

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

ОТЧЕТ
по индивидуальному заданию
по дисциплине «Искусственные нейронные сети»
Тема: Классификация животного по его описанию

Студентка гр. 7381

Алясова А.Н.

Студентка гр. 7381

Кушкочева А.О.

Преподаватель

Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Датасет булевых значений. Включает в себя информацию о 101 животном из зоопарка, которые относятся к 7 классам. Задача заключается в классификации животного по его описанию.

Описание датасета.

Количество экземпляров: 101

Количество атрибутов: 18 (имя животного, 15 булевых значений, 2 цифры)

Таблица 1 - Информация об атрибутах

	Имя атрибута	Тип
1	animal name	unique for each instance
2	hair	boolean
3	feathers	boolean
4	eggs	boolean
5	milk	boolean
6	airborne	boolean
7	aquatic	boolean
8	predator	boolean
9	toothed	boolean
10	backbone	boolean
11	breathes	boolean
12	venomous	boolean
13	fins	boolean
14	legs	Numeric (set of values: {0,2,4,5,6,8})
15	tail	boolean
16	domestic	boolean
17	catsize	boolean
18	type	Numeric (integer values in range [1,7])

Таблица 2 - Информация о типах

	Имя атрибута	Информация
1	animal name	имя животного
2	hair	наличие шерсти
3	feathers	наличие перьев
4	eggs	дает яйца
5	milk	дает молоко
6	airborne	летает
7	aquatic	плавает
8	predator	хищник
9	toothed	зубчатый
10	backbone	позвоночное
11	breathes	наземное
12	venomous	ядовитый
13	fins	наличие плавников
14	legs	количество конечностей
15	tail	наличие хвоста
16	domestic	одомашненный
17	catsize	размер животного больше или примерно с тигром
18	type	принадлежность к классу

Таблица 3 - Наборы животных

Номер класса	Количество животных	Перечень животных
1	41	aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruitbat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, lynx, mink, mole, mongoose, opossum, oryx, platypus, polecat, pony, porpoise, puma, pussycat, raccoon, reindeer, seal, sealion, squirrel, vampire, vole, wallaby, wolf

2	20	chicken, crow, dove, duck, flamingo, gull, hawk, kiwi, lark, ostrich, parakeet, penguin, pheasant, rhea, skimmer, skua, sparrow, swan, vulture, wren
3	5	pitviper, seasnake, slowworm, tortoise, tuatara
4	13	bass, carp, catfish, chub, dogfish, haddock, herring, pike, piranha, seahorse, sole, stingray, tuna
5	4	frog, frog, newt, toad
6	8	flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp
7	10	clam, crab, crayfish, lobster, octopus, scorpion, seawasp, slug, starfish, worm

Разделение ответственности.

Алясова А.Н. – Создание, тестирование модели.

Кушкочева А.О. – Написание callback-ов, обработка данных.

Ход работы.

1) Обработка данных

- Данные не нуждаются в очистке: нет данных выходящих из диапазона, нет пропусков в данных, дубликатов. Для этого краткого анализа не были использованы возможности pandas. Его использование является излишним, т.к. объем данных небольшой.
- Отделили данные для обучения от целей, которые отвечают за принадлежность животного к одному из 7 классов.

```
x_data = zoo[:, 1:-1].astype(float)
y_data = zoo[:, -1:]
```

- Рассмотрим столбец, описывающий количество ног и представляющий значения из множества $\{0, 2, 4, 5, 6, 8\}$. Модель нейронной сети может неправильно определить смысл этих чисел и посчитает, что они находятся в каком-то особом порядке. Для решения этой проблемы воспользуемся методом One Hot Encoder.

```
columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(), [12])],
                                      remainder='passthrough')
x_data = np.array(columnTransformer.fit_transform(x_data))
```

Сравним результаты точности без этого преобразования и с ним.

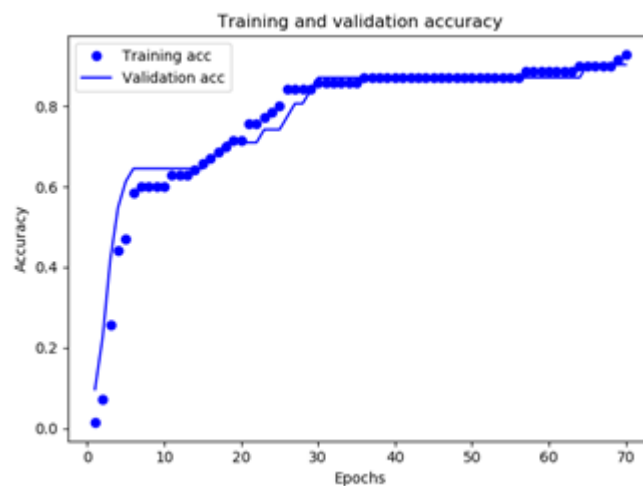


Рисунок 1 – График точности без преобразования (Accuracy: 94.29%)

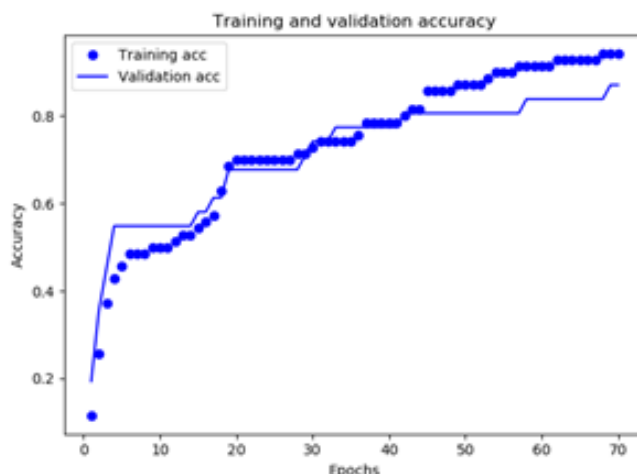


Рисунок 2 – График точности с преобразованием (Accuracy: 94.29%)

Модель, используемая при тестировании:

```
model = Sequential()
model.add(Dense(16, activation='relu'))
model.add(Dense(7, activation='softmax'))
```

- Цели описаны номерами классов {1, 2, 3, 4, 5, 6, 7}. Следовательно, перед нами стоит та же проблема, что и со столбцом, описывающим количество ног.

```
onehot_encoder = OneHotEncoder()
y_data = onehot_encoder.fit_transform(y_data).toarray()
```

- Разобьем данные по тренировочным (80%) и тестируемым (20%) с использованием `random_state` для перетасовки данных.

```
train_x, test_x, train_y, test_y = train_test_split(x_data, y_data, test_size=0.2,
                                                    random_state=42, stratify=y_data)
```

2) Использование callback'ов.

- Keras поддерживает раннюю остановку обучения с помощью callback'а `earlystopping`. Обучение остановится, когда выбранный показатель эффективности перестанет улучшаться, в нашем случае минимальный показатель `'val_loss'`. Но возможен такой исход событий, что модель попала в плато, а обучение уже остановилось. Для решения этой проблемы воспользовались задержкой остановки, при которой убеждаемся в том, что значение потери не увеличиваются.

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=200)
```

- Требуется дополнительный обратный вызов, который сохранит лучшую модель, наблюдаемую во время обучения, для последующего использования. Воспользовались `modelcheckpoint`.

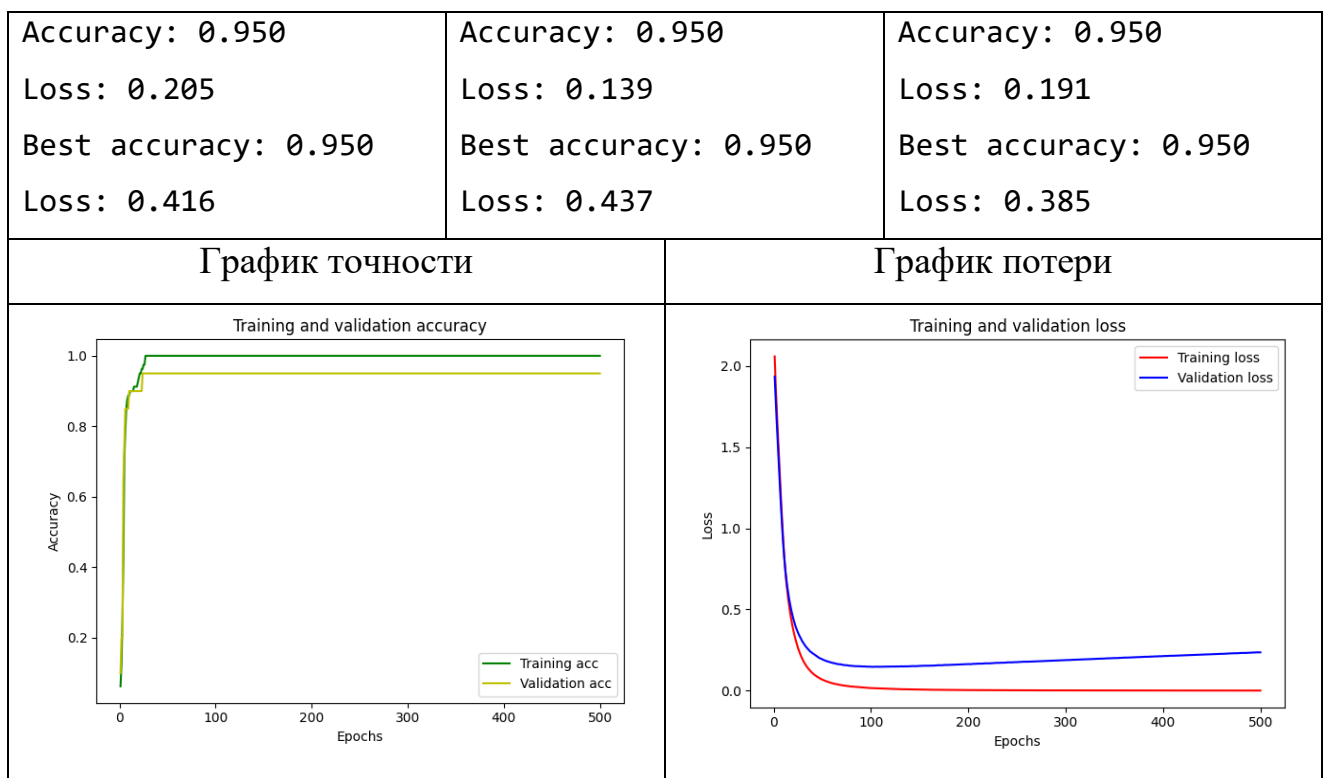
```
mc = ModelCheckpoint(filepath='best_model.h5', monitor='val_accuracy',  
mode='max', verbose=1, save_best_only=True)
```

3) Создание архитектуры модели

Начнем с простой архитектуры модели. Добавим один скрытый слой Dense.

```
model = Sequential()  
model.add(Dense(50, activation='relu'))  
model.add(Dense(7, activation='softmax'))
```

Протестируем ее на 3-ех запусках.



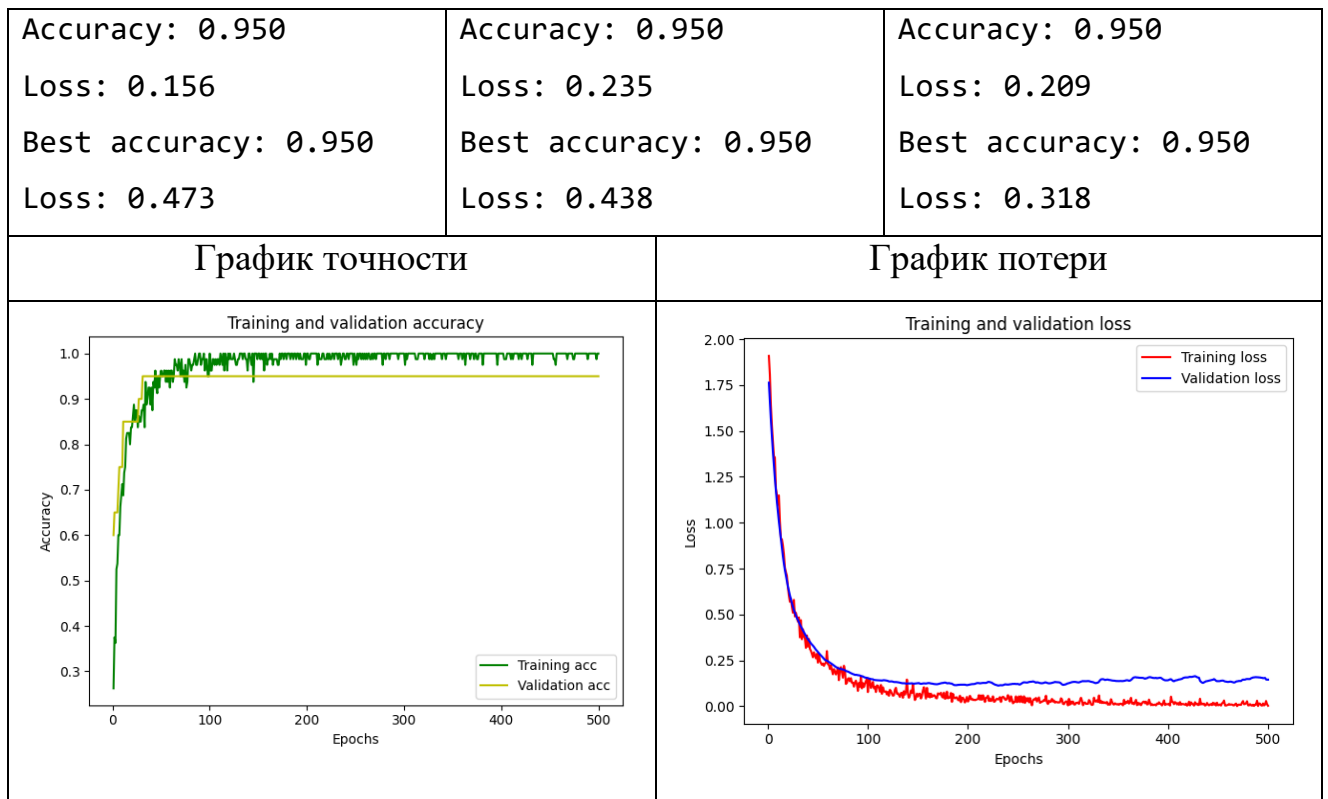
По результатам запусков можно сказать, что данная модель дает хороший результат 95 %. Поэтому будем расширять сеть, пока не найдем более выгодное строение сети.

Для начала, чтобы избежать переобучение, добавим слой Dropout.

```

model = Sequential()
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

```



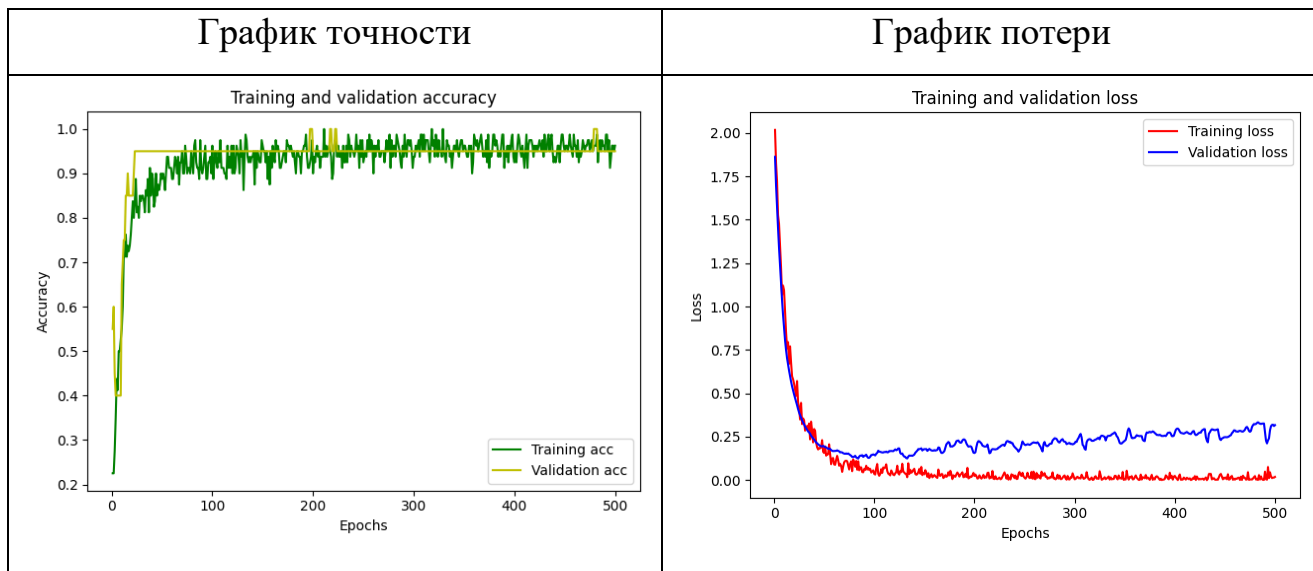
Как видно по графикам на результат не улучшился, значит будем дальше расширять сеть, для этого добавим еще один скрытый слой Dense.

```

model = Sequential()
model.add(Dense(50, activation='relu'))
model.add(Dense(35, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

```

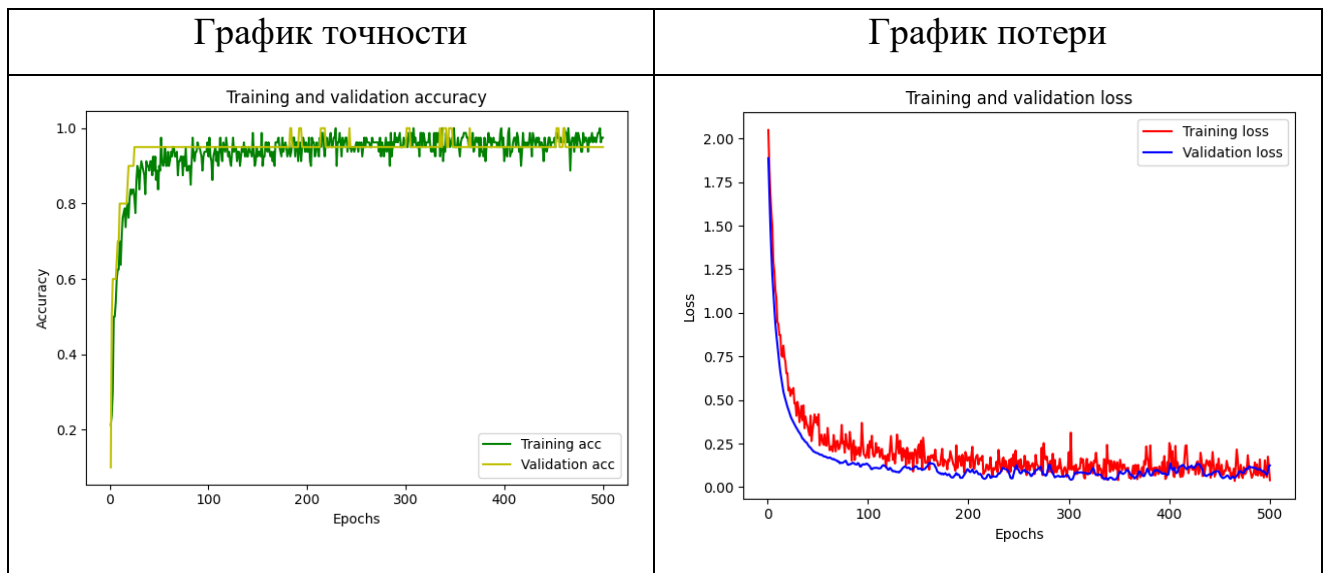
Accuracy: 0.950 Loss: 0.294 Best accuracy: 0.950 Loss: 0.327	Accuracy: 0.950 Loss: 0.286 Best accuracy: 0.950 Loss: 0.353	Accuracy: 0.950 Loss: 0.098 Best accuracy: 1.000 Loss: 0.096
-----------------------------------------------------------------------	-----------------------------------------------------------------------	-----------------------------------------------------------------------



Вот мы уже достигли 100% точности. Попробуем сделать этот результат почаще. Для этого добавим еще один скрытый слой Dropout со значением 0.2. Таким образом мы выполним отсев, то есть в приведенном ниже примере мы добавляем новый слой Dropout между входным и первым скрытым слоем. Частота выпадения установлена 20%, то есть каждый пятый вход будет случайным образом исключен из каждого цикла обновления.

```
model = Sequential()
model.add(Dropout(0.2))
model.add(Dense(50, activation='relu'))
model.add(Dense(35, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

Accuracy: 0.950	Accuracy: 1.000	Accuracy: 0.950
Loss: 0.080	Loss: 0.036	Loss: 0.057
Best accuracy: 0.950	Best accuracy: 1.000	Best accuracy: 1.000
Loss: 0.342	Loss: 0.067	Loss: 0.058



Как мы видим, результаты улучшились. Появилось больше моделей с точностью 100%.

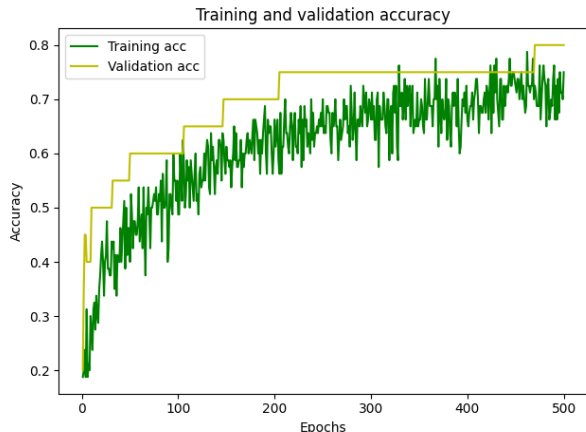
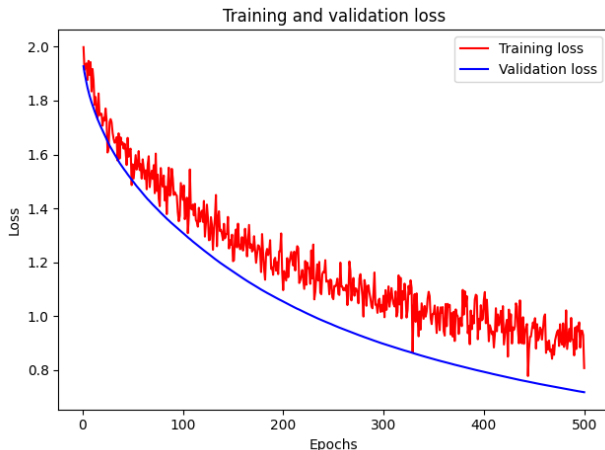
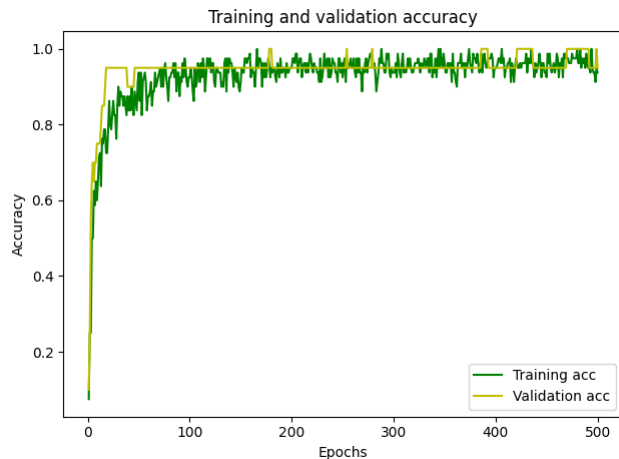
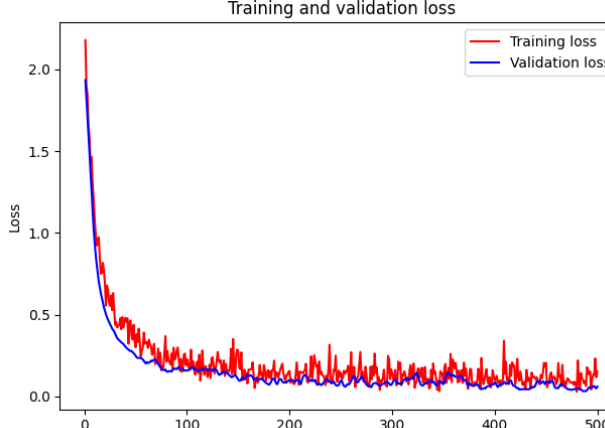
Далее расширяем сеть, но становится очевидно, что прибавление слоев не влияет на результат поэтому остановимся на данной архитектуре.

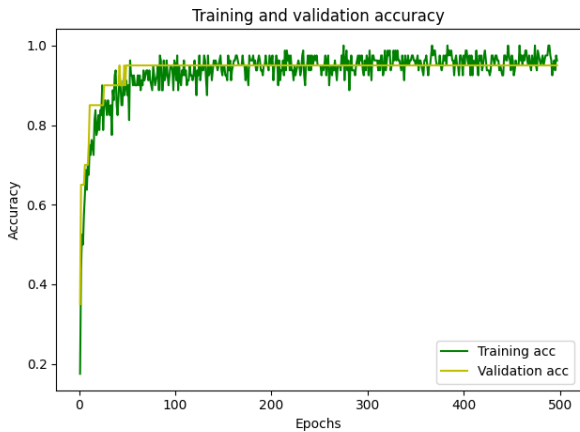
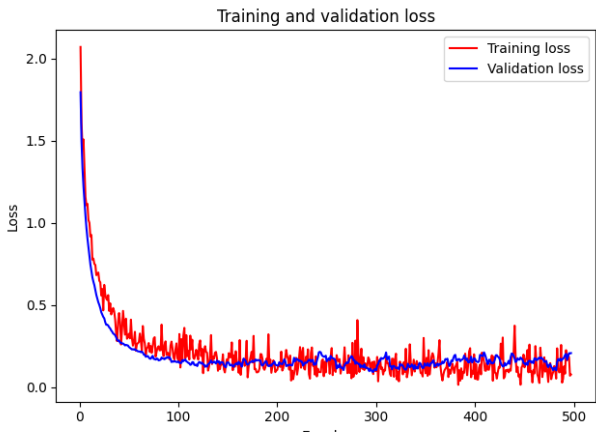
```
model = Sequential()
model.add(Dropout(0.2))
model.add(Dense(50, activation='relu'))
model.add(Dense(35, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

Наша модель состоит из 4-рех скрытых слоев, первый скрытый слой Dropout со значением 0.2 мы используем для отсева данных. Дальше идут два слоя Dense со значениями 50 и 35. Как показано выше, нам не нужно добавлять еще слои Dense, так как это не приводит к существенным изменениям. Далее мы используем скрытый слой Dropout со значением 0.5, который нужен нам для избежания переобучения. С такой архитектурой данная модель отлично справляется со своей задачей.

Выбор оптимизатора.

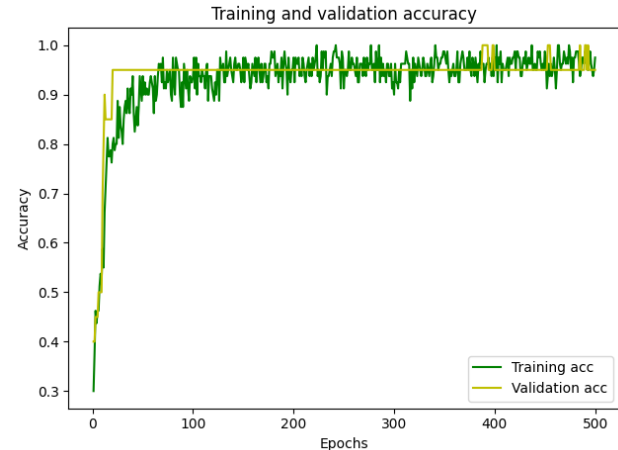
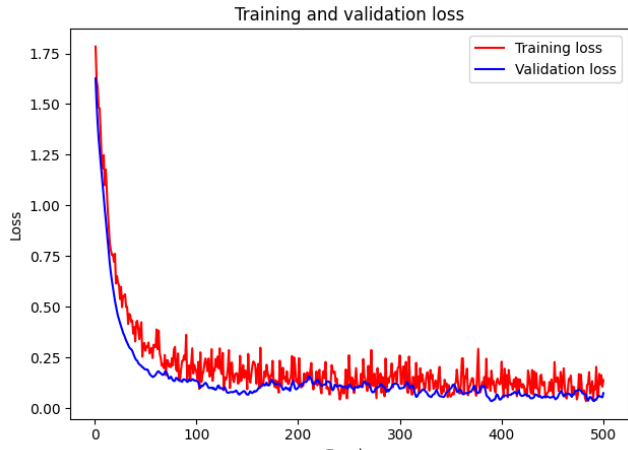
1-ый запуск

График точности	График потери
Оптимизатор 'Adagrad'	
 <p>Training and validation accuracy</p>	 <p>Training and validation loss</p>
Accuracy: 0.800 Loss: 0.718	Best accuracy: 0.800 Loss: 0.738
Оптимизатор 'Adam'	
 <p>Training and validation accuracy</p>	 <p>Training and validation loss</p>
Accuracy: 0.950 Loss: 0.059	Best accuracy: 1.000 Loss: 0.079

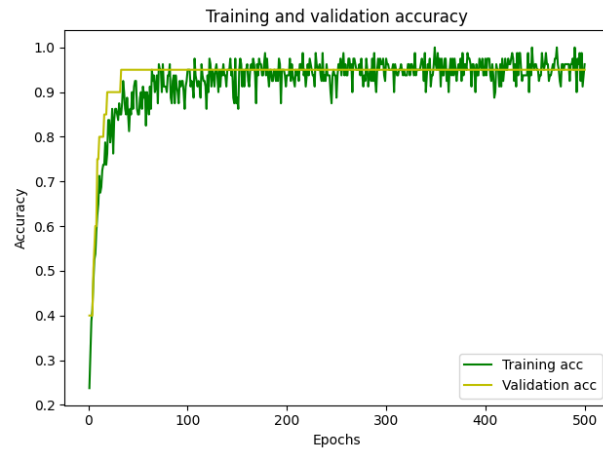
Оптимизатор 'RMSprop'	
	
Accuracy: 0.950 Loss: 0.207	Best accuracy: 0.950 Loss: 0.270

Как мы видим, оптимизатор 'Adagrad' показал плохие результаты. Проведем обучение еще 2 раза с оптимизаторами 'Adam' и 'RMSprop', чтобы выбрать наиболее подходящий оптимизатор.

2-ой запуск

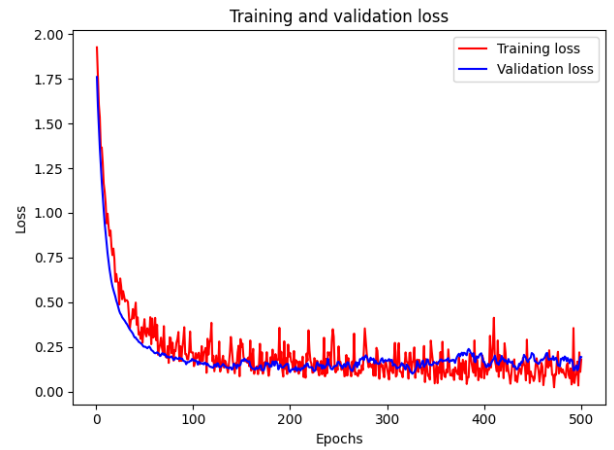
График точности	График потери
Оптимизатор 'Adam'	
	
Accuracy: 0.950 Loss: 0.075	Best accuracy: 1.000 Loss: 0.048

Оптимизатор 'RMSprop'



Accuracy: 0.950

Loss: 0.193



Best accuracy: 0.950

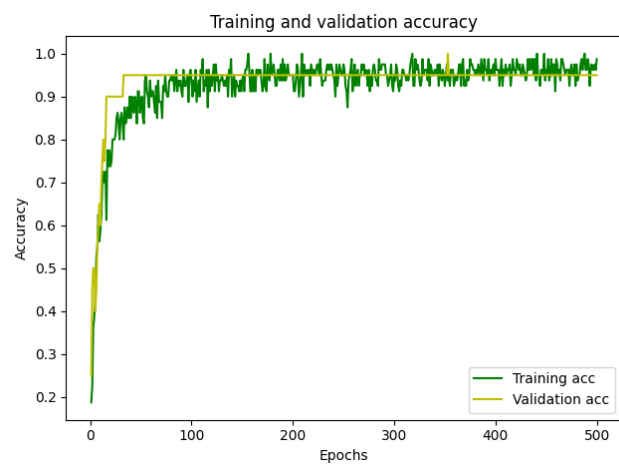
Loss: 0.367

3-ий запуск

График точности

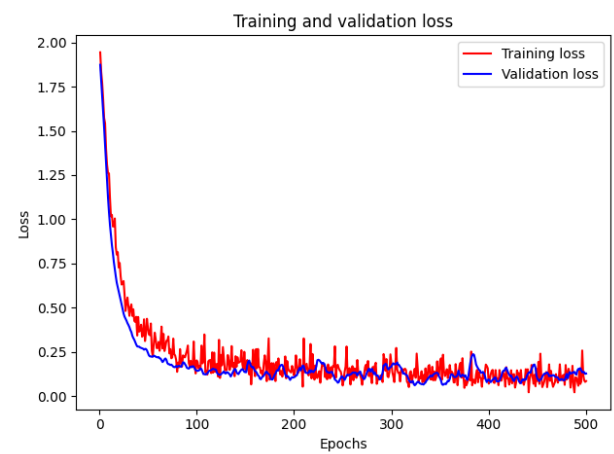
График потери

Оптимизатор 'Adam'



Accuracy: 0.950

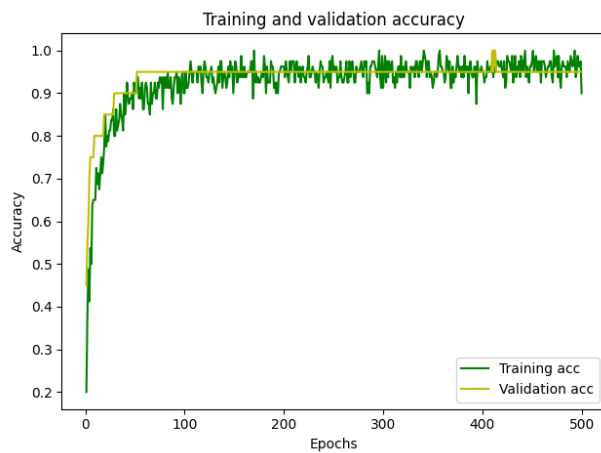
Loss: 0.128



Best accuracy: 1.000

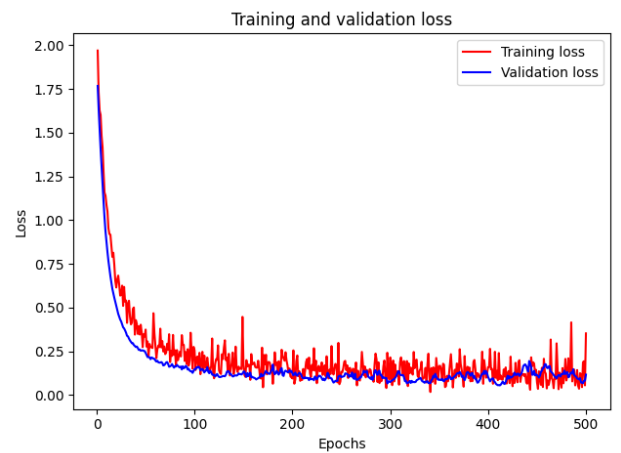
Loss: 0.064

Оптимизатор 'RMSprop'



Accuracy: 0.950

Loss: 0.116



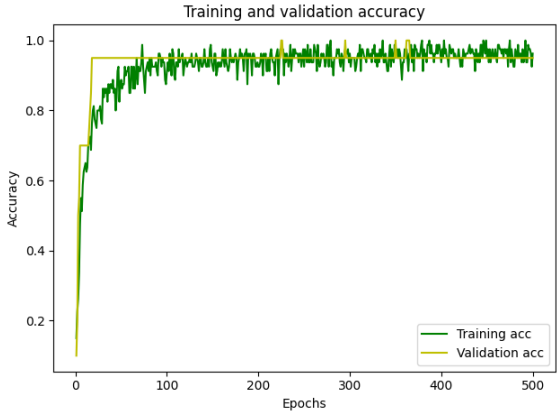
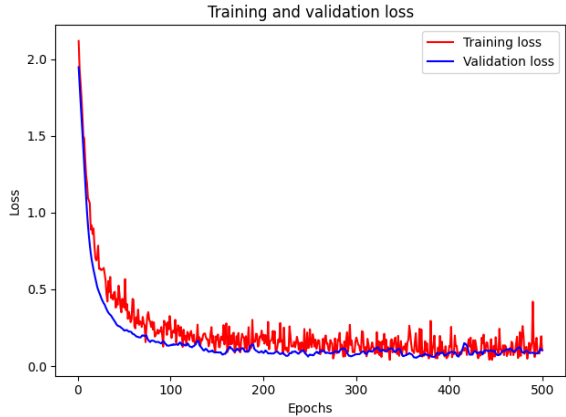
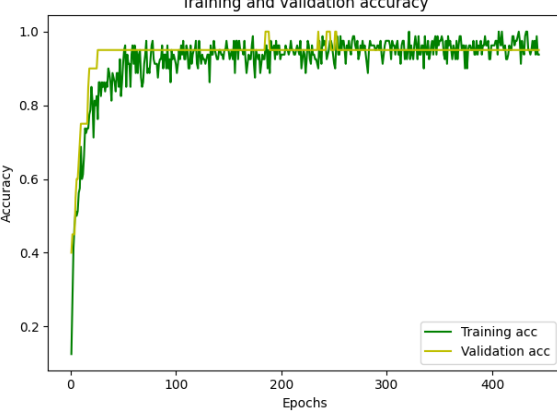
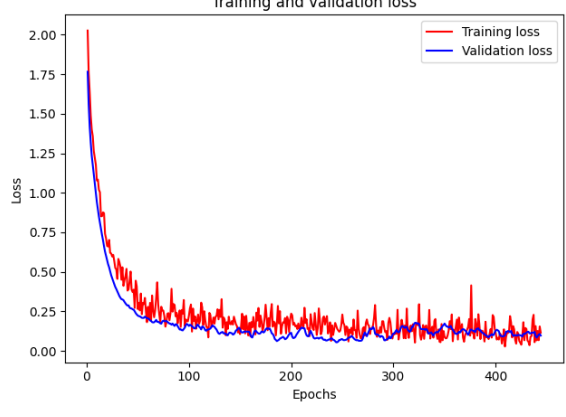
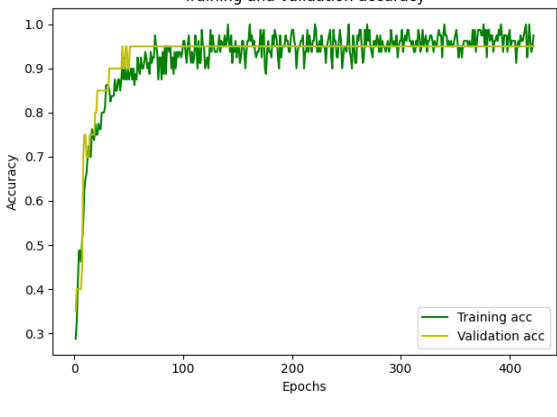
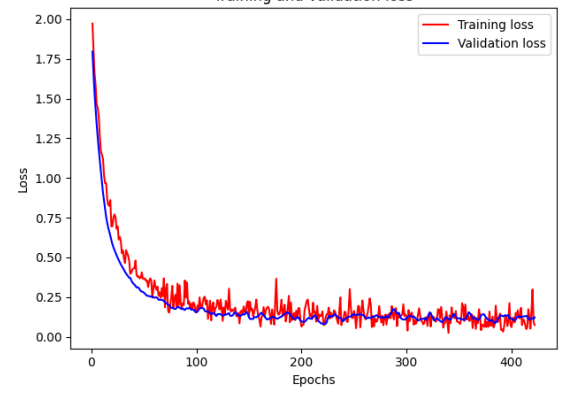
Best accuracy: 1.000

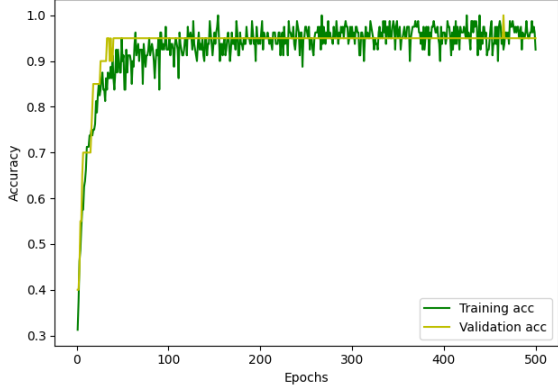
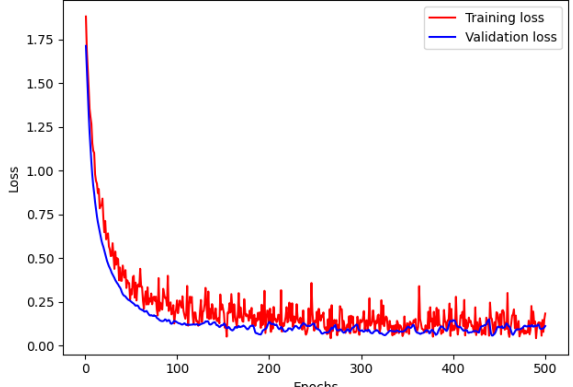
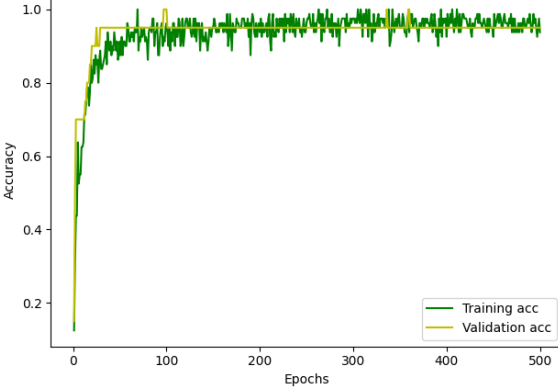
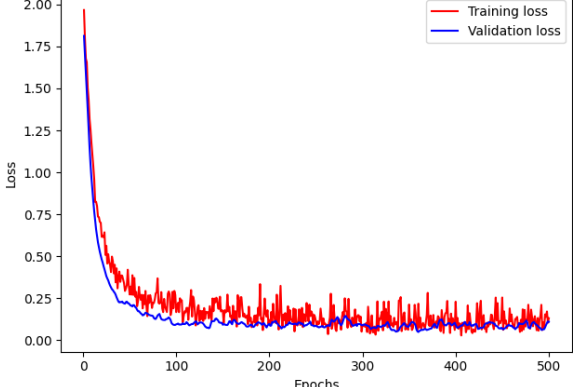
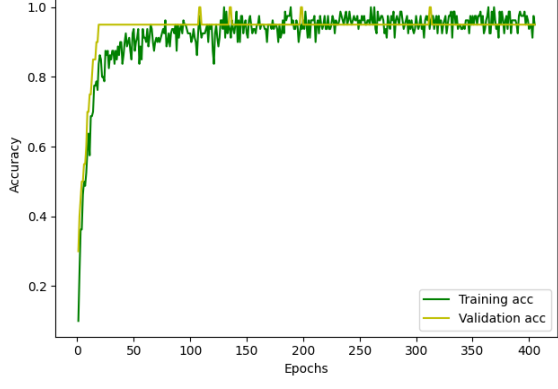
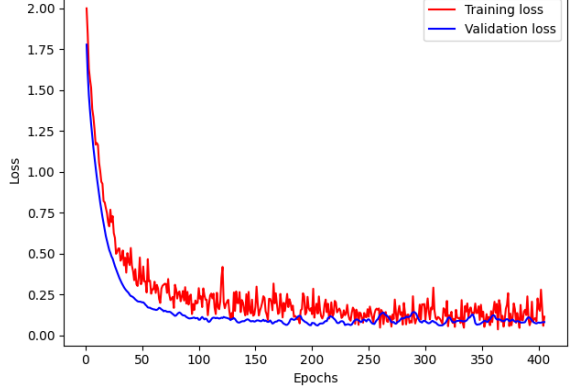
Loss: 0.055

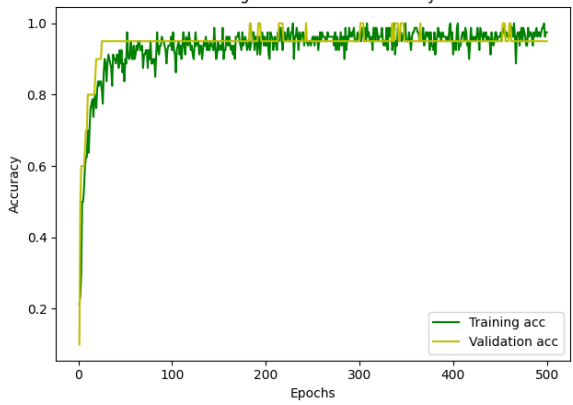
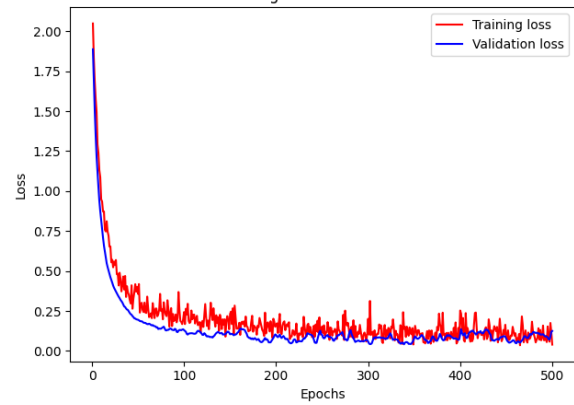
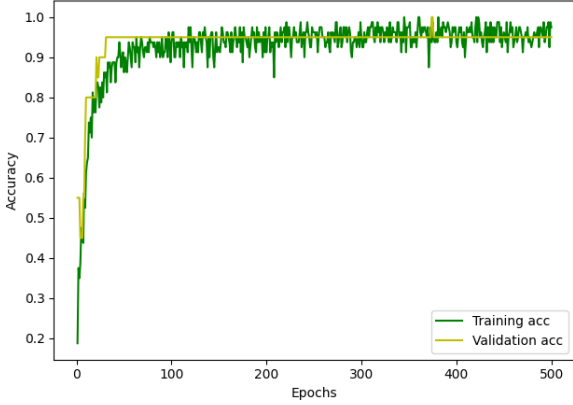
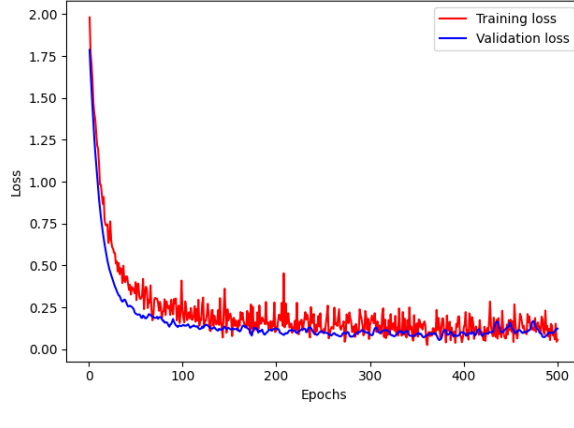
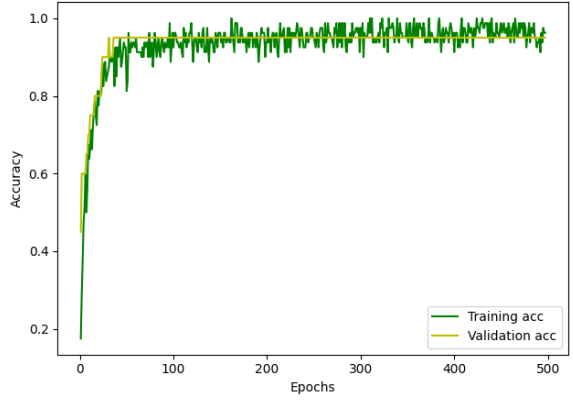
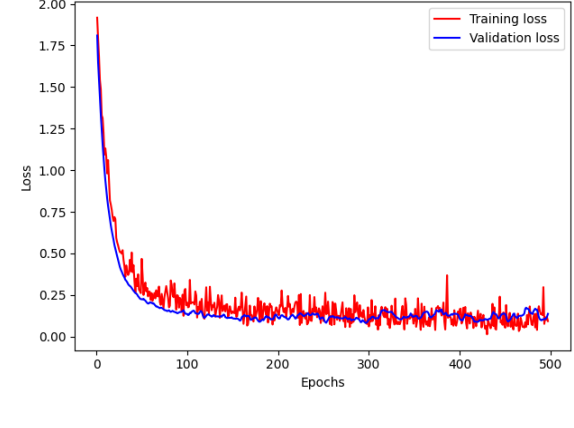
На основании этого выбираем оптимизатор 'Adam'. Так как он более стабилен и чаще выдает точность 100% при меньших потерях.

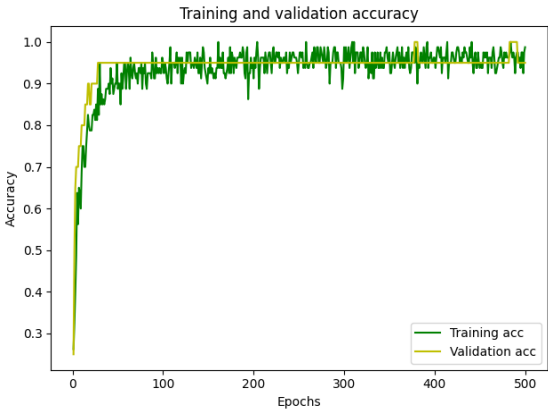
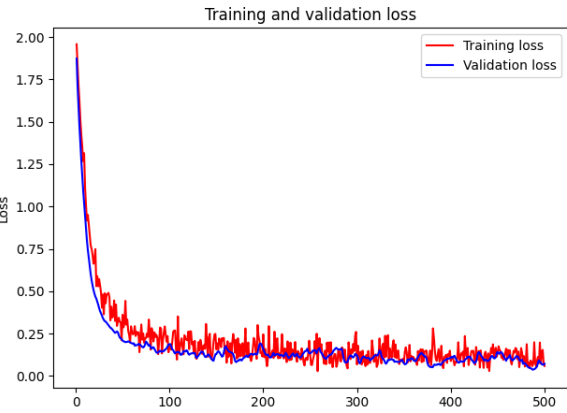
Проверка стабильности модели.

Запустим нашу модель 10 раз, чтобы понять является ли она устойчивой.

№	График точности	График потери
	Оптимизатор Adam	
1	 <p>Accuracy: 0.950 Loss: 0.103</p>	 <p>Best accuracy: 1.000 Loss: 0.066</p>
2	 <p>Accuracy: 0.950 Loss: 0.097</p>	 <p>Best accuracy: 1.000 Loss: 0.069</p>
3		

	<p>Accuracy 0.950</p> <p>Loss 0.122</p>	<p>Best accuracy: 0.950</p> <p>Loss: 0.311</p>
4	<p>Training and validation accuracy</p> 	<p>Training and validation loss</p> 
	<p>Accuracy 0.950</p> <p>Loss 0.112</p>	<p>Best accuracy: 1.000</p> <p>Loss: 0.061</p>
5	<p>Training and validation accuracy</p> 	<p>Training and validation loss</p> 
	<p>Accuracy: 0.950</p> <p>Loss: 0.110</p>	<p>Best accuracy: 1.000</p> <p>Loss: 0.093</p>
6	<p>Training and validation accuracy</p> 	<p>Training and validation loss</p> 
	<p>Accuracy 0.950</p> <p>Loss 0.080</p>	<p>Best accuracy: 1.000</p> <p>Loss: 0.086</p>

7	<p>Training and validation accuracy</p>  <p>Accuracy 0.950 Loss 0.125</p>	<p>Training and validation loss</p>  <p>Best accuracy: 1.000 Loss: 0.058</p>
8	<p>Training and validation accuracy</p>  <p>Accuracy 0.950 Loss 0.123</p>	<p>Training and validation loss</p>  <p>Best accuracy: 1.000 Loss: 0.054</p>
9	<p>Training and validation accuracy</p>  <p>Accuracy 0.950 Loss 0.137</p>	<p>Training and validation loss</p>  <p>Best accuracy: 0.950 Loss: 0.355</p>

10	 <p>Training and validation accuracy</p> <p>Accuracy</p> <p>Epochs</p> <p>Training acc</p> <p>Validation acc</p>	 <p>Training and validation loss</p> <p>Loss</p> <p>Epochs</p> <p>Training loss</p> <p>Validation loss</p>
	<p>Accuracy 0.950</p> <p>Loss 0.072</p>	<p>Best accuracy: 1.000</p> <p>Loss: 0.052</p>

Анализ результирующей модели.

Модель с наилучшей точностью показывает 100% в 8/10 случаях.

Модель с ранней остановкой показывает 95% в 10/10 случаях.

Поэтому можно сказать, что модель является стабильной и отлично справляется с предсказанием класса животного.

На этапе подготовки данных возникли трудности с нормировкой данных, которые были решены с использованием one hot encoder. На этапе обучения модель могла попасть в плато, следовательно, выдавала не самый лучший результат. Данная проблема была решена при помощи использования callback'ов. Относительно результатов точности можно сделать вывод, что модель не нуждается в улучшении.

Выводы.

В ходе выполнения индивидуального задания были закреплены знания по обработке данных и построения персептронов. Была создана стабильная модель, которая выдает отличные результаты 95-100%. Также ознакомились с встроенными callback'ами из библиотеки Keras.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model

EPOCHS = 500
BATCHSIZE = 8

optimizers = [optimizers.Adam(),
               optimizers.RMSprop()]

# Загрузка данных
def load_data():
    dataframe = pd.read_csv("zoo.csv")
    zoo = dataframe.values
    # Деление данных для обучения от целей
    x_data = zoo[:, 1:-1].astype(float)
    y_data = zoo[:, -1:]
    # Кодирование целей
    onehot_encoder = OneHotEncoder()
    y_data = onehot_encoder.fit_transform(y_data).toarray()
    # Нормализация данных
    columnTransformer = ColumnTransformer([('encoder',
OneHotEncoder(),[12])],
                                         remainder='passthrough')
    x_data = np.array(columnTransformer.fit_transform(x_data))
    # Деление на тренировочный и тестовый датасеты
    train_x, test_x, train_y, test_y = train_test_split(x_data, y_data,
test_size=0.2,
                                                         random_state=42,
stratify=y_data)
    return train_x, test_x, train_y, test_y
```

```

# Создание модели
def build_model():
    model = Sequential()
    model.add(Dropout(0.2))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(35, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(7, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Создание модели для тестирования с разными оптимизаторами
def build_model(opt):
    model = Sequential()
    model.add(Dropout(0.25))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(35, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(7, activation='softmax'))

    model.compile(optimizer=opt, loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Тестирование модели
def train_model():
    model = build_model()
    # Обучение сети
    es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=200)
    mc = ModelCheckpoint(filepath='best_model.h5',
monitor='val_accuracy',
mode='max', verbose=1, save_best_only=True)
    history = model.fit(train_x, train_y, validation_data=(test_x,
test_y),
epochs=EPOCHS, batch_size=BATCHSIZE,
callbacks=[es, mc])
    create_graphics(history)

```

```

test_loss, test_acc1 = model.evaluate(test_x, test_y)
print("Accuracy: %.3f" % (test_acc1))
print("Loss: %.3f" % (test_loss))

best_model = load_model('best_model.h5')
test_loss, test_acc2 = best_model.evaluate(test_x, test_y, verbose=0)
print("Best accuracy: %.3f" % (test_acc2))
print("Loss: %.3f" % (test_loss))

if(test_acc1 == test_acc2):
    return model
return best_model

# Тестирование модели с разными оптимизаторами
def test(opt):
    # Вывод настроек метода
    opt_config = opt.get_config()
    print("Researching with optimizer ")
    print(opt_config)

    model = build_model(opt)
    es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=200)
    mc = ModelCheckpoint(filepath='best_model.hdf5',
monitor='val_accuracy',
mode='max', verbose=1, save_best_only=True)

    # Обучение сети
    history = model.fit(train_x, train_y, validation_data=(test_x,
test_y),
epochs=EPOCHS, batch_size=BATCHSIZE,
callbacks=[es, mc])
    create_graphics(history)
    test_loss, test_acc1 = model.evaluate(test_x, test_y)
    result["%s" % (opt_config)] = test_acc1
    print("Accuracy: %.3f" % (test_acc1))
    print("Loss: %.3f" % (test_loss))

    best_model = load_model('best_model.hdf5')
    test_loss, test_acc2 = best_model.evaluate(test_x, test_y, verbose=0)
    print("Best accuracy: %.3f" % (test_acc2))
    print("Loss: %.3f" % (test_loss))

    if(test_acc1 == test_acc2):

```

```
        return model
    return best_model
```

```
# Создание графиков
```

```
def create_graphics(history):
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    epochs = range(1, len(loss) + 1)
    print(len(loss))
    # График потерь
    plt.plot(epochs, loss, 'r', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()
    # График точности
    plt.plot(epochs, acc, 'g', label='Training acc')
    plt.plot(epochs, val_acc, 'y', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

```
if __name__ == '__main__':
    train_x, test_x, train_y, test_y = load_data()
    num = '2'
    if (num == '1'):
        train_model()
    if (num == '2'):
        result = dict()
        for opt in optimizers:
            test(opt)
    # Результаты тестирования
    for res in result:
        print("%s: %s" % (res, result[res]))
```