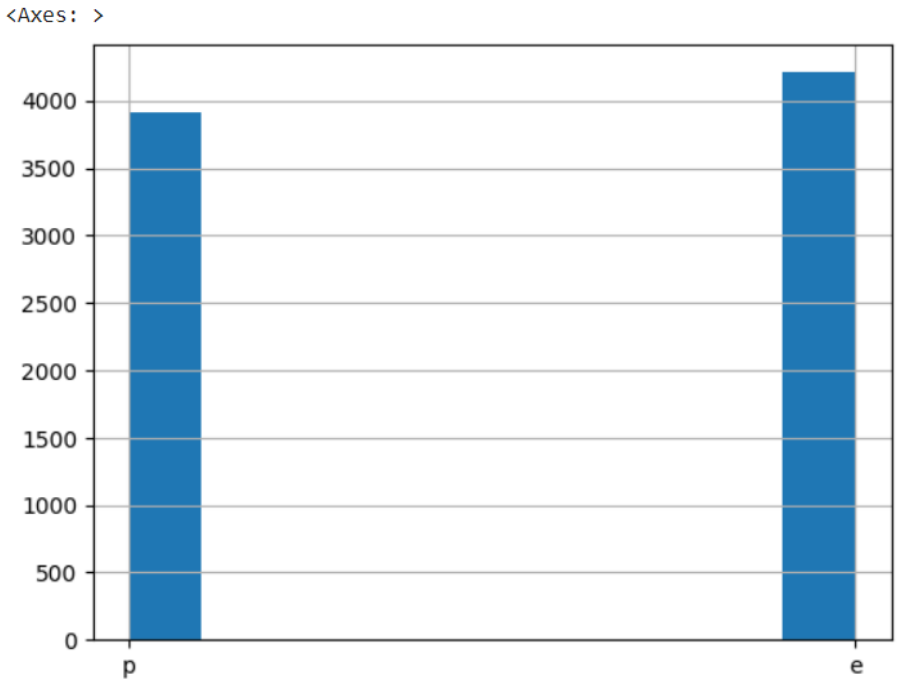
Projekt – sieci neuronowe	Data złożenia projektu:
Numer grupy projektowej:	Imię i nazwisko I: Weronika Pudło Imię i nazwisko II: Oliwia Stebelska

KLASYFIKACJA GRZYBÓW

1. Opis problemu i danych:

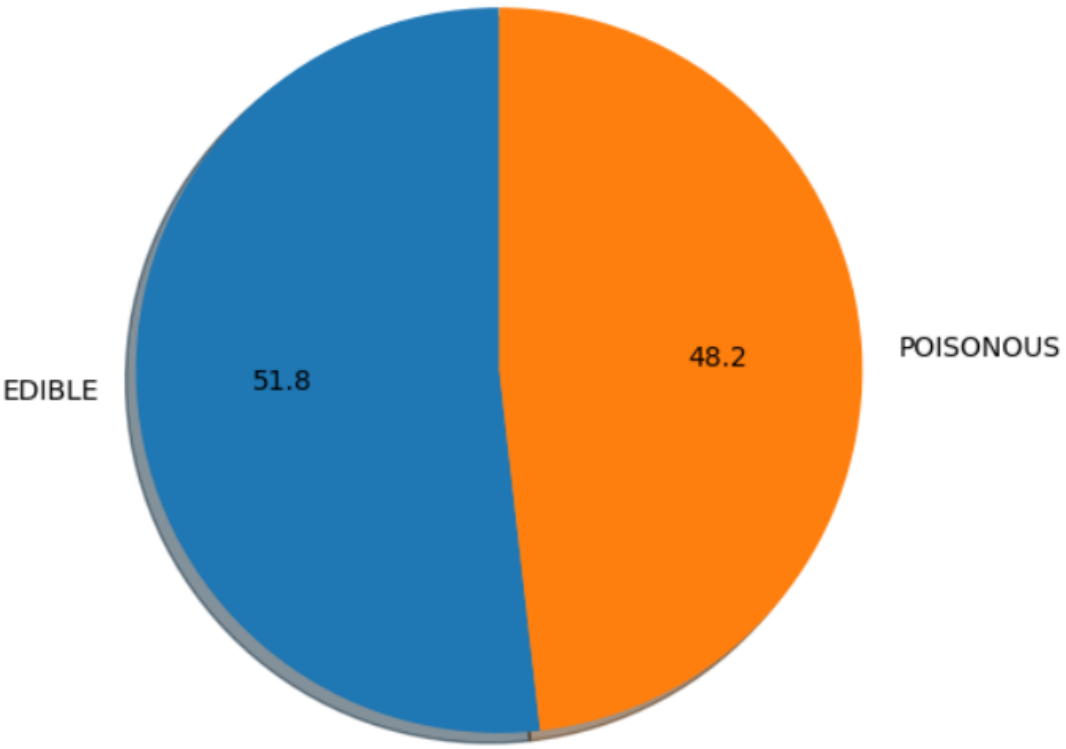
   Dane dotyczą klasyfikacji grzybów na trujące i jadalne. Zbiór danych zawiera informacje dotyczące 22 różnych atrybutów grzybów, takich jak kształt kapelusza, powierzchnia kapelusza, kolor kapelusza, siniaki, zapach, zamocowanie skrzeli, rozstaw skrzeli, rozmiar skrzeli, kolor skrzeli, kształt łodygi, korzeń łodygi, powierzchnia łodygi nad i pod pierścieniem, kolor łodygi powyżej i poniżej pierścienia, typ zasnówki, numer pierścienia, typ pierścieniowy, kolor odcisku zarodników, populacja i siedlisko. Baza danych zawiera 8124 rekordów i składa się z samych zmiennych jakościowych. Celem danego zbioru jest przewidzenie, czy dany grzyb jest jadalny czy trujący na podstawie podanych atrybutów. Naszą zmienną wyjściową jest więc zmienna ‘class’ – klasa grzyba, która przyjmuje wartości ‘e’ – jadalne, bądź ‘p’ – trujące.

   Histogram dla zmiennej wyjściowej:



Rysunek 1

Wykres kołowy dla zmiennej wyjściowej:



Rysunek 2

Udało nam się również zwrócić liczbę wystąpień zmiennej wyjściowej:

```
e    4208
p    3916
Name: class, dtype: int64
```

Rysunek 3

Dzięki powyższym danym możemy stwierdzić, że różnica między liczbą grzybów jadalnych a trujących jest niewielka. Przypuszczamy więc, że modele uczenia maszynowego trenowane na tym zbiorze danych będą miały równe szanse na naukę i rozpoznawanie jadalności grzyba.

2. Obróbka danych:

Na początku sprawdziłyśmy czy w bazie nie brakuje żadnych rekordów. Użyłyśmy do tego metody isnull() w połączeniu z metoda sum(). Dana komenda oblicza sumę brakujących wartości w każdej kolumnie bazy danych. Jeżeli wszystkie wartości okażą się kompletne, wynik dla każdej z kolumn powinien wskazywać 0. Nasza baza danych okazała się w pełni kompletna i nie miałyśmy problemu z pustymi rekordami.

```
braki = data.isnull().sum()
print(braki)
```

```
class                     0
cap-shape                 0
cap-surface               0
cap-color                 0
bruises                   0
odor                      0
gill-attachment           0
gill-spacing              0
gill-size                 0
gill-color                0
stalk-shape               0
stalk-root                0
stalk-surface-above-ring  0
stalk-surface-below-ring  0
stalk-color-above-ring    0
stalk-color-below-ring    0
veil-type                 0
veil-color                0
ring-number               0
ring-type                 0
spore-print-color         0
population                0
habitat                   0
dtype: int64
```

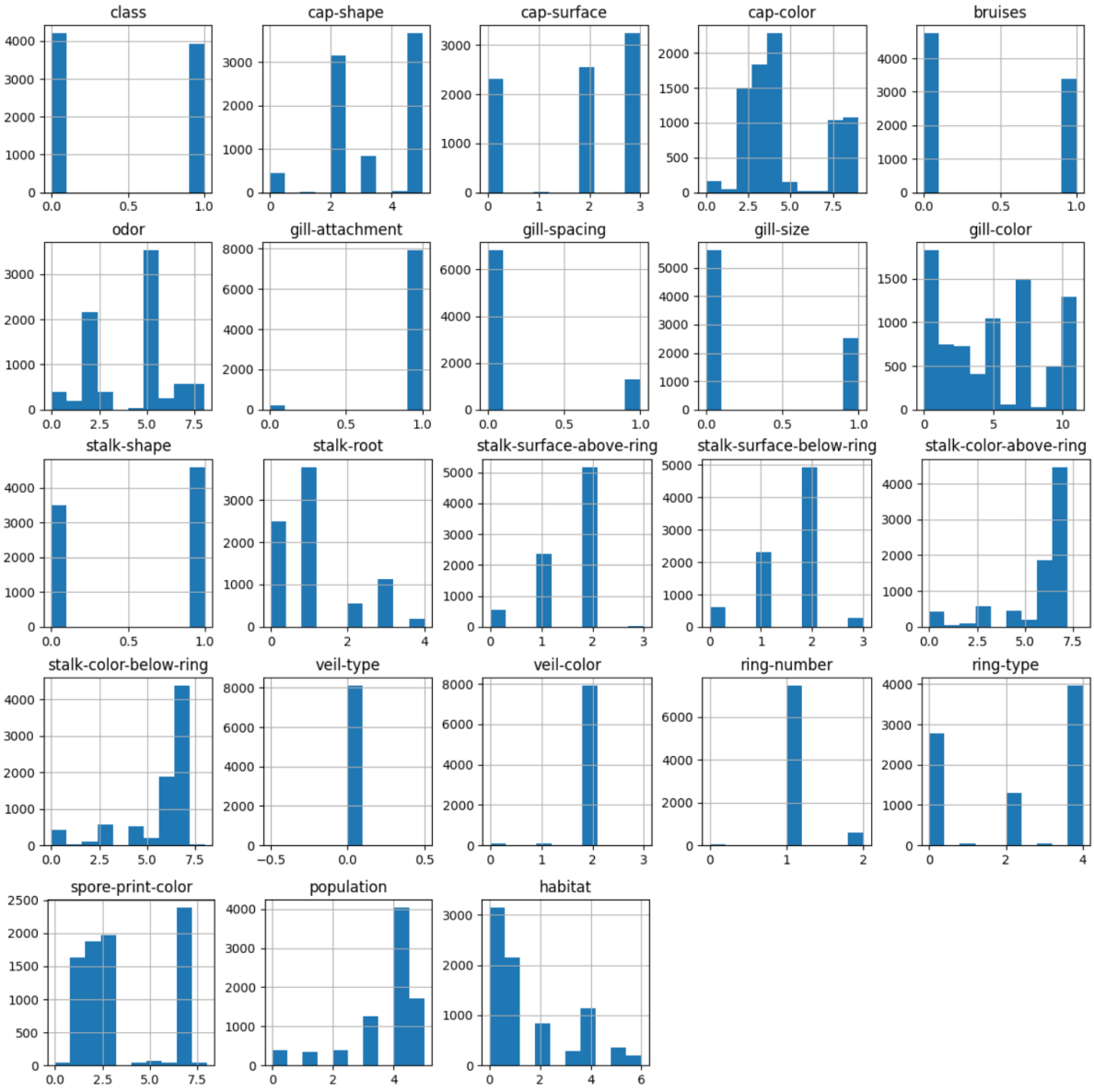
Baza danych składa się z samych zmiennych jakościowych. Aby umożliwić modelom uczenia maszynowego ich przetwarzanie należało bowiem ‘usunąć’ zmienne kategoryczne poprzez przypisanie unikalnych liczb całkowitych do różnych kategorii w każdej zmiennej kategorycznej. Użyłyśmy do tego metody fit_transform():

```
#usuwanie zmiennych kategorycznych
le = LabelEncoder()
data=data.apply(LabelEncoder().fit_transform)
data.head()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	1	5	2	4	1	6	1	0	1	4	...	2	7	7	0	2	1	4	2	3	5
1	0	5	2	9	1	0	1	0	0	4	...	2	7	7	0	2	1	4	3	2	1
2	0	0	2	8	1	3	1	0	0	5	...	2	7	7	0	2	1	4	3	2	3
3	1	5	3	8	1	6	1	0	1	5	...	2	7	7	0	2	1	4	2	3	5
4	0	5	2	3	0	5	1	1	0	4	...	2	7	7	0	2	1	0	3	0	1

5 rows × 23 columns

Po użyciu metody fit_transform utworzyłyśmy również histogramy dla innych zmiennych:



Zbiór treningowy oraz zbiór testujący:

Uznałyśmy, że najlepszym sposobem podziału danych na zbiór treningowy i testujący będzie wykorzystanie podziału losowego.
Dane zostały podzielone w stosunku 75-25. Oznacza to, że 25% danych zostanie przepisanych do zbioru testowego, a 75% do zbioru treningowego.

```
[13] X = data.drop(['class'], axis = 1) #usuwamy kolumne class i zwracamy nowy DataFrame z danymi objaśniającymi
     Y = data['class'] #tworzymy serię zawierającą tylko klase grzyba - klasę, którą próbujemy przewidzieć
     #dzielimy zbiór na zbiór treningowy - 75% i testowy - 25% | random_state - ustawienie wartości ziarna generatora liczb losowych, gwarantuje powtarzalność wyników
     #X_train i X_test zawierają zmienne objaśniające, a Y_train i Y_test zawierają etykiety klas dla odpowiednio zbiorów treningowego i testowego
     X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.25, random_state=0)
```

Dla pewności wyświetliłyśmy liczbę wierszy i kolumn w zbiorach treningowych i testowych:

```
[18] # wyświetlenie liczby wierszy i kolumn w zbiorach danych treningowych i testowych
     print("X_test shape:",X_test.shape)
     print("X_train shape:",X_train.shape)

     X_test shape: (2031, 22)
     X_train shape: (6093, 22)
```

```
[19] print("Y_test shape:",Y_test.shape)
     print("Y_train shape:",Y_train.shape)

     Y_test shape: (2031,)
     Y_train shape: (6093,)
```

3. Opis zastosowanych sieci neuronowych:

Typ sieci:
W projekcie zastosowałyśmy typ sieci: kierunkowe – wielowarstwoweb. W takich sieciach informacja przepływa tylko w jednym kierunku, od warstwy wejściowej do wyjściowej. Oznacza to, że dane nie mogą wracać się do warstw poprzednich. Każda warstwa neuronów połączona jest z warstwą następną. Wpływ danych na każdą warstwę powodują tzw. wagi połączeń między neuronami.

Framework:
Do stworzenia i nauczania sieci neuronowych wykorzystałyśmy framework TensorFlow oraz model Sequential, w którym definiowałyśmy kolejne warstwy: wejściowa, ukryte i wyjściowa.

Parametry sieci:

```
[21] model.compile(
         loss=keras.losses.sparse_categorical_crossentropy, #funkcja straty
         optimizer=keras.optimizers.Adam(), #optymalizator
         metrics=["accuracy"] #metryka
     )
     #model gotowy do proocesu uczenia
```

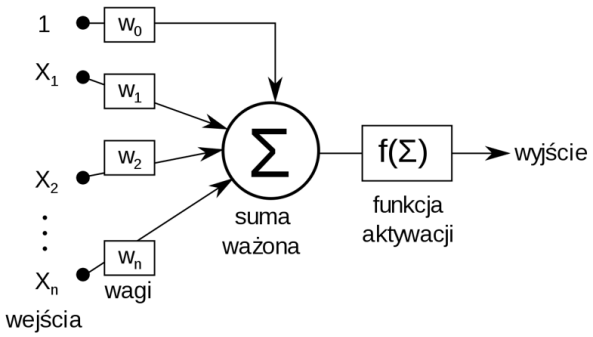
```
[59] model.compile(
         loss=keras.losses.sparse_categorical_crossentropy, #funkcja straty
         optimizer=keras.optimizers.Adam(), #optymalizator
         metrics=['sparse_categorical_accuracy'] #metryka
     )
     #model gotowy do proocesu uczenia
```

Funkcja straty: sparse_categorical_crossentropy – jest ona często używana w problemach klasyfikacji wielowarstwowej, gdzie oczekuje się, że etykiety klas będą reprezentowane za pomocą liczb całkowitych
Optymalizator: Adam.
Metryki:
– Accuracy: oblicza procent poprawnie sklasyfikowanych próbek binarnych w stosunku do wszystkich próbek. Działa poprawnie dla problemów klasyfikacji binarnej, w których etykiety klas są kodowane jako 0 i 1.
-Dokładność Kategoryczna – Categorical Accuracy: jest używana w przypadku problemów klasyfikacji wieloklasowej, gdzie etykiety klas są reprezentowane jako wektory binarne. W przypadku problemu z dwiema klasami, metryka categorical accuracy zachowuje się tak samo jak metryka accuracy.

Ilość epok: 100

```
[39] hist=model.fit(x=X_train, y=Y_train, epochs=100, validation_data=(X_test, Y_test))
```

Sposób uczenia sieci:



Sposób uczenia sieci możemy opisać na podstawie powyższego schematu.
Z informacji wynikających ze schematu sieci neuronowej przedstawionego w materiałach z zajęć możemy zdefiniować trzy warstwy:

- Warstwa wejściowa, która przyjmuje dane wejściowe (wartości liczbowe), Każde wejście jest ważone losową wagą, określającą jego znaczenie dla działania neuronu, która aktualizuje się w trakcie procesu nauki.
  Neuron oblicza ważoną sumę wszystkich wejść, gdzie każde z nich jest pomnożone przez odpowiadającą im wagę.
- Warstwy ukryte, funkcja aktywacji: Relu- odpowiadająca za przekształcenie sumy ważonej wejść na wyjście neuronu.
  Sieci neuronowe mogą mieć jedną lub więcej warstw ukrytych. Im więcej warstw ukrytych, tym bardziej złożone i zdolne do wykrywania skomplikowanych wzorców mogą być sieci. Każda warstwa ukryta składa się z wielu neuronów, które przetwarzają informacje niezależnie i na różne sposoby. Porównywanie wpływu ilości warstw ukrytych i ilości neuronów w nich zawartych na dokładność nauczania przedstawiłyśmy poniżej.
- Warstwa wyjściowa – funkcja aktywacji: softmax. Wyjście neuronu jest wynikiem działania funkcji aktywacji na sumie ważonych wejść, w naszym przypadku jest to liczba neuronów równa 2 (‘e’ – jadalne, bądź ‘p’ – trujące).

Zastosowałyśmy 5 różnych architektur:

A.

```
[20] model = tf.keras.Sequential()
     model.add(tf.keras.layers.Input(shape=X_train.shape[1])) #warstwa wejściowa
     model.add(tf.keras.layers.Dense(2000, activation="relu")) #funckje aktywacji ReLU
     model.add(tf.keras.layers.Dense(1600, activation="relu"))
     model.add(tf.keras.layers.Dense(1200, activation="relu"))
     model.add(tf.keras.layers.Dense(600, activation="relu"))
     model.add(tf.keras.layers.Dense(150, activation="relu"))
     model.add(tf.keras.layers.Dense(2, activation="softmax"))
```

B.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=X_train.shape[1])) #warstwa wejściowa
model.add(tf.keras.layers.Dense(300, activation="relu")) #funckje aktywacji ReLU
model.add(tf.keras.layers.Dense(260, activation="relu"))
model.add(tf.keras.layers.Dense(220, activation="relu"))
model.add(tf.keras.layers.Dense(180, activation="relu"))
model.add(tf.keras.layers.Dense(140, activation="relu"))
model.add(tf.keras.layers.Dense(100, activation="relu"))
model.add(tf.keras.layers.Dense(60, activation="relu"))
model.add(tf.keras.layers.Dense(20, activation="relu"))
model.add(tf.keras.layers.Dense(2, activation="softmax"))
```

C.

```
model = tf.keras.Sequential()
#precision_metric = tf.keras.metrics.Precision()
model.add(tf.keras.layers.Input(shape=X_train.shape[1])) #warstwa wejściowa
model.add(tf.keras.layers.Dense(300, activation="relu"))#funckje aktywacji ReLU
model.add(tf.keras.layers.Dense(280, activation="relu"))
model.add(tf.keras.layers.Dense(260, activation="relu"))
model.add(tf.keras.layers.Dense(240, activation="relu"))
model.add(tf.keras.layers.Dense(220, activation="relu"))
model.add(tf.keras.layers.Dense(200, activation="relu"))
model.add(tf.keras.layers.Dense(180, activation="relu"))
model.add(tf.keras.layers.Dense(160, activation="relu"))
model.add(tf.keras.layers.Dense(140, activation="relu"))
model.add(tf.keras.layers.Dense(120, activation="relu"))
model.add(tf.keras.layers.Dense(100, activation="relu"))
model.add(tf.keras.layers.Dense(80, activation="relu"))
model.add(tf.keras.layers.Dense(60, activation="relu"))
model.add(tf.keras.layers.Dense(40, activation="relu"))
model.add(tf.keras.layers.Dense(20, activation="relu"))
model.add(tf.keras.layers.Dense(2, activation="softmax"))
```

D.

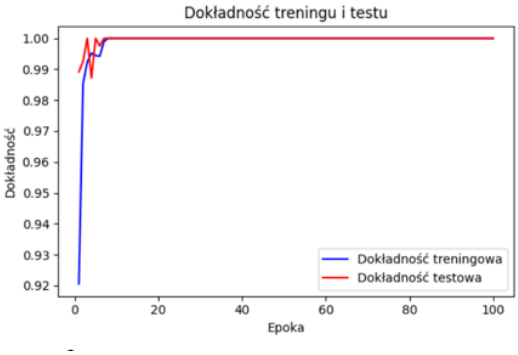
```
[45] model = tf.keras.Sequential()
     model.add(tf.keras.layers.Input(shape=X_train.shape[1])) #warstwa wejściowa
     model.add(tf.keras.layers.Dense(600, activation="relu"))#funckje aktywacji ReLU
     model.add(tf.keras.layers.Dense(300, activation="relu"))
     model.add(tf.keras.layers.Dense(100, activation="relu"))
     model.add(tf.keras.layers.Dense(2, activation="softmax"))
```

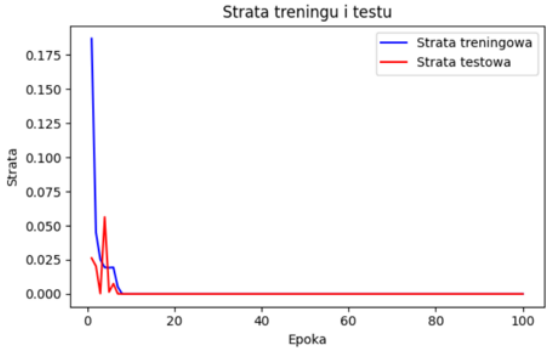
E.

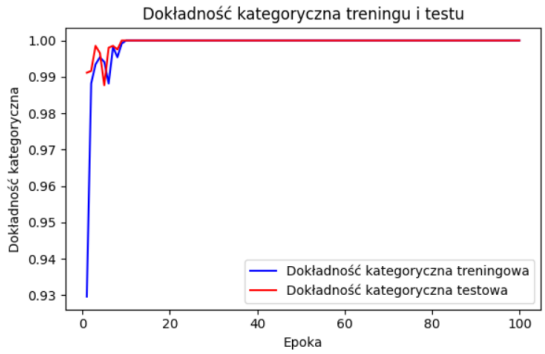
```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=X_train.shape[1])) #warstwa wejściowa
model.add(tf.keras.layers.Dense(100, activation="relu"))#funckje aktywacji ReLU
model.add(tf.keras.layers.Dense(50, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="relu"))
model.add(tf.keras.layers.Dense(2, activation="softmax"))
```

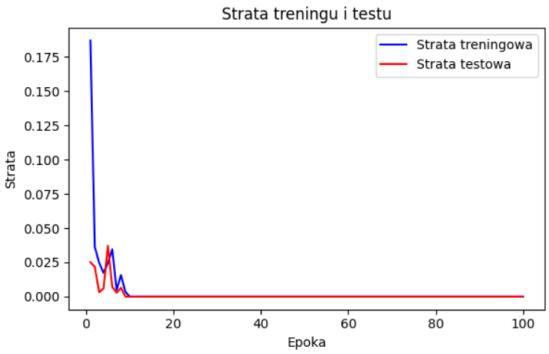
4. Dyskusja wniosków oraz wyniki:

1.

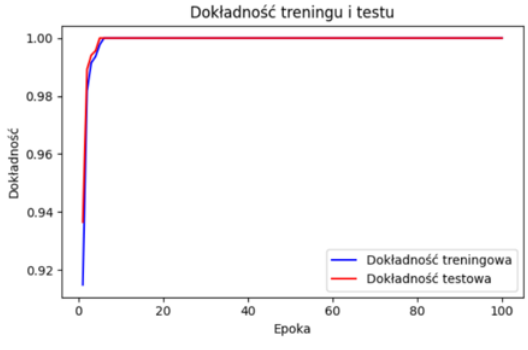


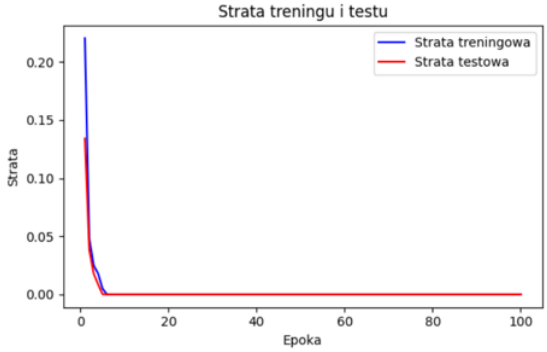


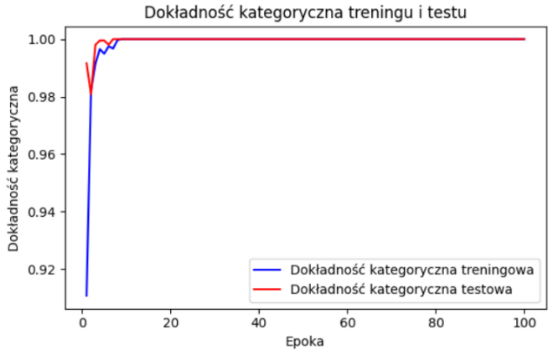


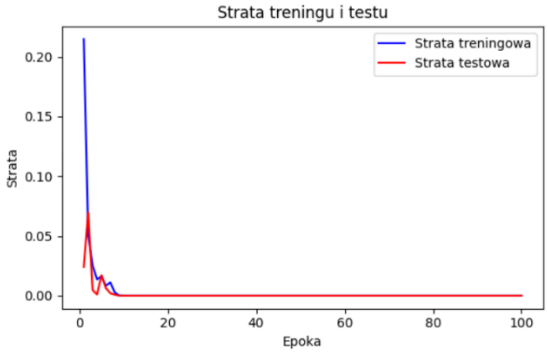


2.

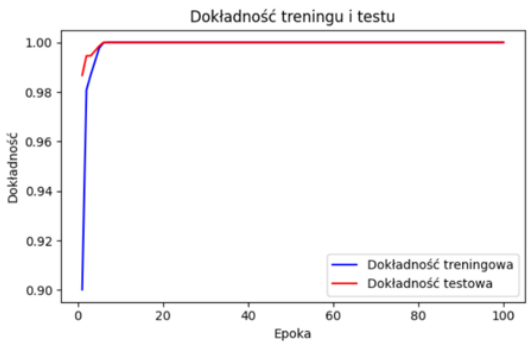


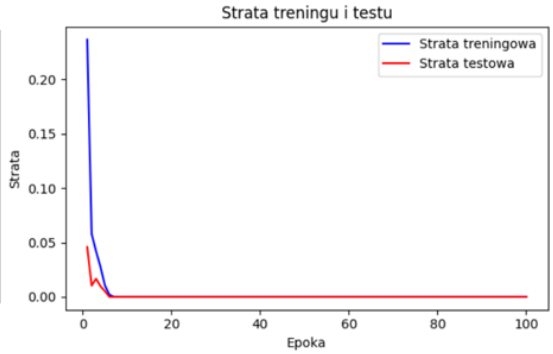


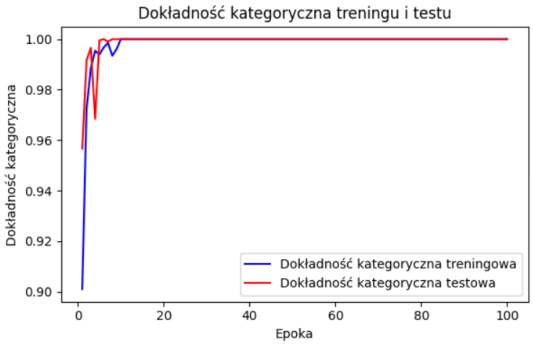


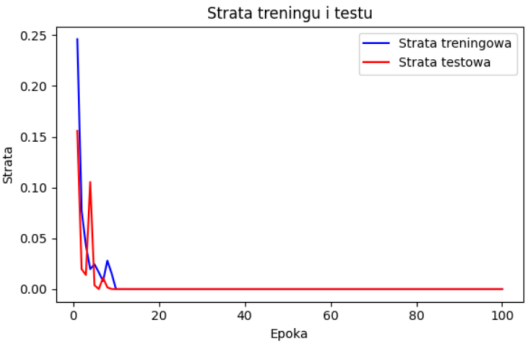


3.

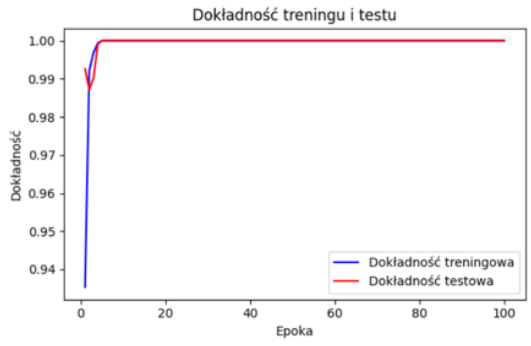

Dokładność treningu i testu
Dokładność
Epoka
Dokładność treningowa
Dokładność testowa


Strata treningu i testu
Strata
Epoka
Strata treningowa
Strata testowa

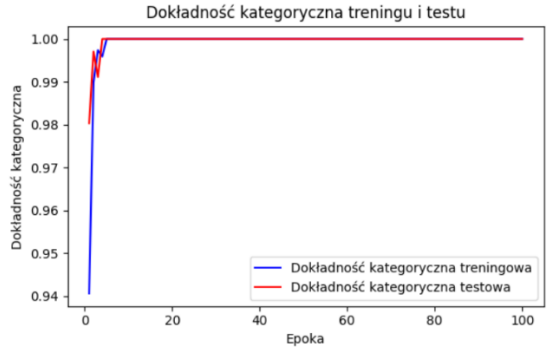

Dokładność kategoryczna treningu i testu
Dokładność kategoryczna
Epoka
Dokładność kategoryczna treningowa
Dokładność kategoryczna testowa

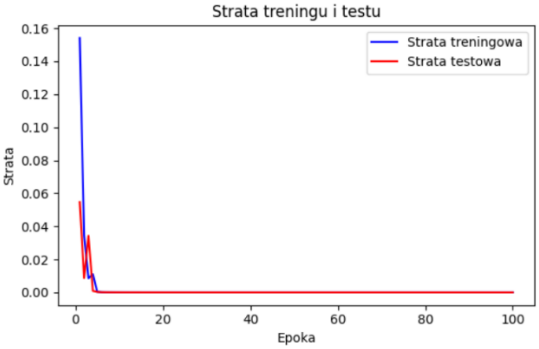

Strata treningu i testu
Strata
Epoka
Strata treningowa
Strata testowa

4.

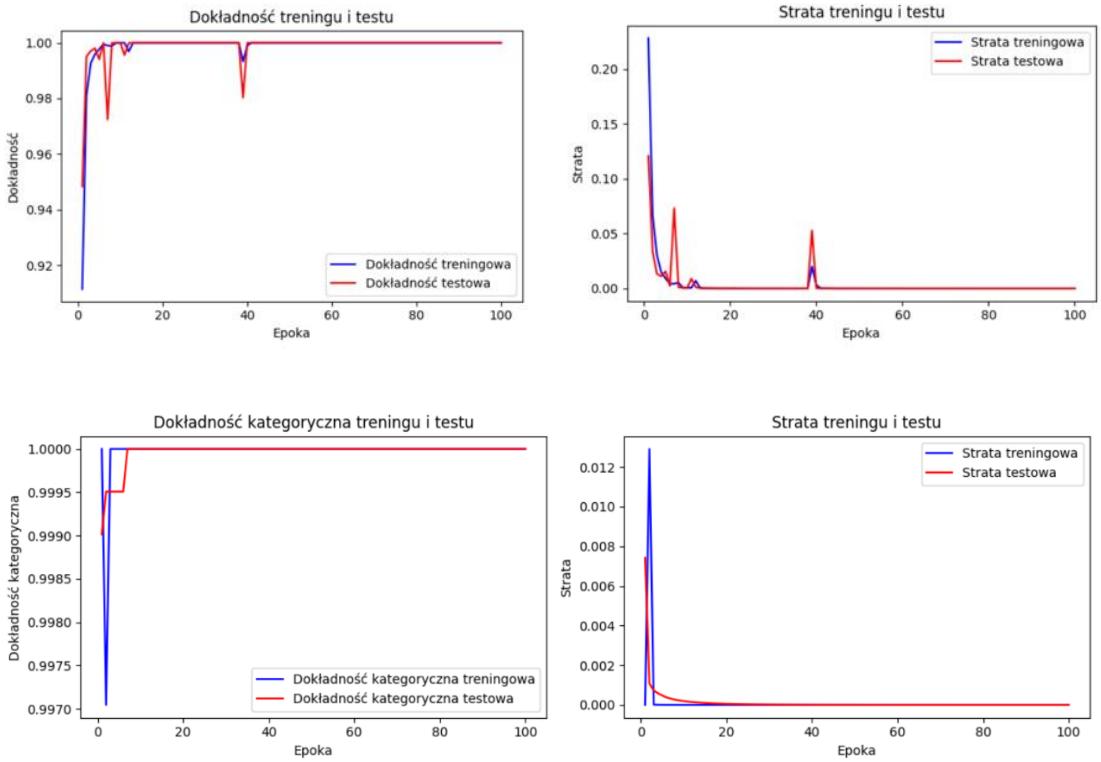

Dokładność treningu i testu
Dokładność
Epoka
Dokładność treningowa
Dokładność testowa


Strata treningu i testu
Strata
Epoka
Strata treningowa
Strata testowa

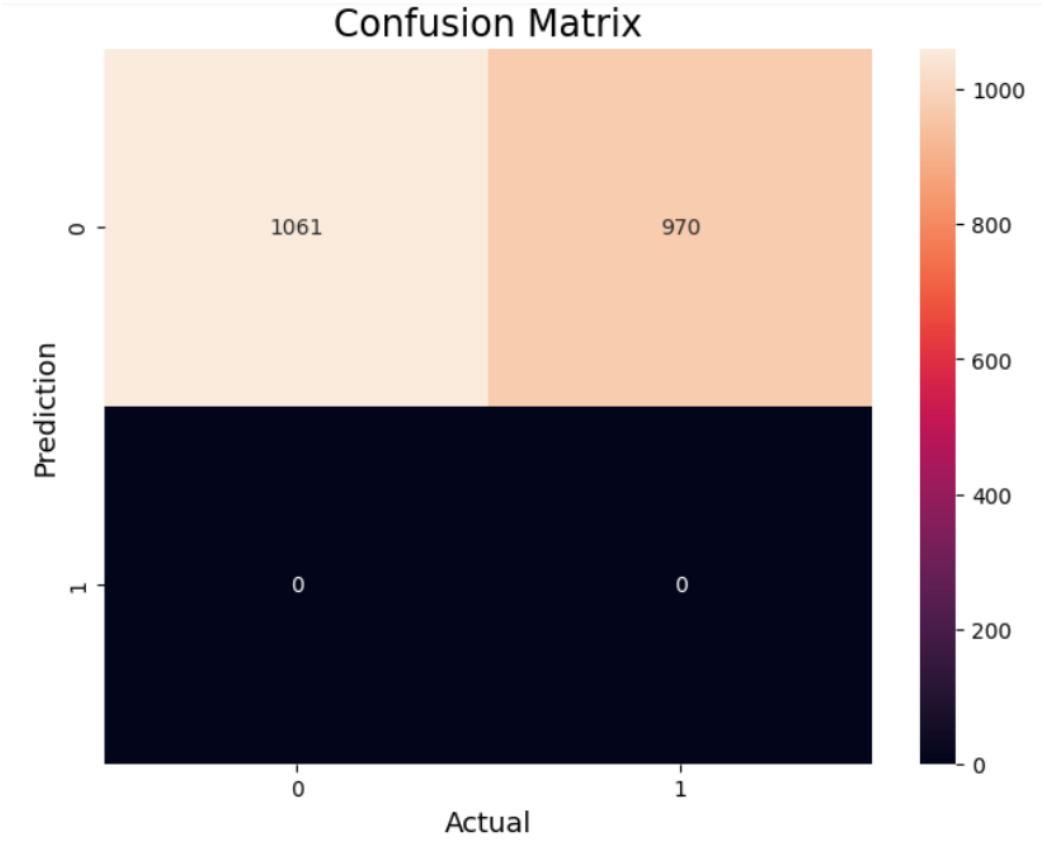

Dokładność kategoryczna treningu i testu
Dokładność kategoryczna
Epoka
Dokładność kategoryczna treningowa
Dokładność kategoryczna testowa


Strata treningu i testu
Strata
Epoka
Strata treningowa
Strata testowa

5.



Confusion matrix – Macierz Pomyłek:




```
Accuracy   : 0.5224027572624323
```

W macierzy pomyłek znajdują się 4 pola:

1. True Positive (TP): Liczba próbek, które zostały poprawnie zaklasyfikowane jako pozytywne (prawdziwie pozytywne)
2. False Positive (FP): Liczba próbek, które zostały błędnie zaklasyfikowane jako pozytywne (fałszywie pozytywne)
3. True Negative (TN): Liczba próbek, które zostały poprawnie zaklasyfikowane jako negatywne (prawdziwie negatywne)
4. False Negative (FN): Liczba próbek, które zostały błędnie zaklasyfikowane jako negatywne (fałszywie negatywne)

Na podstawie tych czterech wartości możemy wyliczyć różne metryki. W naszym projekcie udało nam się wyliczyć z macierzy pomyłek jedną miarę: dokładność, która wyniosła około 52%.

Dokładność – Odsetek prawidłowo zaklasyfikowanych próbek: TP + TN w stosunku do liczby wszystkich próbek.

Oznacza to, że model ma ograniczoną zdolność do poprawnego klasyfikowania danych. Model sklasyfikował poprawnie około połowy próbek. Sugeruje to, że ma on trudności w rozróżnianiu grzybów na podstawie dostępnych cech. Jednak błędne sklasyfikowanie próbek przez maszynę niekoniecznie oznacza, że maszyna nie nauczyła się ich wcale. W naszym modelu czynnikiem, który mógł wpłynąć na te błędy jest: brak wystarczającej ilości danych treningowych.

Wybrałyśmy dwie miary:

a. Accuracy – Dokładność: czyli stosunek liczby poprawnie sklasyfikowanych przykładów do liczby wszystkich przykładów. Wybrałyśmy ją, ponieważ jest ona najpopularniejszą metryką do oceny modeli klasyfikacji.

b. Sparse Categorical Accuracy – dokładność kategoryczna: Jest odpowiednia w przypadku problemów klasyfikacji wieloklasowej, w których etykiety klas są reprezentowane jako liczby całkowite. W naszym projekcie wartość 0 reprezentuje jedną klasę (jadalne grzyby) i wartość 1 reprezentuje drugą klasę (trujące grzyby).W takim przypadku metryka SparseCategoricalAccuracy() jest odpowiednia do oceny skuteczności modelu. Metryka ta automatycznie uwzględnia, że etykiety są reprezentowane jako liczby całkowite, i dostosowuje się do tego podczas obliczania skuteczności modelu.

Wnioski:

- Najlepszą architekturą dla naszego modelu była architektura nr. 2, gdzie liczba neuronów mieściła się w przedziale od 300 do 2.
- Dla architektury nr. 5 możemy zauważyć pojawiające się odchylenia. Mogą być spowodowane zbyt małą liczbą warstw ukrytych, bądź neuronów.
- W przypadku naszego zbioru danych zarówno metryka accuracy, jak i categorical accuracy będą dawać takie same wyniki, ponieważ mamy do czynienia z problemem klasyfikacji binarnej.
- Dokładność modelu udało się uzyskać na poziomie ok. 90%, co jest bardzo satysfakcjonującym wynikiem. Było to dość przewidywalne, ze względu na

mniejszą ilość rekordów oraz specjalne dostosowanie bazy danych pod uczenie maszynowe.

- W celu rozwoju projektu i ulepszenia modelu mogłybyśmy spróbować zmodyfikować nie tylko liczbę warstw i neuronów lecz także liczbę epok. Mogłybyśmy również wypróbować inne parametry sieci: funkcje straty czy optymalizator.