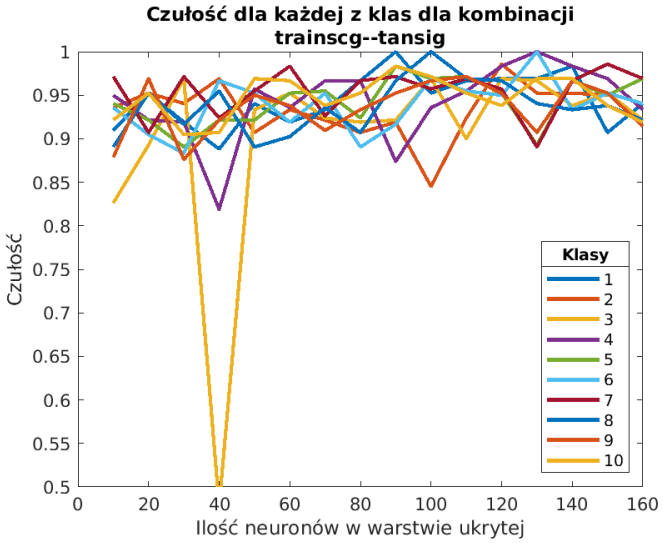





## 2.3 Podsumowanie sieci MLP

Obie sieci osiągnęły wyniki podobnego rzędu. Natomiast zastosowanie dwóch warstw ukrytych poprawiło wynik dla najgorszej kombinacji tj. **trainrp–tansig**. Porównując najlepsze wyniki z obu sieci, **najlepsze wyniki osiągnęła sieć jednowarstwowa**, z algorytmem **traincgb**, funkcją aktywacji **logsig** i **120** neuronami w warstwie ukrytej.

# 3 Algorytm SVM

Algorytm SVM został przetestowany dla dwóch funkcji jądra: gaussian i polynomial. Obliczeń dokonałem, wybierając kolejno dane z jednej klasy i oznaczając je "1", a pozostałe "0". Dla tak stworzonych modeli (per każda klasa) obliczyłem dokładność, czułość oraz F1. Wyniki te zostały uśrednione dla każdej z grup klasyfikatorów i umieszczone w tabeli poniżej. Zera w Tabeli 3 wynikają ze sposobu obliczania danej metryki, pojawia się tam dzielenie przez 0. Dzieje się tak, ponieważ w macierzy pomyłek w polach TP i FP pojawiają się 0 (wszystkie próbki określane są jako 0). Parametry w Tabeli 3 kodowane są w następujący sposób:

1. Dla kernela gaussian: **kernel-sigma-C**
2. Dla kernela polynomial: **kernel-d-C**

Jako najlepszy model wybrałem SVM z parametrami **kernel polynomial, d=2 oraz C=0.1**. Nie był on najlepszy w każdej z meryk, ale był najlepszy w największej liczbie metryk, a w pozostałych przegrywał dopiero na 3-4 miejscu po przecinku.

| SVM | acc | std(acc) | precision | recall | F1 score |
|---|---|---|---|---|---|
| polynomial-2-0.1 | 0.9923 | 0.0097 | 0.9823 | **0.9475** | **0.9618** |
| polynomial-2-10 | **0.9925** | 0.0091 | 0.9847 | 0.9454 | 0.9617 |
| polynomial-2-1000 | 0.9914 | 0.0095 | 0.9776 | 0.9447 | 0.9584 |
| polynomial-3-1000 | 0.9906 | 0.011 | 0.9782 | 0.9342 | 0.9522 |
| gaussian-20-1000 | 0.9884 | 0.0112 | 0.9667 | 0.9318 | 0.9448 |
| gaussian-20-10 | 0.9891 | 0.0113 | 0.9723 | 0.9298 | 0.9463 |
| polynomial-3-10 | 0.9894 | 0.0131 | 0.9779 | 0.921 | 0.9442 |
| polynomial-3-0.1 | 0.9895 | 0.0146 | 0.9686 | 0.9193 | 0.9398 |
| gaussian-5-1000 | 0.9858 | 0.0136 | 0.9961 | 0.8642 | 0.9189 |
| gaussian-5-10 | 0.9861 | 0.0125 | **0.9969** | 0.8626 | 0.9195 |
| gaussian-5-0.1 | 0.911 | 0.0105 | 0.27 | 0.1102 | 0.1456 |
| gaussian-0.5-0.1 | 0.9001 | 0.0069 | 0 | 0 | 0 |
| gaussian-20-0.1 | 0.9001 | **0.007** | 0 | 0 | 0 |
| gaussian-0.5-10 | 0.9 | 0.0071 | 0 | 0 | 0 |
| gaussian-0.5-1000 | 0.9 | 0.0071 | 0 | 0 | 0 |

Tabela 3: Wyniki działania algorytmów SVM z różnymi parametrami

## 3.1 Zmiana parametru Standardize

Dla najlepszego modelu dokonałem trenowania ponownie zmieniając parametr Standardize. Tabele Tabela 4 oraz Tabela 5 przedstawiają wyniki. Znajdują się w nich metryki obliczone dla każdej z klas oraz wyniki uśrednione. Lepszy okazał sie model z parametrem **standardize=true**. Jednak zmiana tego parametru nie zmieniła tych wyników znacząco, w większości różnią się na 3 miejscu po przecinku.

| Klasa | acc | std(acc) | prec | recall | F1 |
|---|---|---|---|---|---|
| 1 | 0.9953 | 0.0105 | 1 | 0.9524 | 0.9723 |
| 2 | 0.9921 | 0.0083 | 0.9528 | 0.9833 | 0.9658 |
| 3 | 0.9968 | 0.01 | 1 | 0.9667 | 0.98 |
| 4 | 0.9907 | 0.0108 | 0.9889 | 0.9262 | 0.953 |
| 5 | 0.9953 | 0.0105 | 0.9875 | 0.9714 | 0.9767 |
| 6 | 0.9921 | 0.0083 | 0.9875 | 0.9357 | 0.9584 |
| 7 | 1 | 0 | 1 | 1 | 1 |
| 8 | 0.9891 | 0.0105 | 0.9639 | 0.9429 | 0.9487 |
| 9 | 0.9843 | 0.0146 | 0.9532 | 0.9107 | 0.9273 |
| 10 | 0.9859 | 0.0173 | 0.9597 | 0.9113 | 0.9308 |
| **Średnia** | **0.9922** | **0.0101** | **0.9793** | **0.9501** | **0.9613** |

Tabela 4: Tabela dla **polynomial-2-0.1** ze **standardize=true**

Piotr Krawiec

| Klasa | acc | std(acc) | prec | recall | F1 |
|---|---|---|---|---|---|
| 1 | 0.9953 | 0.0075 | 1 | 0.9524 | 0.9741 |
| 2 | 0.989 | 0.0148 | 0.9635 | 0.9417 | 0.9501 |
| 3 | 0.9938 | 0.008 | 0.9875 | 0.9548 | 0.9689 |
| 4 | 0.9906 | 0.011 | 0.9732 | 0.9429 | 0.956 |
| 5 | 0.9938 | 0.008 | 0.9764 | 0.9714 | 0.9721 |
| 6 | 0.9922 | 0.0082 | 0.9764 | 0.9524 | 0.9616 |
| 7 | 0.9985 | 0.0049 | 0.9889 | 1 | 0.9941 |
| 8 | 0.9906 | 0.0133 | 0.9818 | 0.9357 | 0.9532 |
| 9 | 0.9828 | 0.0188 | 0.9625 | 0.8905 | 0.9123 |
| 10 | 0.978 | 0.0199 | 0.9257 | 0.9089 | 0.9118 |
| **Średnia** | **0.9904** | **0.0115** | **0.9736** | **0.9451** | **0.9554** |

Tabela 5: Tabela dla **polynomial-2-0.1** ze **standardize=false**