МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по Индивидуальному заданию
по дисциплине «Искусственные нейронные сети»
Тема: Многоклассовая классификация грибов

Студент гр. 7382 Дрозд А.С.

Преподаватель Жукова Н.А.

Санкт-Петербург
2020

Цель работы:

Провести классификацию грибов по категориальным признакам провести классификацию грибов по категориальным признакам.

Задачи.

1. Изучить датасет.
2. Провести начальный анализ данных.
3. Построить начальную архитектуру ИНС.
4. Изучить различные архитектуры ИНС (Разное кол-во слоев, разное кол-во нейронов на слоях)
5. Построить графики ошибок и точности в ходе обучения.
6. Выбрать наилучшую модель.

Ход работы.

1. Был рассмотрен предлагаемый датасет «mushrooms».

   Информация датасета представлена в таблице 1.

Таблица 1

Название атрибута	Значение атрибута	Сокращение значения
классы	съедобные	e
	ядовитые	p
форма колпачка	колокол	b
	конический	c
	выпуклый	x
	плоский	f
	ручка	k
	утонувший	s
поверхность колпачка	волокнистая	f
	бороздки	g
	чешуйчатая	y
	гладкая	s
цвет колпачка	коричневый	n
	бафф	b

	корица	c
	серый	g
	зеленый	r
	розовый	p
	фиолетовый	u
	красный	e
	белый	w
	желтый	y
синяки	есть	t
	нет	f
запах	миндаль	a
	анис	l
	креозот	c
	рыбный	y
	грязный	f
	затхлый	m
	без запаза	n
	острый	p
	пряный	s
жаберная насадка	прикрепленная	a
	нисходящая	d
	свободная	f
	зубчатая	n
расстояние между жабрами	близко	c
	средне	w
	далеко	d
размер жабр	широкий	b
	узкий	n
цвет жабр	черный	k
	коричневый	n
	бафф	b
	шоколад	h
	серый	g
	зеленый	r
	оранжевый	o
	розовый	p
	фиолетовый	u
	красный	e
	белый	w
	желтый	y

форма ножки	увеличивающаяся	e
	сужающаяся	t
Форма корня	луковичный	b
	клюшка	c
	чашка	u
	ровный	e
	корневище	z
	укорененый	r
	корень отсутствует	?
поверхность стебля над кольцом	волокнистая	f
	чешуйчатая	y
	шелковистая	k
	гладкая	s
поверхность стебля под кольцом	волокнистая	f
	чешуйчатая	y
	шелковистая	k
	гладкая	s
цвет стебля над кольцом	коричневый	n
	бафф	b
	корица	c
	серый	g
	оранжевый	o
	розовый	p
	красный	e
	белый	w
	желтый	y
цвет стебля под кольцом	коричневый	n
	бафф	b
	корица	c
	серый	g
	оранжевый	o
	розовый	p
	красный	e
	белый	w
	желтый	y
тип вуали	частичная	p
	универсальная	u
цвет вуали	коричневый	n
	оранжевый	o
	белый	w

	желтый	y
количество колец	ноль	n
	одно	o
	два	t
тип кольца	паутина	c
	мимолетное	e
	факельное	f
	большое	l
	нет кольца	n
	подвесное	p
	обшивка	s
	зона	z
цвет спор	черный	k
	коричневый	n
	бафф	b
	шоколад	h
	зеленый	r
	оранжевый	o
	фиолетовый	u
	белый	w
	желтый	y
распространение	в изобилии	a
	кластеризовано	c
	многочисленно	n
	разбросано	s
	несколько	v
	одиночно	y
среда обитания	травы	g
	листья	l
	луга	m
	тропы	p
	городские	u
	отходы	w
	леса	d

2. Был проведен начальный анализ и изменение данных.

Так как нейронная сеть работает с числовыми данными, каждый критерий был разбит на несколько новых так, чтобы значениям критериев можно было дать числовую величину.

Примером может служить разбиение критерия “форма шляпки” на высоту центра гриба относительно края и высоту полуцентра.

```
cap_shape_middle = [] #Критерий форма шляпки на коорбинаты
cap_shape_near_midle = []
for i in B:
   if i[0] == 'b':
      cap_shape_near_midle.append(0.4)
      cap_shape_middle.append(0.5)
   elif i[0] == 'c':
      cap_shape_near_midle.append(0.8)
      cap_shape_middle.append(1.)
   elif i[0] == 'x':
      cap_shape_near_midle.append(0.5)
      cap_shape_middle.append(1.)
   elif i[0] == 'f':
      cap_shape_near_midle.append(0.4)
      cap_shape_middle.append(0.8)
   elif i[0] == 'k':
      cap_shape_near_midle.append(-0.25)
      cap_shape_middle.append(-0.5)
   elif i[0] == 's':
      cap_shape_near_midle.append(-0.5)
      cap_shape_middle.append(-1.)
   else :
      print('error_cap_shape')
```

Или же разбиение критерия “цвет вуали” на 3 критерия по RGB.

```
veil_color_r = [] #Критерий цвета вуали по RGB
veil_color_g = []
veil_color_b = []
for i in B:
   if i[16] ==  'n':
      veil_color_r.append(150)
      veil_color_g.append(75)
      veil_color_b.append(0)
   elif i[16] == 'o':
```

```
        veil_color_r.append(255)
        veil_color_g.append(165)
        veil_color_b.append(0)
    elif i[16] == 'w':
        veil_color_r.append(255)
        veil_color_g.append(255)
        veil_color_b.append(255)
    elif i[16] == 'y':
        veil_color_r.append(255)
        veil_color_g.append(255)
        veil_color_b.append(0)
    else:
        print('error_veil_color')
```

После этого данные были собраны в новую таблицу и нормализованы.

После нормализации было замечено что некоторые критерии были однотипными (все 0). В связи с этим такие критерии были убраны из данных.

3. Была построена начальная архитектура ИНС

```
model = Sequential()
    model.add(Dense(40, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    H = model.fit(X, dummy_y, epochs=20, batch_size=10, validation_split=0.1)
```

Начальный слой Dense с 40 нейронами, т.к. после изменения датасета и уменьшения критериев их осталось 40. Несколько скрытых слоев и Выходной слой с 2 нейронами.

4. После создания стартовой модели было проведено несколько тестов с разными изменениями модели:

- Изменение количества нейронов в скрытых слоях.

  Используя функцию test_num_of_neurons() проверим несколько ИНС с разным количеством нейронов в скрытых слоях и выберем лучшую.

  Графики ошибок и точности показаны на рис. 1, 2.

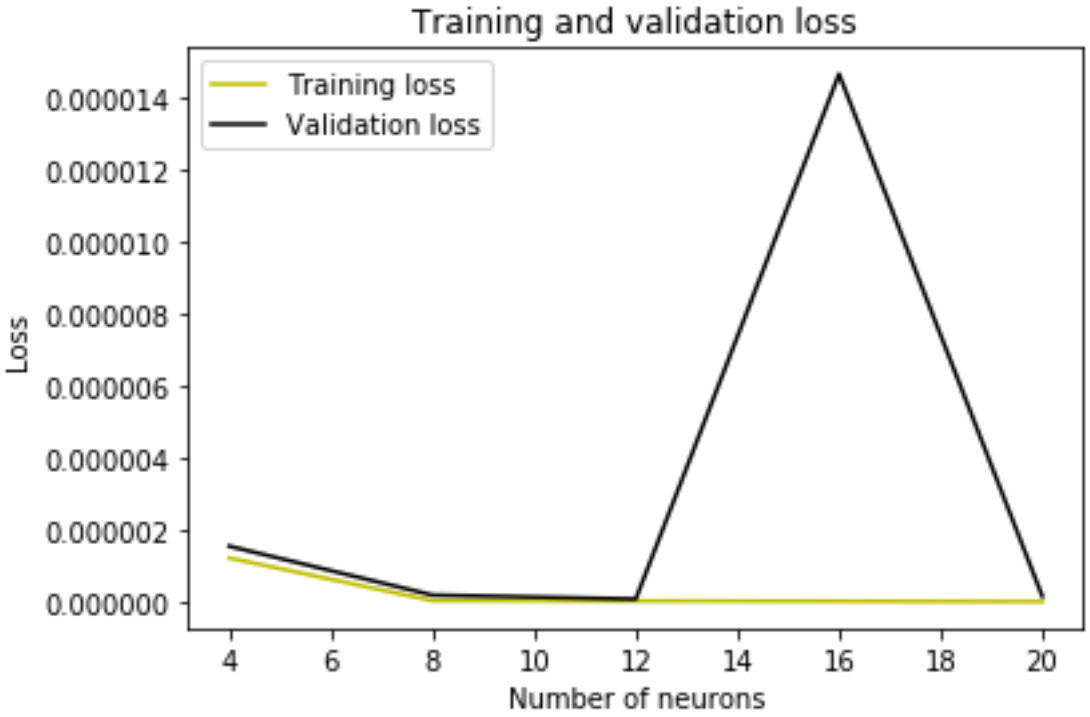

Рисунок 1 — Ошибки в зависимости от числа нейронов

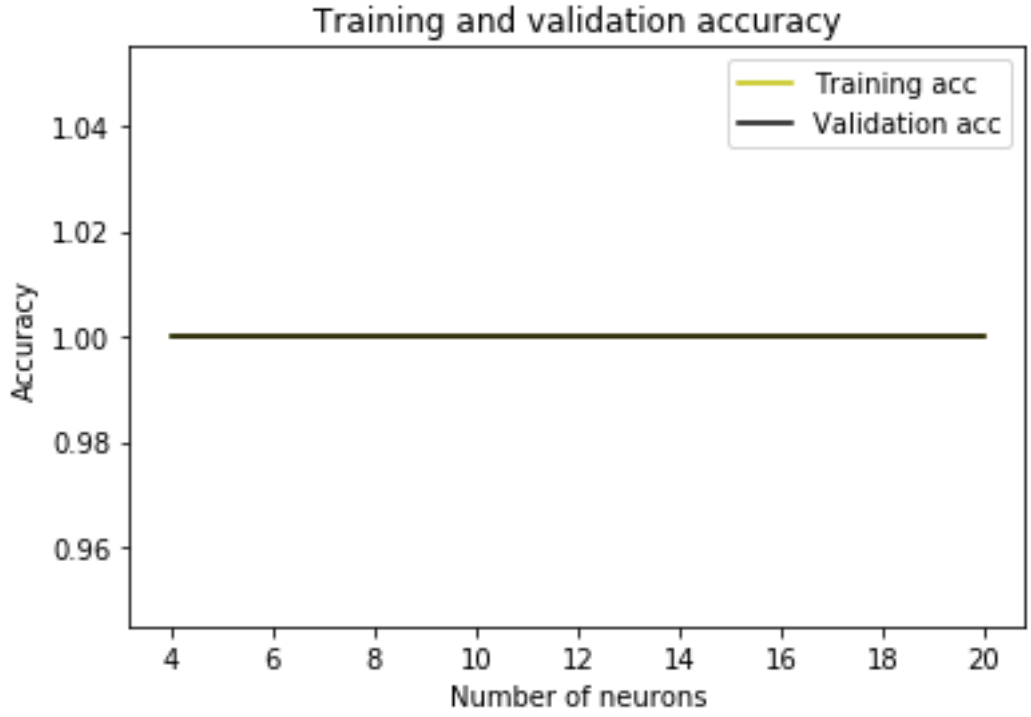

Рисунок 2 — Зависимость точности от числа нейронов

Исходя из графиков делаем вывод что точность везде одинаковая, а лучшие потери при 12 нейронах в слое.

- Изменение количества скрытых слоев

Используя функцию test_num_of_layers() проверим несколько ИНС с разным количеством скрытых слоев и выберем лучшую.

Графики ошибок и точности показаны на рис. 3, 4.

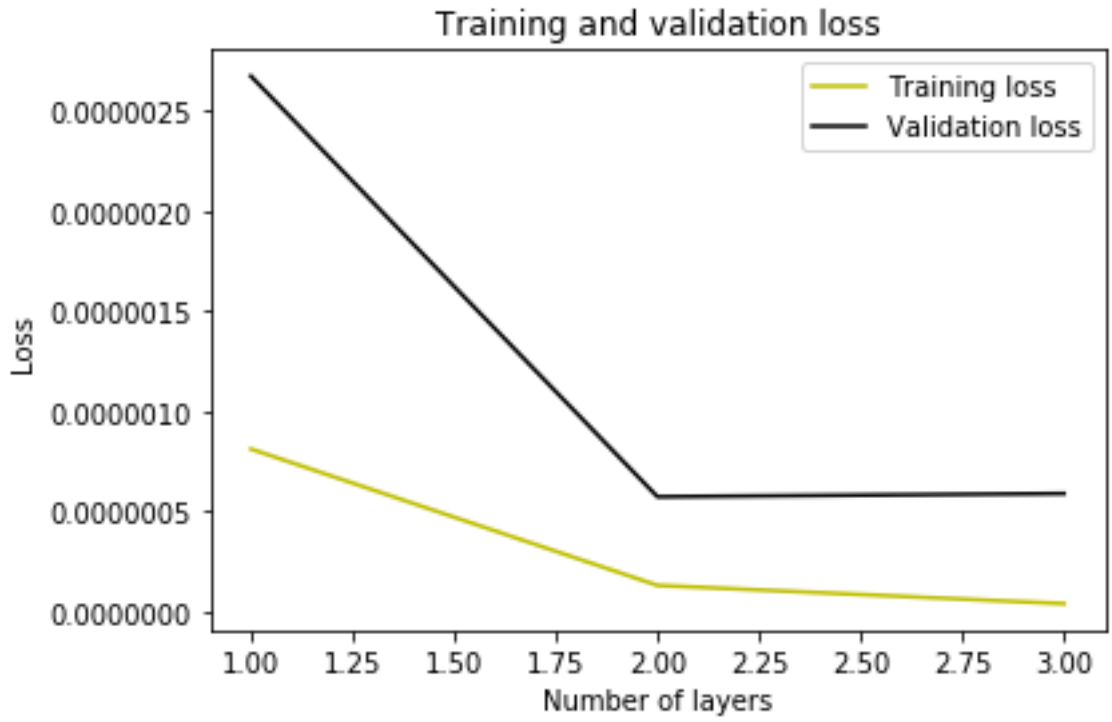

Рисунок 3 - Ошибки в зависимости от числа слоев

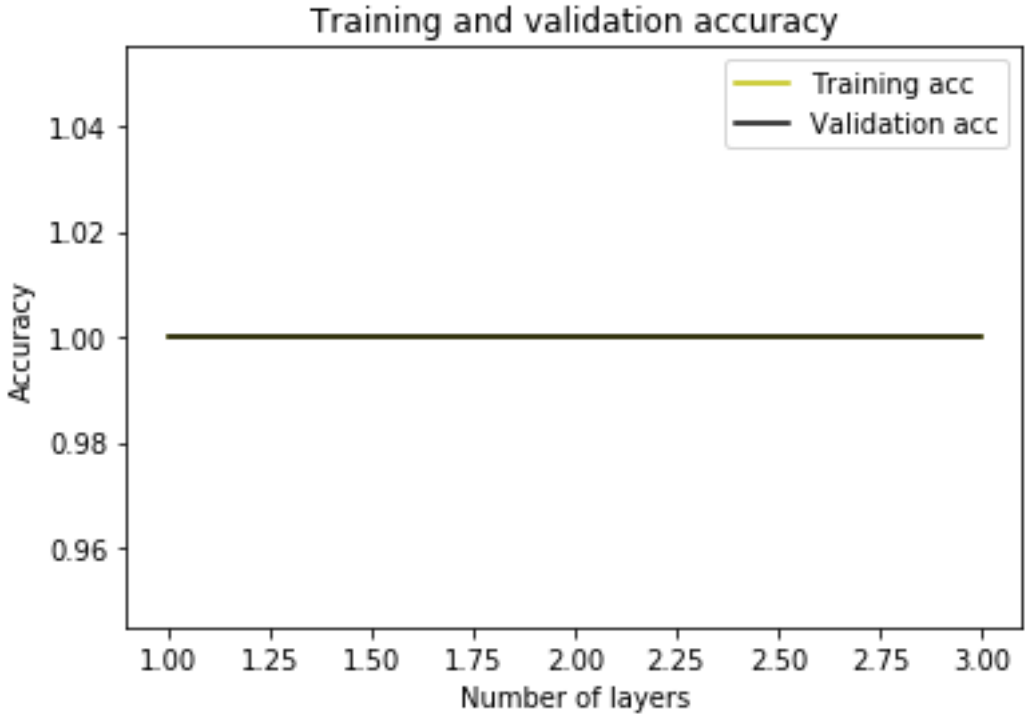

Рисунок 4 — Зависимость точности от числа слоев

Исходя из графиков делаем вывод что точность везде одинаковая, а лучшие потери при 2 скрытых слоях.

- Изменение параметра epochs
  Используя функцию test_epochs() проверим несколько ИНС с разным количеством эпох и выберем лучшую.

  Графики ошибок и точности показаны на рис. 5, 6.

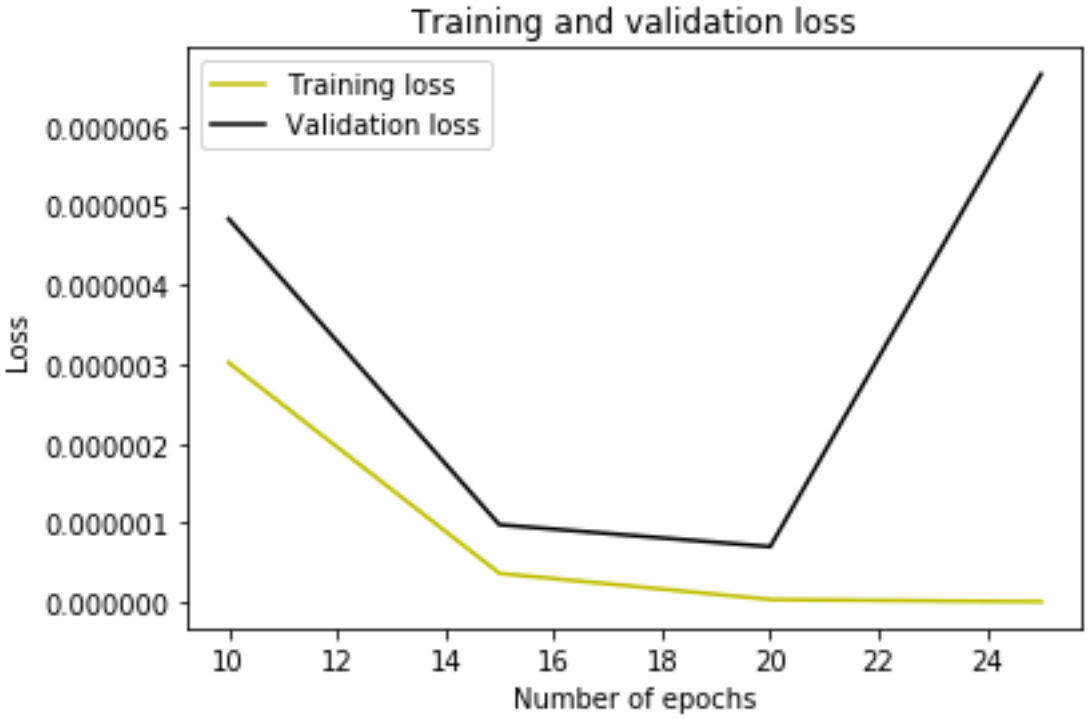

Рисунок 5 - Ошибки в зависимости от параметра epochs

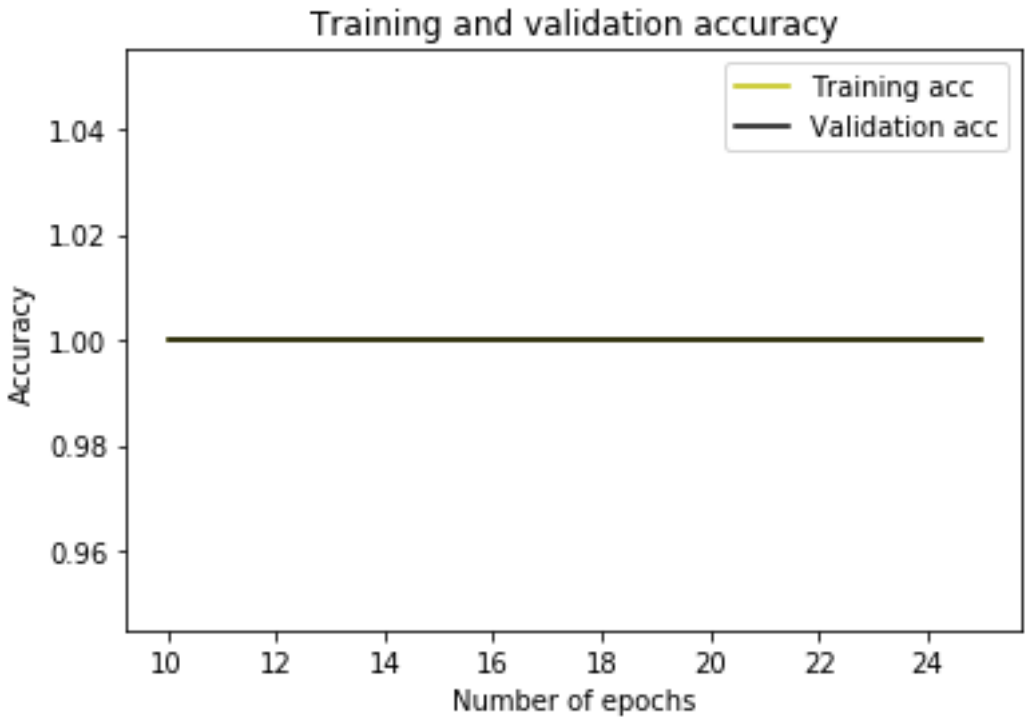

Рисунок 6 — Зависимость точности от параметра epochs

Исходя из графиков делаем вывод что точность везде одинаковая, а лучшие потери при 15 эпохах в слое.

- Изменение параметра batch_size
  Используя функцию test_batch_size() проверим несколько ИНС с разным количеством данных в сете и выберем лучшую.

  Графики ошибок и точности показаны на рис. 7, 8.

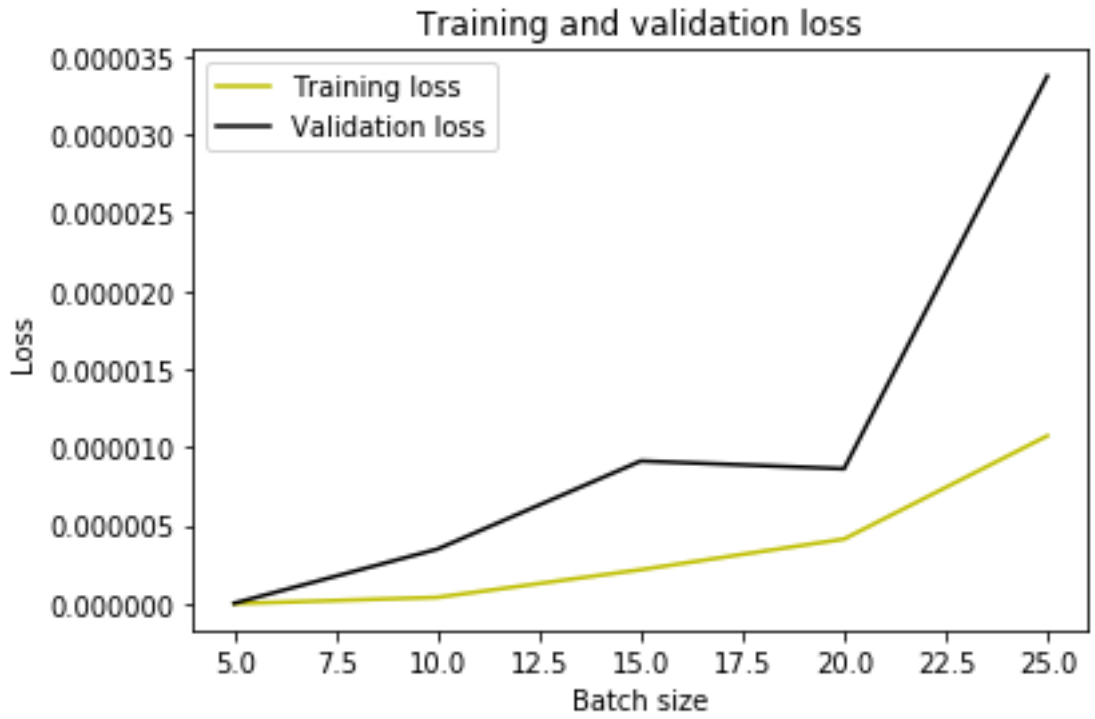

Рисунок 7 - Ошибки в зависимости от параметра batch_size

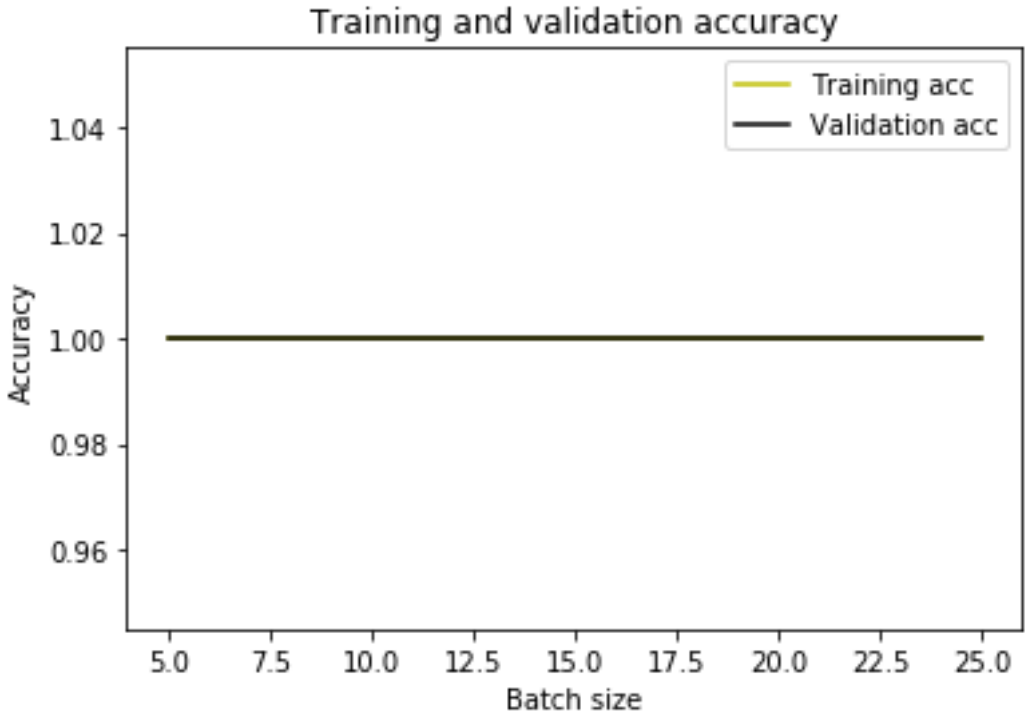

Рисунок 8 — Зависимость точности от параметра batch_size

Исходя из графиков делаем вывод что точность везде одинаковая, а лучшие потери при стаке из 20 данных (batch_size равное 5 не берем , т.к. присутствует разница по времени).

- Изменение параметра validation_split

Используя функцию validation_test() проверим несколько ИНС с разным количеством данных тестов и обучения, и выберем лучшую.

Графики ошибок и точности показаны на рис. 9, 10.

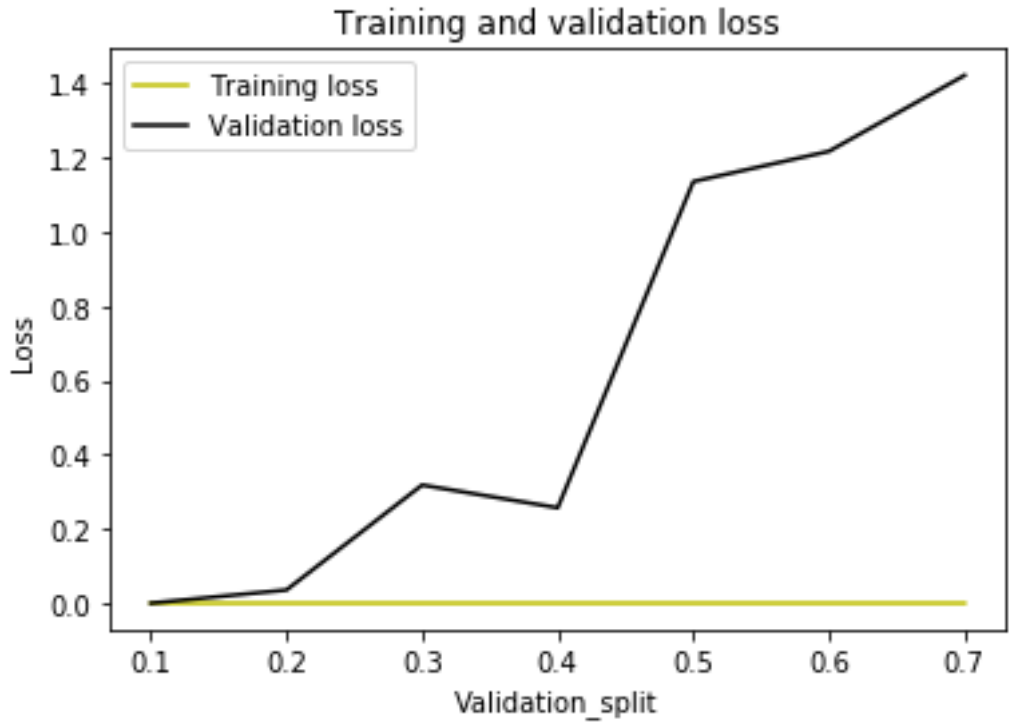

Рисунок 9 - Ошибки в зависимости от параметра validation_split

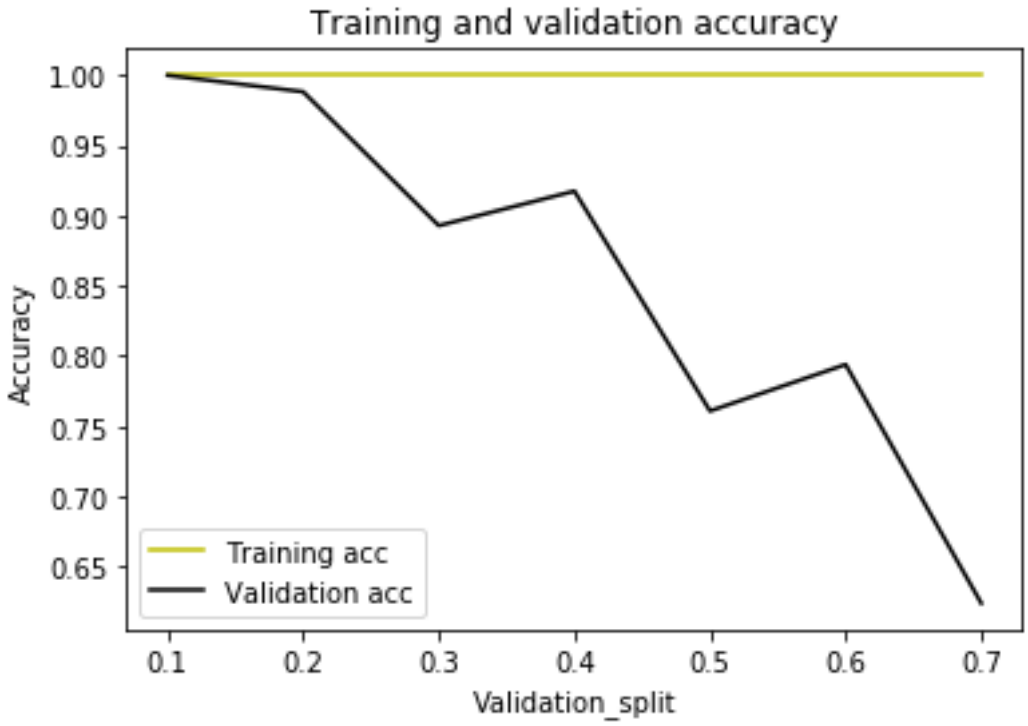

Рисунок 10 — Зависимость точности от параметра validation_split

Исходя из графиков делаем вывод что лучшая точность и ошибка при наибольшем обучающем материале.

5. Исходя из вышеперечисленных тестов лучшей моделью является ИНС

```
model = Sequential()
  model.add(Dense(40, activation='relu'))
  model.add(Dense(12, activation='relu'))
  model.add(Dense(12, activation='relu'))
  model.add(Dense(2, activation='softmax'))
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
H = model.fit(X, dummy_y, epochs=15, batch_size=20, validation_split=0.1)
```

Выводы:

Были изучены основы работы с искусственными нейронными сетями на языке python. Было исследовано поведение сети в зависимости от ее модели и параметров обучения. Была выбрана наилучшая модель.

ПРИЛОЖЕНИЕ А

```
import pandas
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

dataframe = pandas.read_csv("mushrooms.csv", header=None)
dataset = dataframe.values
B = dataset[1:, 1:23]

#№1
cap_shape_middle = [] #Критерий форма шляпки на коорбинаты
cap_shape_near_midle = []
for i in B:
    if i[0] == 'b':
        cap_shape_near_midle.append(0.4)
        cap_shape_middle.append(0.5)
    elif i[0] == 'c':
        cap_shape_near_midle.append(0.8)
        cap_shape_middle.append(1.)
    elif i[0] == 'x':
        cap_shape_near_midle.append(0.5)
        cap_shape_middle.append(1.)
    elif i[0] == 'f':
        cap_shape_near_midle.append(0.4)
        cap_shape_middle.append(0.8)
    elif i[0] == 'k':
        cap_shape_near_midle.append(-0.25)
        cap_shape_middle.append(-0.5)
    elif i[0] == 's':
        cap_shape_near_midle.append(-0.5)
        cap_shape_middle.append(-1.)
    else :
        print('error_cap_shape')

#№2
cap_surface = [] #Критерий поверхнось шляпки по шершавости
for i in B:
```

```
    if i[1] == 'f':
        cap_surface.append(0.5)
    elif i[1] == 'g':
        cap_surface.append(0.25)
    elif i[1] == 'y':
        cap_surface.append(1.)
    elif i[1] == 's':
        cap_surface.append(0.)
    else:
        print('error_cap_surface')

#№3
cap_color_r = [] #Критерий цвет шляпки по RGB
cap_color_g = []
cap_color_b = []
for i in B:
    if i[2] == 'n':
        cap_color_r.append(150)
        cap_color_g.append(75)
        cap_color_b.append(0)
    elif i[2] == 'b':
        cap_color_r.append(240)
        cap_color_g.append(220)
        cap_color_b.append(130)
    elif i[2] == 'c':
        cap_color_r.append(32)
        cap_color_g.append(2)
        cap_color_b.append(4)
    elif i[2] == 'g':
        cap_color_r.append(0)
        cap_color_g.append(0)
        cap_color_b.append(50)
    elif i[2] == 'r':
        cap_color_r.append(0)
        cap_color_g.append(128)
        cap_color_b.append(0)
    elif i[2] == 'p':
        cap_color_r.append(255)
        cap_color_g.append(192)
        cap_color_b.append(203)
```

```
    elif i[2] == 'u':
        cap_color_r.append(128)
        cap_color_g.append(0)
        cap_color_b.append(128)
    elif i[2] == 'e':
        cap_color_r.append(255)
        cap_color_g.append(0)
        cap_color_b.append(0)
    elif i[2] == 'w':
        cap_color_r.append(255)
        cap_color_g.append(255)
        cap_color_b.append(255)
    elif i[2] == 'y':
        cap_color_r.append(255)
        cap_color_g.append(255)
        cap_color_b.append(0)
    else:
        print('error_cap_color')

#№4
bruises = [] #Критерий синяков по есть/нет
for i in B:
    if i[3] == 't':
        bruises.append(1)

    elif i[3] == 'f':
        bruises.append(0)
    else:
        print('error_bruises')

#№5
odor = [] #Критерий запаха по расслаблению
for i in B:
    if i[4] == 'a':
        odor.append(0.3)
    elif i[4] == 'l':
        odor.append(0.2)
    elif i[4] == 'c':
        odor.append(-0.2)
    elif i[4] == 'y':
```

```
        odor.append(-0.5)
    elif i[4] == 'f':
        odor.append(-0.7)
    elif i[4] == 'm':
        odor.append(-1.)
    elif i[4] == 'n':
        odor.append(0.)
    elif i[4] == 'p':
        odor.append(-0.4)
    elif i[4] == 's':
        odor.append(0.5)
    else:
        print('error_odor')

#№6
gill_attachment_x = [] #Критерий жаберной насадки по точке сростания
gill_attachment_y = []
for i in B:
    if i[5] == 'a':
        gill_attachment_x.append(0)
        gill_attachment_y.append(0)
    elif i[5] == 'd':
        gill_attachment_x.append(0)
        gill_attachment_y.append(-1)
    elif i[5] == 'f':
        gill_attachment_x.append(1)
        gill_attachment_y.append(0)
    elif i[5] == 'n':
        gill_attachment_x.append(1)
        gill_attachment_y.append(1)
    else:
        print('error_gill_attachment')

#№7
gill_spacing = [] #Критерий жаберной насадки по расстоянию между жабрами
for i in B:
    if i[6] == 'c':
        gill_spacing.append(0)
    elif i[6] == 'w':
        gill_spacing.append(0.5)
```

```
    elif i[6] == 'd':
        gill_spacing.append(1)
    else:
        print('error_gill_spacing')

#№8
gill_size = [] #Критерий жаберной насадки по размеру
for i in B:
    if i[7] == 'b':
        gill_size.append(0)
    elif i[7] == 'n':
        gill_size.append(1)
    else:
        print('error_gill_size')

#№9
gill_color_r = [] #Критерий жаберной насадки по цвету
gill_color_g = []
gill_color_b = []
for i in B:
    if i[8] == 'k':
        gill_color_r.append(0)
        gill_color_g.append(0)
        gill_color_b.append(0)
    elif i[8] == 'n':
        gill_color_r.append(150)
        gill_color_g.append(75)
        gill_color_b.append(0)
    elif i[8] == 'b':
        gill_color_r.append(240)
        gill_color_g.append(220)
        gill_color_b.append(130)
    elif i[8] == 'h':
        gill_color_r.append(210)
        gill_color_g.append(105)
        gill_color_b.append(30)
    elif i[8] == 'g':
        gill_color_r.append(0)
        gill_color_g.append(0)
        gill_color_b.append(50)
```

```
    elif i[8] == 'r':
        gill_color_r.append(0)
        gill_color_g.append(128)
        gill_color_b.append(0)
    elif i[8] == 'o':
        gill_color_r.append(255)
        gill_color_g.append(165)
        gill_color_b.append(0)
    elif i[8] == 'p':
        gill_color_r.append(255)
        gill_color_g.append(192)
        gill_color_b.append(203)
    elif i[8] == 'u':
        gill_color_r.append(128)
        gill_color_g.append(0)
        gill_color_b.append(128)
    elif i[8] == 'e':
        gill_color_r.append(255)
        gill_color_g.append(0)
        gill_color_b.append(0)
    elif i[8] == 'w':
        gill_color_r.append(255)
        gill_color_g.append(255)
        gill_color_b.append(255)
    elif i[8] == 'y':
        gill_color_r.append(255)
        gill_color_g.append(255)
        gill_color_b.append(0)
    else:
        print('error_gill_color')

#№10
stalk_shape = [] #Критерий формы стебля по нарастанию
for i in B:
    if i[9] == 'e':
        stalk_shape.append(1)

    elif i[9] == 't':
        stalk_shape.append(-1)
    else:
```

```
        print('error_stalk_shape')

#№11
stalk_root_bulbous = [] #Критерий корня стебля
stalk_root_club = []
stalk_root_cup = []
stalk_root_equal = []
stalk_root_rhizomorphs = []
stalk_root_rooted = []
for i in B:
    if i[10] == 'b':
        stalk_root_bulbous.append(1)
        stalk_root_club.append(0)
        stalk_root_cup.append(0)
        stalk_root_equal.append(0)
        stalk_root_rhizomorphs.append(0)
        stalk_root_rooted.append(0)
    elif i[10] == 'c':
        stalk_root_bulbous.append(0)
        stalk_root_club.append(1)
        stalk_root_cup.append(0)
        stalk_root_equal.append(0)
        stalk_root_rhizomorphs.append(0)
        stalk_root_rooted.append(0)
    elif i[10] == 'u':
        stalk_root_bulbous.append(0)
        stalk_root_club.append(0)
        stalk_root_cup.append(1)
        stalk_root_equal.append(0)
        stalk_root_rhizomorphs.append(0)
        stalk_root_rooted.append(0)
    elif i[10] == 'e':
        stalk_root_bulbous.append(0)
        stalk_root_club.append(0)
        stalk_root_cup.append(0)
        stalk_root_equal.append(1)
        stalk_root_rhizomorphs.append(0)
        stalk_root_rooted.append(0)
    elif i[10] == 'z':
        stalk_root_bulbous.append(0)
```

```
        stalk_root_club.append(0)
        stalk_root_cup.append(0)
        stalk_root_equal.append(0)
        stalk_root_rhizomorphs.append(1)
        stalk_root_rooted.append(0)
    elif i[10] == 'r':
        stalk_root_bulbous.append(0)
        stalk_root_club.append(0)
        stalk_root_cup.append(0)
        stalk_root_equal.append(0)
        stalk_root_rhizomorphs.append(0)
        stalk_root_rooted.append(1)
    elif i[10] == '?':
        stalk_root_bulbous.append(0)
        stalk_root_club.append(0)
        stalk_root_cup.append(0)
        stalk_root_equal.append(0)
        stalk_root_rhizomorphs.append(0)
        stalk_root_rooted.append(0)
    else:
        print('error_stalk_root')

#№12
stalk_surface_above_ring = [] #Критерий поверхности стебля над кольцом по шершавости
for i in B:
    if i[11] == 'f':
        stalk_surface_above_ring.append(0.5)
    elif i[11] == 'k':
        stalk_surface_above_ring.append(0.3)
    elif i[11] == 'y':
        stalk_surface_above_ring.append(1.)
    elif i[11] == 's':
        stalk_surface_above_ring.append(0.)
    else:
        print('error_stalk_surface_above_ring')

#№13
stalk_surface_below_ring = [] #Критерий поверхности стебля под кольцом по шершавости
for i in B:
    if i[12] == 'f':
```

```
        stalk_surface_below_ring.append(0.5)
    elif i[12] == 'k':
        stalk_surface_below_ring.append(0.3)
    elif i[12] == 'y':
        stalk_surface_below_ring.append(1.)
    elif i[12] == 's':
        stalk_surface_below_ring.append(0.)
    else:
        print('error_stalk_surface_below_ring')

#№14
stalk_color_above_ring_r = [] #Критерий цвета стебля над кольцом по RGB
stalk_color_above_ring_g = []
stalk_color_above_ring_b = []
for i in B:
    if i[13] ==  'n':
        stalk_color_above_ring_r.append(150)
        stalk_color_above_ring_g.append(75)
        stalk_color_above_ring_b.append(0)
    elif i[13] == 'b':
        stalk_color_above_ring_r.append(240)
        stalk_color_above_ring_g.append(220)
        stalk_color_above_ring_b.append(130)
    elif i[13] == 'c':
        stalk_color_above_ring_r.append(32)
        stalk_color_above_ring_g.append(2)
        stalk_color_above_ring_b.append(4)
    elif i[13] == 'g':
        stalk_color_above_ring_r.append(0)
        stalk_color_above_ring_g.append(0)
        stalk_color_above_ring_b.append(50)
    elif i[13] == 'o':
        stalk_color_above_ring_r.append(255)
        stalk_color_above_ring_g.append(165)
        stalk_color_above_ring_b.append(0)
    elif i[13] == 'p':
        stalk_color_above_ring_r.append(255)
        stalk_color_above_ring_g.append(192)
        stalk_color_above_ring_b.append(203)
    elif i[13] == 'e':
```

```
        stalk_color_above_ring_r.append(255)
        stalk_color_above_ring_g.append(0)
        stalk_color_above_ring_b.append(0)
    elif i[13] == 'w':
        stalk_color_above_ring_r.append(255)
        stalk_color_above_ring_g.append(255)
        stalk_color_above_ring_b.append(255)
    elif i[13] == 'y':
        stalk_color_above_ring_r.append(255)
        stalk_color_above_ring_g.append(255)
        stalk_color_above_ring_b.append(0)
    else:
        print('error_stalk_color_above_ring')

#№15
stalk_color_below_ring_r = [] #Критерий цвета стебля под кольцом по RGB
stalk_color_below_ring_g = []
stalk_color_below_ring_b = []
for i in B:
    if i[14] ==  'n':
        stalk_color_below_ring_r.append(150)
        stalk_color_below_ring_g.append(75)
        stalk_color_below_ring_b.append(0)
    elif i[14] == 'b':
        stalk_color_below_ring_r.append(240)
        stalk_color_below_ring_g.append(220)
        stalk_color_below_ring_b.append(130)
    elif i[14] == 'c':
        stalk_color_below_ring_r.append(32)
        stalk_color_below_ring_g.append(2)
        stalk_color_below_ring_b.append(4)
    elif i[14] == 'g':
        stalk_color_below_ring_r.append(0)
        stalk_color_below_ring_g.append(0)
        stalk_color_below_ring_b.append(50)
    elif i[14] == 'o':
        stalk_color_below_ring_r.append(255)
        stalk_color_below_ring_g.append(165)
        stalk_color_below_ring_b.append(0)
    elif i[14] == 'p':
```

```
        stalk_color_below_ring_r.append(255)
        stalk_color_below_ring_g.append(192)
        stalk_color_below_ring_b.append(203)
    elif i[14] == 'e':
        stalk_color_below_ring_r.append(255)
        stalk_color_below_ring_g.append(0)
        stalk_color_below_ring_b.append(0)
    elif i[14] == 'w':
        stalk_color_below_ring_r.append(255)
        stalk_color_below_ring_g.append(255)
        stalk_color_below_ring_b.append(255)
    elif i[14] == 'y':
        stalk_color_below_ring_r.append(255)
        stalk_color_below_ring_g.append(255)
        stalk_color_below_ring_b.append(0)
    else:
        print('error_stalk_color_below_ring')

#№16
veil_type = [] #Критерий вуали по типу
for i in B:
    if i[15] == 'p':
        veil_type.append(1)
    elif i[15] == 'u':
        veil_type.append(0)
    else:
        print('error_veil_type')

#№17
veil_color_r = [] #Критерий цвета вуали по RGB
veil_color_g = []
veil_color_b = []
for i in B:
    if i[16] ==  'n':
        veil_color_r.append(150)
        veil_color_g.append(75)
        veil_color_b.append(0)
    elif i[16] == 'o':
        veil_color_r.append(255)
        veil_color_g.append(165)
```

```
        veil_color_b.append(0)
    elif i[16] == 'w':
        veil_color_r.append(255)
        veil_color_g.append(255)
        veil_color_b.append(255)
    elif i[16] == 'y':
        veil_color_r.append(255)
        veil_color_g.append(255)
        veil_color_b.append(0)
    else:
        print('error_veil_color')

#№18
ring_number = [] #Критерий по числу колец
for i in B:
    if i[17] ==  'n':
        ring_number.append(0)
    elif i[17] == 'o':
        ring_number.append(1)
    elif i[17] == 't':
        ring_number.append(2)
    else:
        print('error_veil_type')

#№19
ring_type_cobwebby = [] #Критерий колец по типу
ring_type_evanescent = []
ring_type_flaring = []
ring_type_large = []
ring_type_pendant = []
ring_type_sheathing = []
ring_type_zone = []
for i in B:
    if i[18] ==  'c':
        ring_type_cobwebby.append(1)
        ring_type_evanescent.append(0)
        ring_type_flaring.append(0)
        ring_type_large.append(0)
        ring_type_pendant.append(0)
        ring_type_sheathing.append(0)
```

```
        ring_type_zone.append(0)
    elif i[18] == 'e':
        ring_type_cobwebby.append(0)
        ring_type_evanescent.append(1)
        ring_type_flaring.append(0)
        ring_type_large.append(0)
        ring_type_pendant.append(0)
        ring_type_sheathing.append(0)
        ring_type_zone.append(0)
    elif i[18] == 'f':
        ring_type_cobwebby.append(0)
        ring_type_evanescent.append(0)
        ring_type_flaring.append(1)
        ring_type_large.append(0)
        ring_type_pendant.append(0)
        ring_type_sheathing.append(0)
        ring_type_zone.append(0)
    elif i[18] == 'l':
        ring_type_cobwebby.append(0)
        ring_type_evanescent.append(0)
        ring_type_flaring.append(0)
        ring_type_large.append(1)
        ring_type_pendant.append(0)
        ring_type_sheathing.append(0)
        ring_type_zone.append(0)
    elif i[18] == 'p':
        ring_type_cobwebby.append(0)
        ring_type_evanescent.append(0)
        ring_type_flaring.append(0)
        ring_type_large.append(0)
        ring_type_pendant.append(1)
        ring_type_sheathing.append(0)
        ring_type_zone.append(0)
    elif i[18] == 's':
        ring_type_cobwebby.append(0)
        ring_type_evanescent.append(0)
        ring_type_flaring.append(0)
        ring_type_large.append(0)
        ring_type_pendant.append(0)
        ring_type_sheathing.append(1)
```

```
        ring_type_zone.append(0)
    elif i[18] == 'z':
        ring_type_cobwebby.append(0)
        ring_type_evanescent.append(0)
        ring_type_flaring.append(0)
        ring_type_large.append(0)
        ring_type_pendant.append(0)
        ring_type_sheathing.append(0)
        ring_type_zone.append(1)
    elif i[18] == 'n':
        ring_type_cobwebby.append(0)
        ring_type_evanescent.append(0)
        ring_type_flaring.append(0)
        ring_type_large.append(0)
        ring_type_pendant.append(0)
        ring_type_sheathing.append(0)
        ring_type_zone.append(0)
    else:
        print('error_ring_type')

#№20
spore_print_color_r = [] #Критерий цвета спор по RGB
spore_print_color_g = []
spore_print_color_b = []
for i in B:
    if i[19] == 'k':
        spore_print_color_r.append(0)
        spore_print_color_g.append(0)
        spore_print_color_b.append(0)
    elif i[19] == 'n':
        spore_print_color_r.append(150)
        spore_print_color_g.append(75)
        spore_print_color_b.append(0)
    elif i[19] == 'b':
        spore_print_color_r.append(240)
        spore_print_color_g.append(220)
        spore_print_color_b.append(130)
    elif i[19] == 'h':
        spore_print_color_r.append(210)
        spore_print_color_g.append(105)
```

```
        spore_print_color_b.append(30)
    elif i[19] == 'r':
        spore_print_color_r.append(0)
        spore_print_color_g.append(128)
        spore_print_color_b.append(0)
    elif i[19] == 'o':
        spore_print_color_r.append(255)
        spore_print_color_g.append(165)
        spore_print_color_b.append(0)
    elif i[19] == 'u':
        spore_print_color_r.append(128)
        spore_print_color_g.append(0)
        spore_print_color_b.append(128)
    elif i[19] == 'w':
        spore_print_color_r.append(255)
        spore_print_color_g.append(255)
        spore_print_color_b.append(255)
    elif i[19] == 'y':
        spore_print_color_r.append(255)
        spore_print_color_g.append(255)
        spore_print_color_b.append(0)
    else:
        print('error_spore_print_color')

#№21
population_number = [] #Критерий по распространению
population_distances = []
for i in B:
    if i[20] ==  'a':
        population_number.append(1)
        population_distances.append(1)
    elif i[20] == 'c':
        population_number.append(0.6)
        population_distances.append(0.5)
    elif i[20] == 'n':
        population_number.append(1)
        population_distances.append(0.4)
    elif i[20] == 's':
        population_number.append(0)
        population_distances.append(0.5)
```

```
    elif i[20] == 'v':
        population_number.append(0.2)
        population_distances.append(1)
    elif i[20] == 'y':
        population_number.append(0)
        population_distances.append(1)
    else:
        print('error_population')

#№22
habitat = [] #Критерий среды обитания по схожести с лесом
for i in B:
    if i[21] ==  'g':
        habitat.append(0.6)
    elif i[21] == 'l':
        habitat.append(0.8)
    elif i[21] == 'm':
        habitat.append(0.7)
    elif i[21] == 'p':
        habitat.append(0.4)
    elif i[21] == 'u':
        habitat.append(0.2)
    elif i[21] == 'w':
        habitat.append(0)
    elif i[21] == 'd':
        habitat.append(1)
    else:
        print('error_habitat')

C = {"cap_shape_middle":cap_shape_middle,
     "cap_surface":cap_surface,
     "cap_color_r":cap_color_r,
     "cap_color_g":cap_color_g,
     "cap_color_b":cap_color_b,
     "bruises":bruises,
     "odor":odor,
     "gill_attachment_x":gill_attachment_x,
#     "gill_attachment_y":gill_attachment_y,#
     "gill_spacing":gill_spacing,
     "gill_size":gill_size,
```

```
    "gill_color_r":gill_color_r,
    "gill_color_g":gill_color_g,
    "gill_color_b":gill_color_b,
    "stalk_shape":stalk_shape,
    "stalk_root_bulbous":stalk_root_bulbous,
    "stalk_root_club":stalk_root_club,
#    "stalk_root_cup":stalk_root_cup,#
    "stalk_root_equal":stalk_root_equal,
#    "stalk_root_rhizomorphs":stalk_root_rhizomorphs,#
    "stalk_root_rooted":stalk_root_rooted,
    "stalk_surface_above_ring":stalk_surface_above_ring,
    "stalk_surface_below_ring":stalk_surface_below_ring,
    "stalk_color_above_ring_r":stalk_color_above_ring_r,
    "stalk_color_above_ring_g":stalk_color_above_ring_g,
    "stalk_color_above_ring_b":stalk_color_above_ring_b,
    "stalk_color_below_ring_r":stalk_color_below_ring_r,
    "stalk_color_below_ring_g":stalk_color_below_ring_g,
    "stalk_color_below_ring_b":stalk_color_below_ring_b,
#    "veil_type":veil_type,#
    "veil_color_r":veil_color_r,
    "veil_color_g":veil_color_g,
    "veil_color_b":veil_color_b,
    "ring_number":ring_number,
#    "ring_type_cobwebby":ring_type_cobwebby,#
    "ring_type_evanescent":ring_type_evanescent,
    "ring_type_flaring":ring_type_flaring,
    "ring_type_large":ring_type_large,
    "ring_type_pendant":ring_type_pendant,
#    "ring_type_sheathing":ring_type_sheathing,#
#    "ring_type_zone":ring_type_zone,#
    "spore_print_color_r":spore_print_color_r,
    "spore_print_color_g":spore_print_color_g,
    "spore_print_color_b":spore_print_color_b,
    "population_number":population_number,
    "population_distances":population_distances,
    "habitat":habitat}

D = pandas.DataFrame(C)
#print(D)
"""
```

```
X = pandas.DataFrame(C)
print(X)
"""
E=(D-D.mean())/D.std()
X=E.values
#print(X)

Y = dataset[1:, 0]
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
dummy_y = to_categorical(encoded_Y)
# Тестирование при различных параметрах


def test_num_of_neurons():
    loss = []
    val_loss = []
    acc = []
    val_acc = []
    vect_num_neurs = (4, 8, 12, 16, 20)
    # 2 слоя
    for i in vect_num_neurs:
        model = Sequential()
        model.add(Dense(40, activation='relu'))
        model.add(Dense(i, activation='relu'))
        model.add(Dense(i, activation='relu'))
        model.add(Dense(2, activation='softmax'))
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        H = model.fit(X, dummy_y, epochs=20, batch_size=10, validation_split=0.1)
        loss.append(H.history['loss'][-1])
        val_loss.append(H.history['val_loss'][-1])
        acc.append(H.history['acc'][-1])
        val_acc.append(H.history['val_acc'][-1])
    draw_test(vect_num_neurs, 'Number of neurons', loss, val_loss, acc, val_acc)

def test_num_of_layers():
    loss = []
    val_loss = []
    acc = []
```

```
    val_acc = []
    vect_num_layers = (1, 2, 3)
    for i in vect_num_layers:
        model = Sequential()
        model.add(Dense(40, activation='relu'))
        for j in range(1, i):
            model.add(Dense(12, activation='relu'))
        model.add(Dense(2, activation='softmax'))
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        H = model.fit(X, dummy_y, epochs=20, batch_size=10, validation_split=0.1)
        loss.append(H.history['loss'][-1])
        val_loss.append(H.history['val_loss'][-1])
        acc.append(H.history['acc'][-1])
        val_acc.append(H.history['val_acc'][-1])
    draw_test(vect_num_layers, 'Number of layers', loss, val_loss, acc, val_acc)

def test_epochs():
    loss = []
    val_loss = []
    acc = []
    val_acc = []
    vect_epochs = range(10, 30, 5)
    for i in vect_epochs:
        model = Sequential()
        model.add(Dense(40, activation='relu'))
        model.add(Dense(12, activation='relu'))
        model.add(Dense(12, activation='relu'))
        model.add(Dense(2, activation='softmax'))
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        H = model.fit(X, dummy_y, epochs=i, batch_size=10, validation_split=0.1)
        loss.append(H.history['loss'][-1])
        val_loss.append(H.history['val_loss'][-1])
        acc.append(H.history['acc'][-1])
        val_acc.append(H.history['val_acc'][-1])
    draw_test(vect_epochs, 'Number of epochs', loss, val_loss, acc, val_acc)

def test_batch_size():
    loss = []
    val_loss = []
    acc = []
```

```
    val_acc = []
    vect_batch = range(5,30,5)
    for i in vect_batch:
        model = Sequential()
        model.add(Dense(40, activation='relu'))
        model.add(Dense(12, activation='relu'))
        model.add(Dense(12, activation='relu'))
        model.add(Dense(2, activation='softmax'))
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        H = model.fit(X, dummy_y, epochs=15, batch_size=i, validation_split=0.1)
        loss.append(H.history['loss'][-1])
        val_loss.append(H.history['val_loss'][-1])
        acc.append(H.history['acc'][-1])
        val_acc.append(H.history['val_acc'][-1])
    draw_test(vect_batch, 'Batch size', loss, val_loss, acc, val_acc)


def validation_test():
    loss = []
    val_loss = []
    acc = []
    val_acc = []
    vect_validation = []
    for i in range(1, 8):
        vect_validation.append(i*0.1)
        model = Sequential()
        model.add(Dense(40, activation='relu'))
        model.add(Dense(12, activation='relu'))
        model.add(Dense(12, activation='relu'))
        model.add(Dense(2, activation='softmax'))
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        H = model.fit(X, dummy_y, epochs=15, batch_size=20, validation_split=i*0.1)
        loss.append(H.history['loss'][-1])
        val_loss.append(H.history['val_loss'][-1])
        acc.append(H.history['acc'][-1])
        val_acc.append(H.history['val_acc'][-1])
    draw_test(vect_validation, 'Validation_split', loss, val_loss, acc, val_acc)

def draw_test(arg, label, loss, val_loss, acc, val_acc):
    plt.plot(arg, loss, 'y', label='Training loss')
```

```
    plt.plot(arg, val_loss, 'k', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel(label)
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()
    plt.plot(arg, acc, 'y', label='Training acc')
    plt.plot(arg, val_acc, 'k', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel(label)
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

def best_model():
    model = Sequential()
    model.add(Dense(40, activation='relu'))
    model.add(Dense(12, activation='relu'))
    model.add(Dense(12, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    H = model.fit(X, dummy_y, epochs=15, batch_size=20, validation_split=0.1)
    loss = H.history['loss']
    val_loss = H.history['val_loss']
    acc = H.history['acc']
    val_acc = H.history['val_acc']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, 'y', label='Training loss')
    plt.plot(epochs, val_loss, 'k', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()
    plt.plot(epochs, acc, 'y', label='Training acc')
    plt.plot(epochs, val_acc, 'k', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
```

```
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

def first_model():
    model = Sequential()
    model.add(Dense(40, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    H = model.fit(X, dummy_y, epochs=20, batch_size=10, validation_split=0.1)
    loss = H.history['loss']
    val_loss = H.history['val_loss']
    acc = H.history['acc']
    val_acc = H.history['val_acc']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, 'y', label='Training loss')
    plt.plot(epochs, val_loss, 'k', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()
    plt.plot(epochs, acc, 'y', label='Training acc')
    plt.plot(epochs, val_acc, 'k', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

#first_model()
test_num_of_neurons()
#test_num_of_layers()
#test_epochs()
#test_batch_size()
#validation_test()
#best_model()
```