Projekt – sieci neuronowe	Data złożenia projektu: 10.05.2023
Numer grupy projektowej: 1	Imię i nazwisko I: Szymon Talar Imię i nazwisko II: Michał Golec

Klasyfikacja grzybów pod kątem jadalności

1. Opis problemu i danych

Punktem wyjścia dla rozpoczęcia pracy nad projektem sieci neuronowych było znalezienie odpowiedniej bazy danych. W naszym przypadku jest to baza zawierająca informacje na temat różnych parametrów grzybów tj. rodzaju nóżki, koloru kapelusza (łącznie 22 parametry) na podstawie których próbowaliśmy określić, czy dany grzyb jest jadalny czy też nie. Staraliśmy się wybrać w miarę możliwości bazę danych, zapewniającą:

a) Odpowiednią ilość rekordów, pozwalającą na wykonanie dokładniejszych predykcji (na zajęciach podano liczbę min. 10 000)
b) Zawierającą dość zróżnicowane dane, tj. podobną liczbę grzybów jadalnych i niejadalnych, a także rekordy różniące się pomiędzy sobą znacząco pod względem poszczególnych parametrów tak, aby algorytm uczący miał więcej danych do pracy.

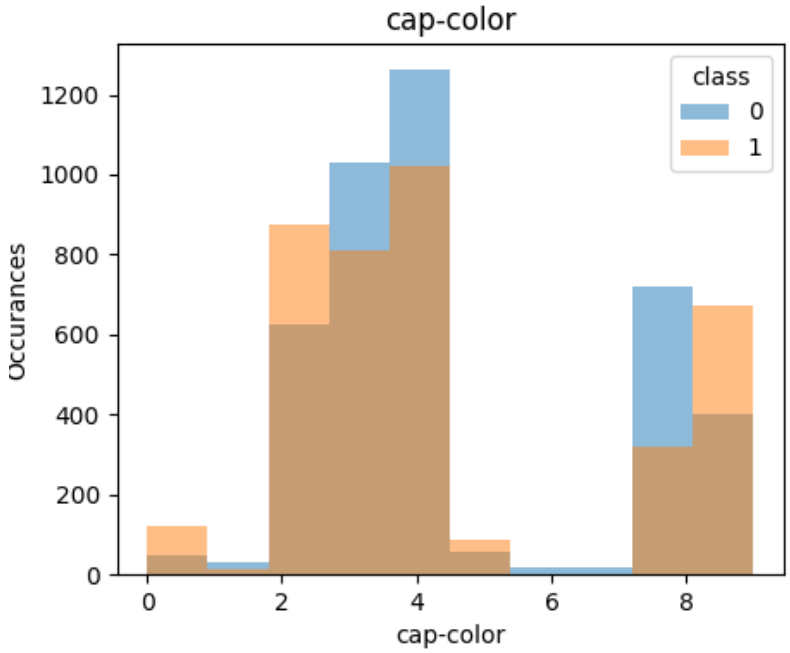
Z powyższych celów pierwszy udało się spełnić połowicznie, ponieważ o ile liczba rekordów na której pracujemy jest minimalnie niższa, powinna wystarczyć dla tego zadania, natomiast drugi spełniliśmy w stopniu satysfakcjonującym nas.

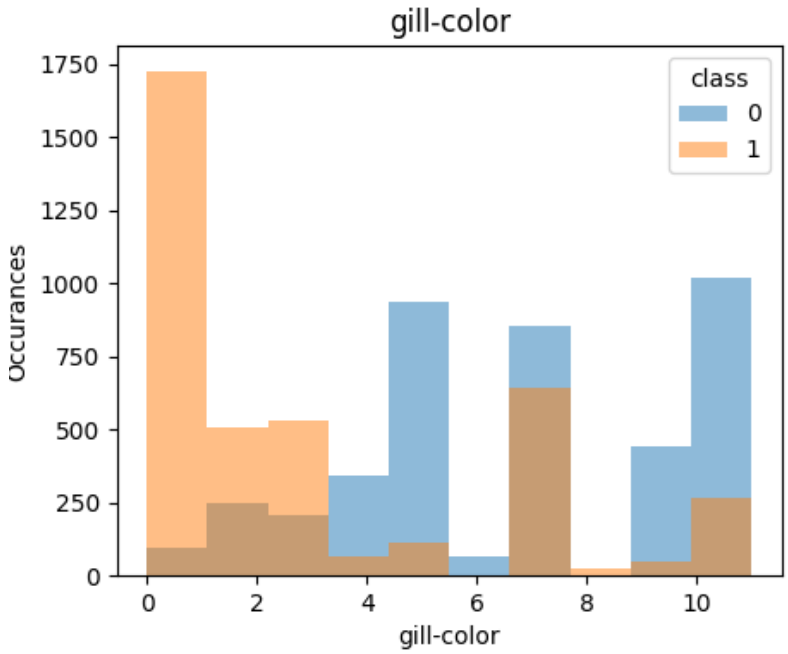
```
class
b'e'    4208
b'p'    3916
Name: count, dtype: int64
```

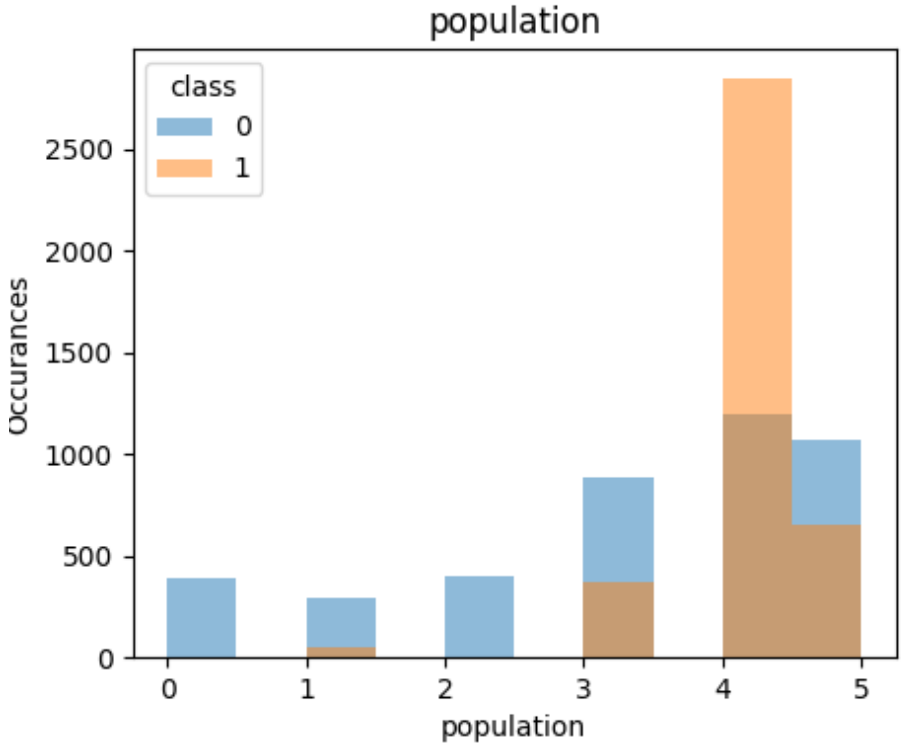
```
       cap-shape cap-surface cap-color  ... population habitat class
count       8124        8124      8124  ...       8124    8124  8124
unique         6           4        10  ...          6       7     2
top          b'x'        b'y'      b'n'  ...       b'v'    b'd'  b'e'
freq        3656        3244      2284  ...       4040    3148  4208

[4 rows x 23 columns]
```

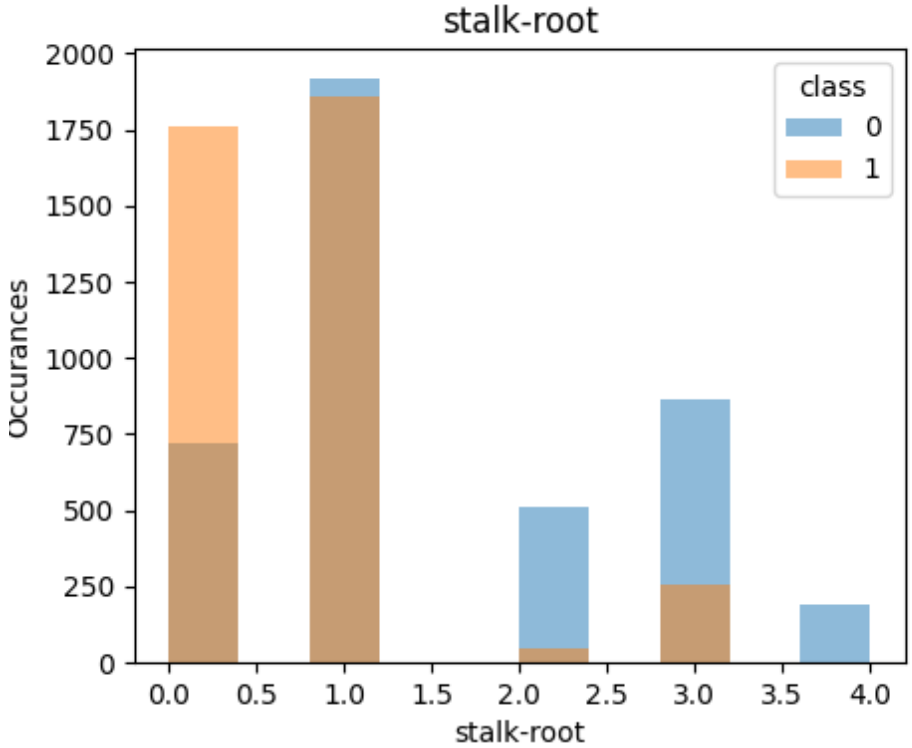
Na pierwszym screenie pokazane są ilości grzybów b’e’(edible) i b’p’(poisonous). Na drugim zaś Ilości kategorii wraz z najczęściej się pojawiającymi.

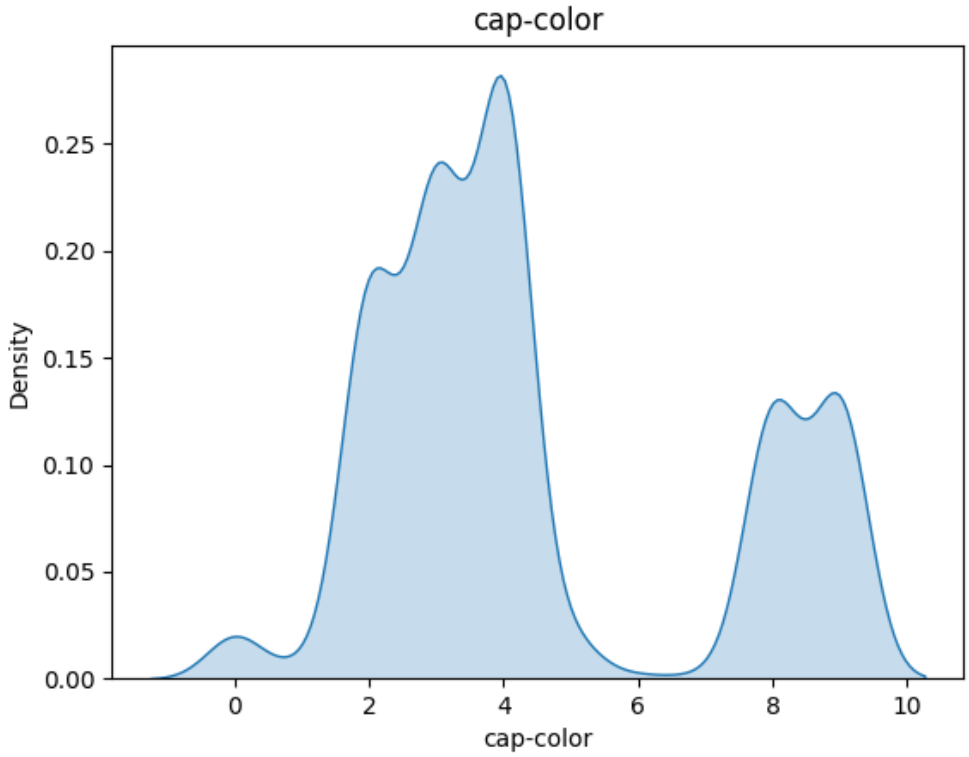
Dla takich danych wejściowych mogliśmy również wykonać szereg wykresów pokazujące różne wartości statystyczne. Dla przykładu sporządziłem wykresy dla 4 parametrów (Wszystkie 22 zajęłyby zbyt dużo miejsca w sprawozdanie) pokazujące ilość okazów jadalnych i niejadalnych posiadających określoną cechę oraz gęstość występowania poszczególnych opcji wśród wybranych cech.

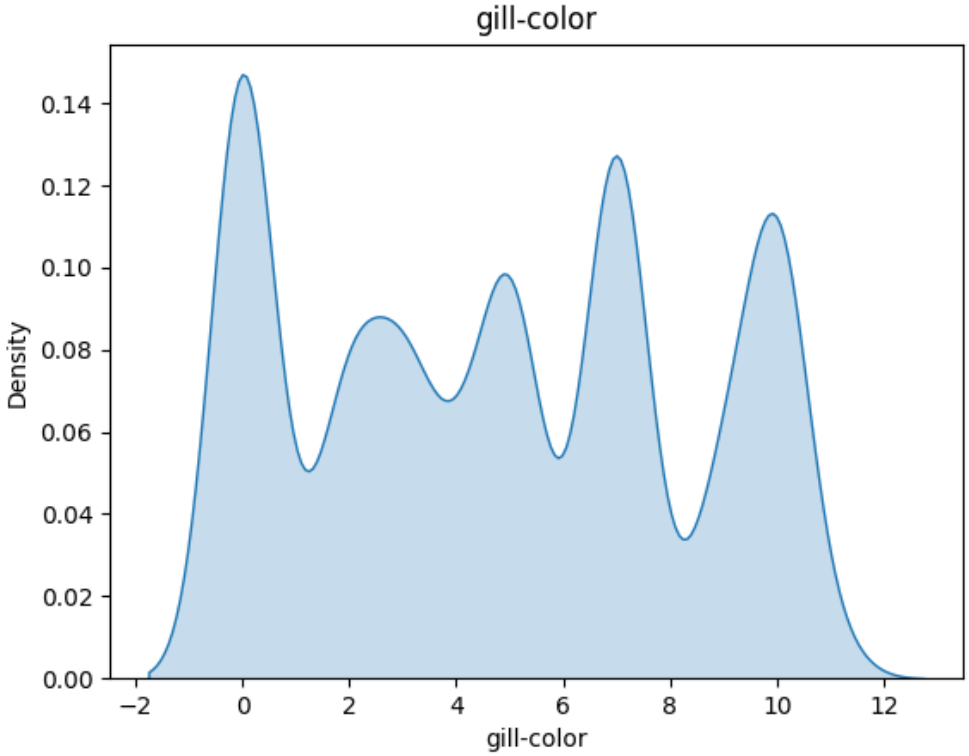


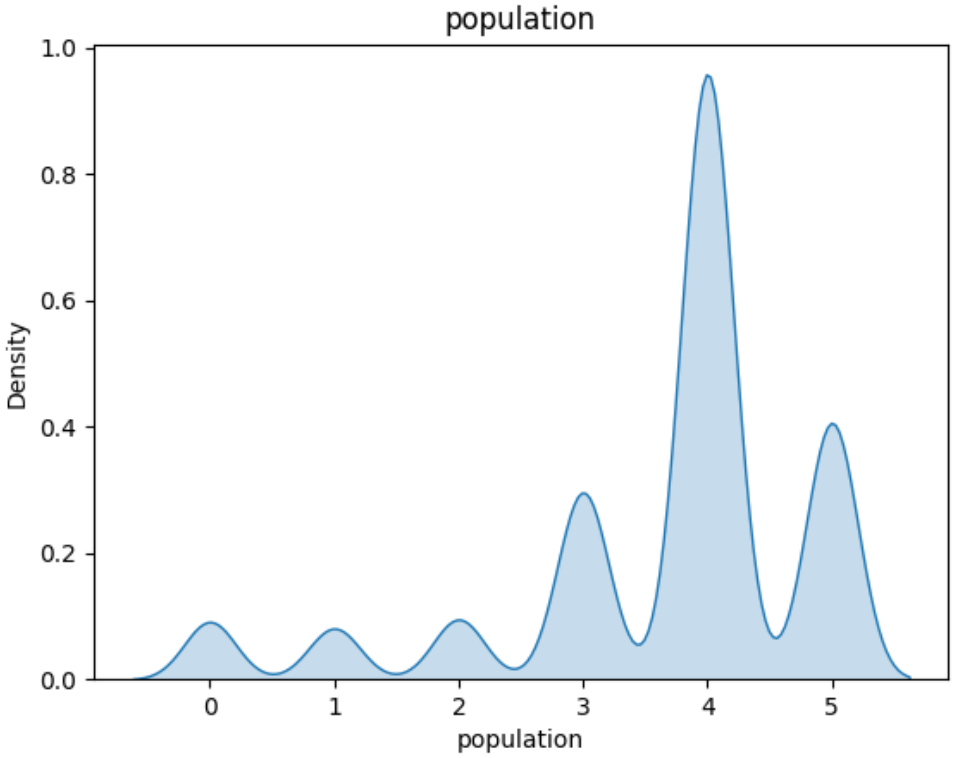


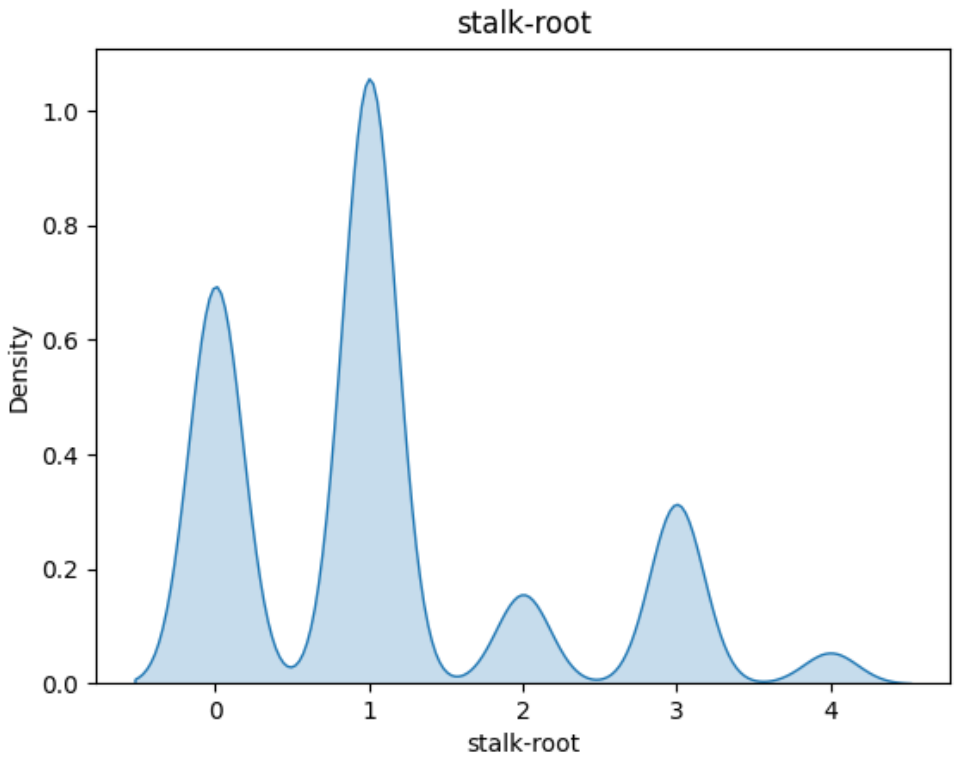

population
class
0
1
2500
2000
1500
1000
500
0
Occurances
0
1
2
3
4
5
population


stalk-root
class
0
1
2000
1750
1500
1250
1000
750
500
250
0
Occurances
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
stalk-root


cap-color
Density
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10
cap-color


gill-color
Density
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−2
0
2
4
6
8
10
12
gill-color


population
Density
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
population



Do wykonania zarówno tych, jak i dalszych wykresów zastosowaliśmy następujące biblioteki:

```
import warnings
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
import seaborn as sns
```

a) Pandas do wczytania danych z pliku .csv

b) Matplotlib oraz Seaborn do stworzenia wykresów przedstawiających wyszczególnione statystyki

c) Sklearn dla samego wykorzystania sieci i możliwości wykonania predykcji dla różnych ustawień początkowych

2. Obróbka danych

Na nasze szczęście dane które pobraliśmy były kompletne, więc nie było potrzeby uzupełniania żadnych pól losowymi wartościami.
W celu dostosowania wyciągniętych danych do dalszej pracy, konieczne okazało się natomiast zmienienie ich kodowania. Domyślnie, jak można było już zobaczyć na zdjęciach ze wstępu wszystkie kolumny zawierały opis cech w postaci znaków. W celu zamiany danych tekstowych na liczbowe, wykorzystałem instancję klasy LabelEncoder, która w pętli dokonywała transformacji danych na możliwe do obróbki.

```
labelEncoder = LabelEncoder()
for i in columns:
    data[i] = labelEncoder.fit_transform(data[i])
```

W ramach przygotowań musieliśmy także stworzyć kilka grup danych: Wejściowe treningowe, wejściowe dla analizy, wyjściowe treningowe oraz wyjściowe dla analizy.
Całą obróbkę danych udało się streścić w jednej krótkiej funkcji:

```
def GetData():
    data = pd.read_csv("mushroom.csv")

    CheckData(data)

    X = data.drop(['class'], axis=1)
    Y = data['class']

    print("Data has been loaded...\n")
    return train_test_split(X, Y, test_size=0.2, random_state=101)
```

Zwracającej cztery tablice dla grup danych wymienionych powyżej

```
X_train, X_test, Y_train, Y_test = GetData()
```

3. Opis zastosowanych sieci neuronowych

Jak już wyżej wspomniałem, dla przeprowadzenia klasyfikacji użyłem różnych funkcji z biblioteki Sklearn.

Badania przeprowadziłem łącznie dla 7 różnych konfiguracji:

a) Danych testowych, dla upewnienia się że wszystkie poprzednie kroki zostały wykonane prawidłowo. Jeśli dla zbioru testowego program nie byłby w stanie dokonać poprawnej predykcji, byłby to jednoznaczny sygnał że coś zostało wcześniej błędnie wykonane

b) Z wykorzystaniem ADABoostClassifier, który nie korzysta z sieci neuronowych tylko zbioru różnych słabszych klasyfikatorów, w celu połączenia ich rezultatów i na ich podstawie uzyskania jednej trafniejszej predykcji. Została umieszczona wyłącznie w celach porównawczych.

c) 5 różnych ustawień MLPClassifier, czyli naszego punktu docelowego, klasyfikatora wykorzystującego sieci neuronowe. Wśród rzeczy zmienianych były: liczba neuronów, tempo uczenia się sieci, rodzaje solwera czy też możliwości adaptacji. Szczegóły znajdują się w kodzie programu.

Standardowy wydruk z programu pokazywał nam dwie rzeczy:
a) Końcowy raport na temat skuteczności danej predykcji

```
Test set values predicted with MLP custom setup(sgd solver, slow learning rate with adaptive abilities):

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       851
           1       1.00      1.00      1.00       774

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

b) Średnią wartość straty, liczonej na podstawie pewnej funkcji celu zależnej od ustawień.

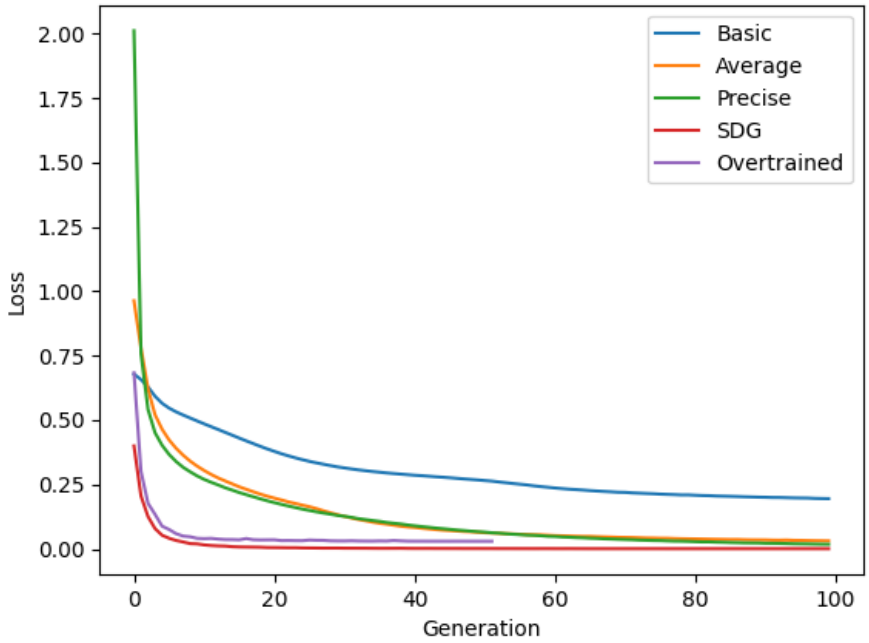
```
Iteration 1, loss = 0.39978711
Iteration 2, loss = 0.20547528
Iteration 3, loss = 0.12552406
Iteration 4, loss = 0.07896873
Iteration 5, loss = 0.05314957
Iteration 6, loss = 0.04140741
Iteration 7, loss = 0.03281645
Iteration 8, loss = 0.02710342
Iteration 9, loss = 0.02065002
Iteration 10, loss = 0.01955184
```

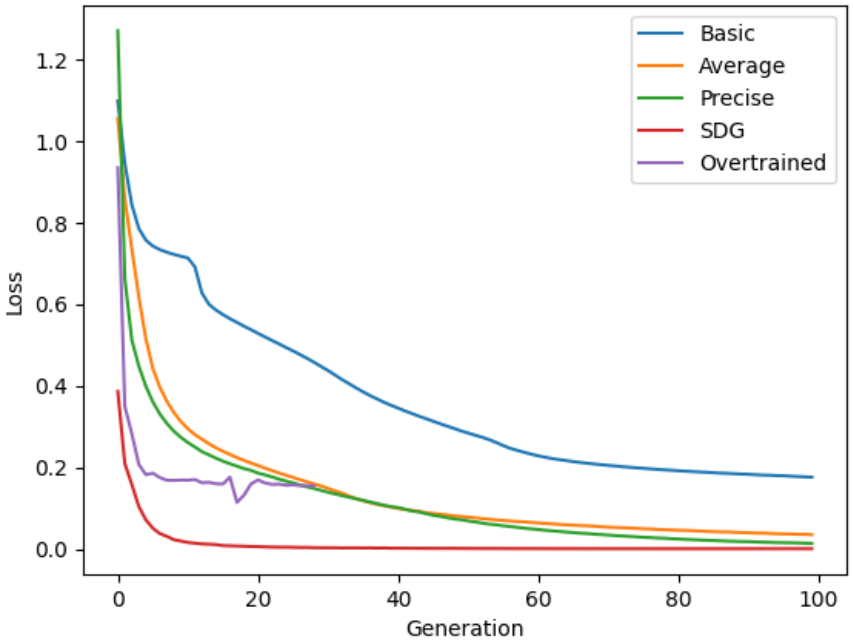
Na podstawie tak zebranych danych będziemy tworzyć opracowanie wyników.

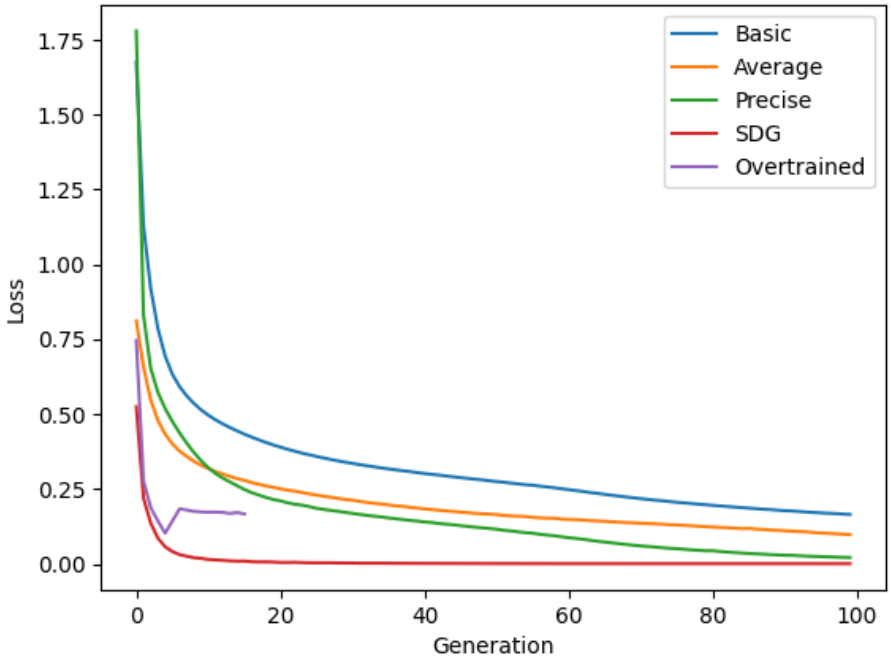
4. Dyskusja wyników oraz wnioski

Poniżej zamieszczamy wykresy wykonane na podstawie

- Wartości strat w poszczególnych iteracjach programów dla trzech różnych uruchomień:






Basic
Average
Precise
SDG
Overtrained
Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100
Generation

- Procenta precyzji trafień dla wszystkich 7 algorytmów:

```
Train set values:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       851
           1       1.00      1.00      1.00       774

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

```
Test set values predicted with ADA:

              precision    recall  f1-score   support

           0       0.78      0.89      0.83       749
           1       0.89      0.79      0.84       876

    accuracy                           0.84      1625
   macro avg       0.84      0.84      0.84      1625
weighted avg       0.84      0.84      0.84      1625
```

```
Test set values predicted with MLP basic setup(2 neurons):

              precision    recall  f1-score   support

           0       0.99      0.90      0.94       940
           1       0.87      0.99      0.93       685

    accuracy                           0.93      1625
   macro avg       0.93      0.94      0.93      1625
weighted avg       0.94      0.93      0.93      1625
```

Test set values predicted with MLP average setup(7 neurons):

	precision	recall	f1-score	support
0	1.00	0.97	0.99	871
1	0.97	1.00	0.98	754
accuracy			0.99	1625
macro avg	0.98	0.99	0.99	1625
weighted avg	0.99	0.99	0.99	1625

Test set values predicted with MLP precise setup(25 neurons):

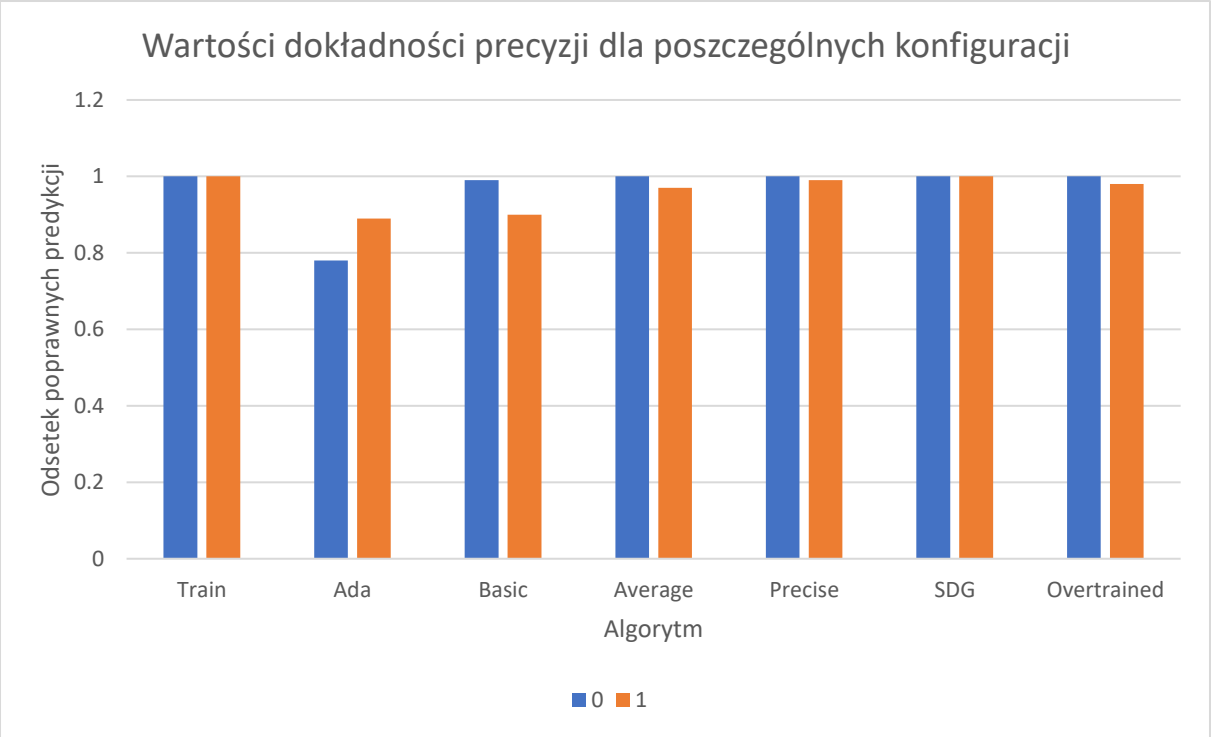
	precision	recall	f1-score	support
0	1.00	0.99	0.99	862
1	0.99	1.00	0.99	763
accuracy			0.99	1625
macro avg	0.99	0.99	0.99	1625
weighted avg	0.99	0.99	0.99	1625

Test set values predicted with MLP custom setup(sgd solver, slow learning rate with adaptive abilities):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	851
1	1.00	1.00	1.00	774
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

Test set values predicted with MLP custom setup(sgd solver, too high learning rate for given problem):

	precision	recall	f1-score	support
0	1.00	0.98	0.99	865
1	0.98	1.00	0.99	760
accuracy			0.99	1625
macro avg	0.99	0.99	0.99	1625
weighted avg	0.99	0.99	0.99	1625



Wybrane przez nas wartości najlepiej oddają charakterystykę oraz skuteczność różnych ustawień sieci, i na podstawie tych właśnie własności rezultatów jesteśmy w stanie już wyciągnąć pewne wnioski.

Pierwszą rzeczą jaka się rzuca w oczy jest fakt, że jakikolwiek algorytm implementujący sieci neuronowe jest znacznie skuteczniejszy od dostępnych alternatyw. Już dla 2 neuronów uzyskujemy predykcje skuteczniejsze o nawet 10% w porównianiu do tych otrzymanych dla ADABoostClassifier. Jednocześnie, nie zauważono znaczącej różnicy pod kątem wydajności programu dla obu tych podejść: mimo braku mierzenia czasu wykonywania się poszczególnych programów ośmielamy się stwierdzić, że predykcje zajmowały podobną ilość czasu.

Dalej, patrząc po wynikach widać że zastosowanie tylko kilku neuronów więcej daje bardzo pozytywne wyniki – celność predykcji na poziomie 98-99%. Każe nam to się zastanowić, jak często w rzeczywistości przydatniejsze będą obliczenia wykonywane przy znacznie większej liczbie neuronów skoro już tutaj mamy niemalże pewne wyniki. Inną sprawą jest też jednak to, że nasz problem był stosunkowo prosty i dla bardziej zaawansowanych zagadnień, zawierających większą liczbę możliwości do klasyfikacji bądź też problemów związanych z regresją taka sama ilość neuronów może być o wiele mniej dokładna. Pokazuje to, że nie powinniśmy w każdej sytuacji używać podobnych ustawień tylko dobierać je indywidualnie pod konkretne zadanie.

Idąc dalej, dla algorytmów jeszcze precyzyjniejszych i korzystających z innych solwerów otrzymaliśmy skuteczności kolejno 99% i nawet równe 100% dla SDG. Wynik ten powtarzał się przy każdym uruchomieniu programu, więc zakładamy że nie

jest to kwestia losowa tylko po prostu problem okazał się na tyle prosty że sieć nauczyła się w pełni rozwiązywać wszystkie przypadki podane w danych testowych.

Sprawa nie jest jednak tak jednoznaczna w przypadku algorytmu z większym krokiem czasowym, czyli Overtrained. Tam różnice potrafią być znacząco różne. Dla tego konkretnego badania otrzymaliśmy również 99%, jednakże były badania gdzie takowa była znacznie niższa:

Test set values predicted with MLP custom setup(sgd solver, too high learning rate for given problem):

	precision	recall	f1-score	support
0	0.98	0.90	0.94	923
1	0.89	0.98	0.93	702
accuracy			0.93	1625
macro avg	0.93	0.94	0.93	1625
weighted avg	0.94	0.93	0.94	1625

Daje nam to jasno do zrozumienia, że dla optymalnego działania algorytmu powinniśmy bardzo ostrożnie dobierać parametry uczenia się, gdyż nawet niewielkie zawyżenie np. tempa nauki może spowodować wielkie straty w dokładności.

Wniosek ten potwierdzają wykresy strat: dla trzech prób w jednej z nich program okazał się skuteczniejszy od nawet precyzyjnego ustawienia z wieloma neuronami, w innych natomiast zatrzymał się już na poziomie 0.2, czyli około 4x większego niż większość pozostałych ustawień.

Dla dalszej rozbudowy programu, można by przede wszystkim wybrać bardziej wymagające zagadnienie, pozwalające na dostrzeżenie większych różnic w podejściach do problemu, a także dodać parametry czasowe/zasobowe komputera, aby zbadać różnice w wydajności. Znając dokładniej biblioteki Pythona zapewne również możliwe jest wyciągnięcie większej liczby statystyk do przeanalizowania, co również pomogłoby wyciągnąć pełniejsze wnioski na temat różnic w działaniu poszczególnych opcji.