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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Искусственные нейронные сети»

Тема: Бинарная классификация отраженных сигналов радара

Студентка гр. 8383 _____ Максимова А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Реализовать классификацию между камнями (R) и металлическими цилиндрами (M) на основе данных об отражении сигналов радара от поверхностей.

60 входных значений показывают силу отражаемого сигнала под определенным углом. Входные данные нормализованы и находятся в промежутке от 0 до 1.

Задачи.

- Ознакомиться с задачей бинарной классификации
- Загрузить данные
- Создать модель ИНС в tf.Keras
- Настроить параметры обучения
- Обучить и оценить модель
- Изменить модель и провести сравнение. Объяснить результаты

Требования.

1. Изучить влияние количества нейронов на слое на результат обучения модели.
2. Изучить влияние количества слоев на результат обучения модели.
3. Построить графики ошибки и точности в ходе обучения.
4. Провести сравнение полученных сетей, объяснить результат.

Основные теоретические положения.

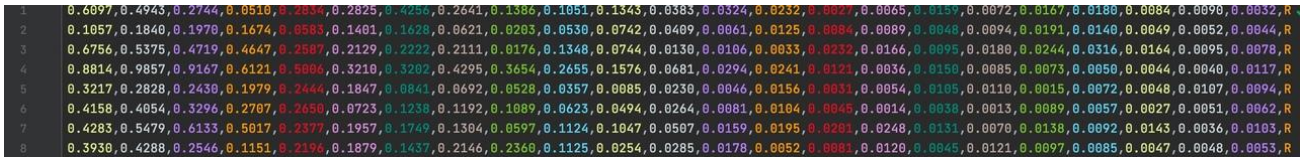
Задача бинарной классификации: задача классификации элементов заданного множества в две группы (предсказание, какой из групп принадлежит каждый элемент множества) на основе правил классификации.

Выполнение работы.

1. Были импортированы все необходимые для работы классы и функции.

```
import pandas
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense #полносвязанный слой
from tensorflow.keras.models import Sequential #сеть прямого распространения
from sklearn.preprocessing import LabelEncoder #для кодирования из строк в целочисленные значения
```

2. Был скачан и помещен в директорию проекта файл "sonar.all-data", содержащий набор входных данных, и переименован в "sonar.csv". Каждая строка файла содержит 60 входных параметров и 1 выходной.



3. Набор данных был загружен напрямую с помощью pandas. После данные были разделены на входные параметры: X и выходные параметры: Y.

```
dataframe = pandas.read_csv("sonar.csv", header=None)
dataset = dataframe.values
X = dataset[:, 0:60].astype(float)
Y = dataset[:, 60]
```

4. Выходные данные, представленные строками "R" и "M", были переведены в целочисленные значения 0 и 1 с помощью кода, представленного ниже.

```
encoder = LabelEncoder()
encoder_Y = encoder.fit_transform(Y)
print(encoder_Y)
```

Результат:

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

5. После была создана модель ИНС прямого распространения, состоящая из трех слоев: первый (входной) слой содержит 60 нейронов, второй (скрытый) слой содержит также 60 нейронов, функция активации - Relu:

max(0, x); выходной слой - 1 нейрон, функция активации - Sigmoid: $\frac{1}{1+e^{-x}}$.

Установка начальных случайных весов в слоях обеспечивается параметром kernel_initializer='normal'.

```
model.add(Dense(60, input_dim=60, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
```

6. Были определены параметры обучения сети: в качестве функции потерь используется "binary_crossentropy" - функция, которая в основном используется при бинарной классификации, метрика - точность, оптимизатор "adam".

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

7. Было запущено обучение сети с помощью метода fit (адаптирует модель под обучающие данные).

```
model.fit(X, encoder_Y, epochs=100, batch_size=10, validation_split=0.1)
```

8. Для проверки работоспособности программы она была запущена. В процессе обучения нейронной сети отображаются четыре величины: потери сети на обучающих и тестовых данных соответственно - loss и val_loss, точность - accuracy и val_accuracy.

```
Epoch 93/100
19/19 [==============================] - 0s 2ms/step - loss: 0.3075 - accuracy: 0.8983 - val_loss: 0.4717 - val_accuracy: 0.7619
Epoch 94/100
19/19 [==============================] - 0s 2ms/step - loss: 0.3074 - accuracy: 0.8602 - val_loss: 0.3538 - val_accuracy: 0.8095
Epoch 95/100
19/19 [==============================] - 0s 2ms/step - loss: 0.2740 - accuracy: 0.9081 - val_loss: 0.5134 - val_accuracy: 0.6667
Epoch 96/100
19/19 [==============================] - 0s 2ms/step - loss: 0.3117 - accuracy: 0.8983 - val_loss: 0.3672 - val_accuracy: 0.8095
Epoch 97/100
19/19 [==============================] - 0s 1ms/step - loss: 0.2939 - accuracy: 0.8893 - val_loss: 0.3783 - val_accuracy: 0.8095
Epoch 98/100
19/19 [==============================] - 0s 2ms/step - loss: 0.3136 - accuracy: 0.8911 - val_loss: 0.4170 - val_accuracy: 0.8095
Epoch 99/100
19/19 [==============================] - 0s 2ms/step - loss: 0.2829 - accuracy: 0.9027 - val_loss: 0.3724 - val_accuracy: 0.8095
Epoch 100/100
19/19 [==============================] - 0s 2ms/step - loss: 0.2575 - accuracy: 0.9043 - val_loss: 0.4064 - val_accuracy: 0.8095
```

9. Для построения графиков ошибок и точности в ходе обучения сети был использован следующий код:

```python
loss = hist.history['loss']
acc = hist.history['accuracy']
val_loss = hist.history['val_loss']
val_acc = hist.history['val_accuracy']
epochs = range(1, len(loss) + 1)

#Построение графика ошибки
plt.plot(epochs, loss, label='Training loss', linestyle='--', linewidth=2, color="darkmagenta")
plt.plot(epochs, val_loss, 'b', label='Validation loss', color="lawngreen")
plt.title('Training and validation loss')                    #оглавление на рисунке
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#Построение графика точности
plt.clf()
plt.plot(epochs, acc, label='Training acc', linestyle='--', linewidth=2, color="darkmagenta")
plt.plot(epochs, val_acc, 'b', label='Validation acc', color="lawngreen")
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Изучение различных архитектур ИНС.

Модель 1 (исходная)

```python
model = Sequential()
model.add(Dense(60, input_dim=60, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist = model.fit(X, encoder_Y, epochs=100, batch_size=10, validation_split=0.1)
```

График ошибок:

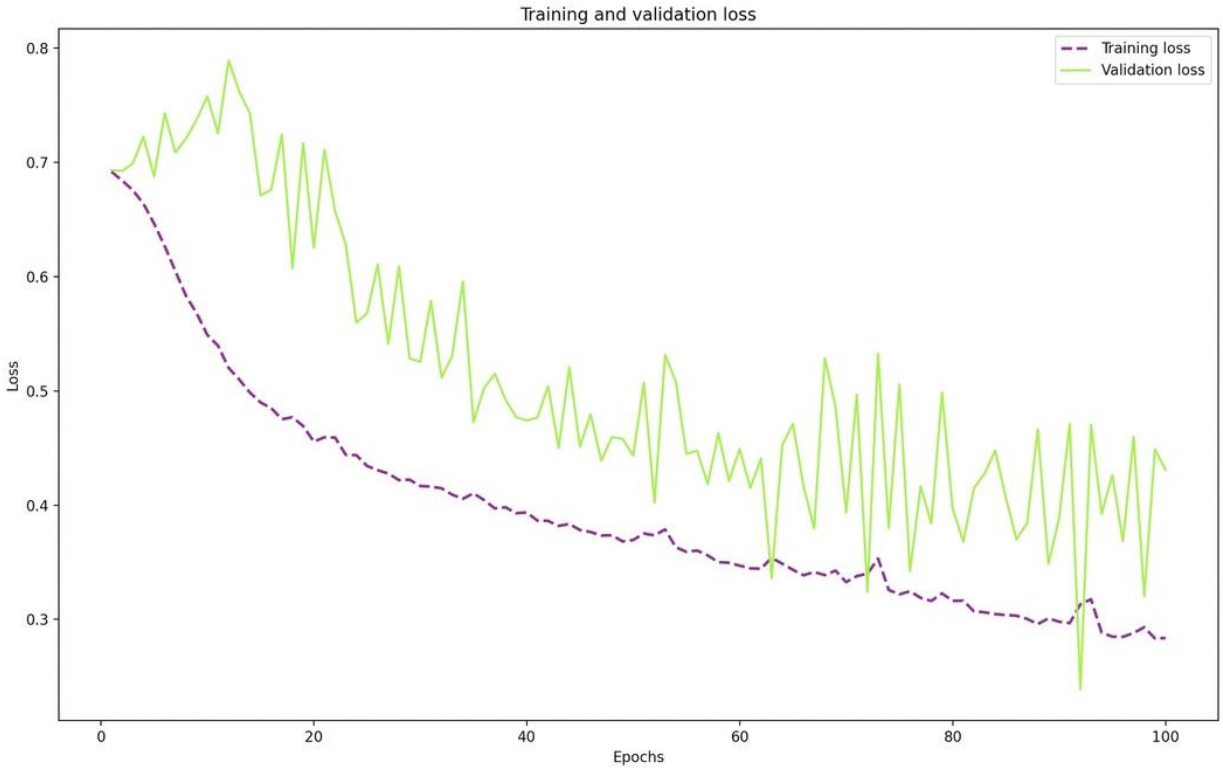
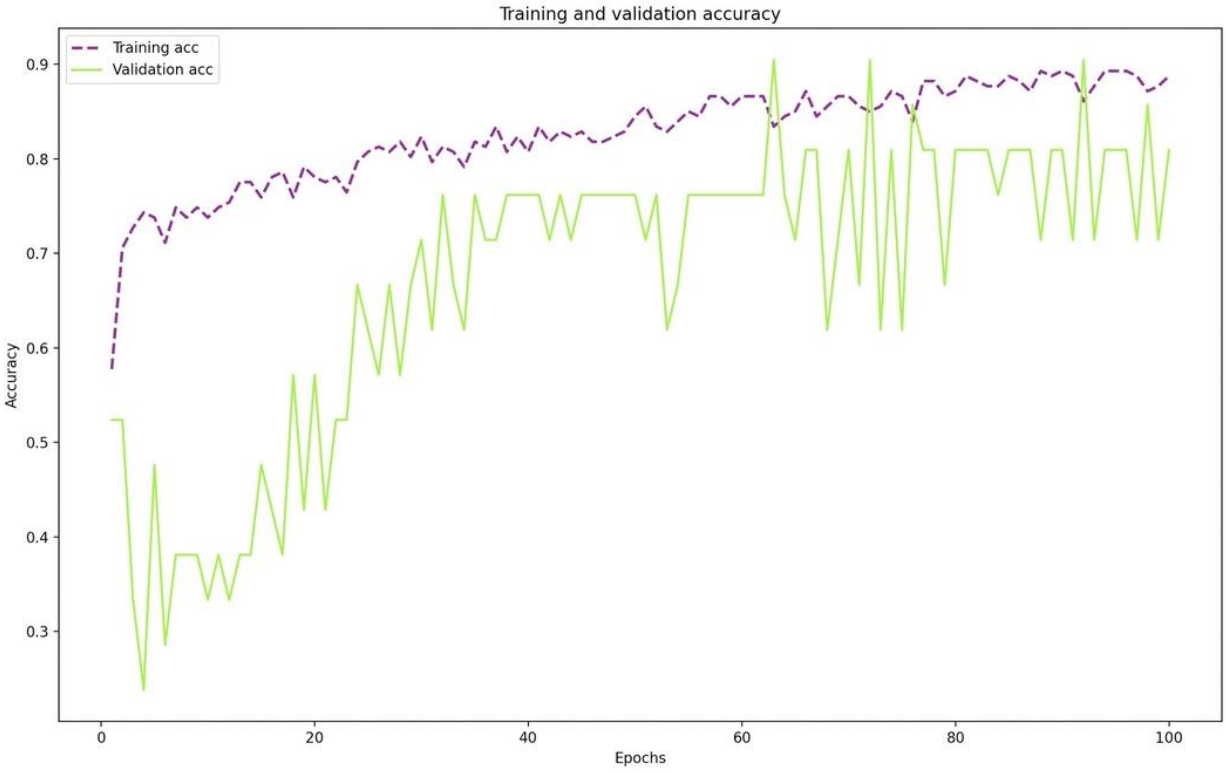


График точности:



Изменения показателей сети:

1 запуск		**2 запуск**		**3 запуск**	
Потери сети					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
0.3337	0.4913	0.2746	0.4308	0.2925	0.3329
Точность (%)					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
87	67	90	81	89	86

Вывод: как видно из графиков и таблицы "Изменения показателей сети", данная модель не достигает относительно высокой точности и низких потерь сети, в особенности на тестовых данных. Таким образом, данная ИНС не является подходящей для решения имеющийся задачи.

В представленном наборе данных присутствует некоторая избыточность, так как с разных углов описывается один и тот же сигнал. Вероятно, что некоторые углы отражения сигнала имеют большую значимость, чем другие, выясним какие (первая половина или вторая).

Модель 2 (уменьшение количества нейронов в входном слое с 60 до 30)

```
model = Sequential()
model.add(Dense(60, input_dim=30, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist = model.fit(X, encoder_Y, epochs=100, batch_size=10, validation_split=0.1)
```

Данные:

```
X = dataset[:, 0:30].astype(float)
Y = dataset[:, 60]
```

График ошибок:

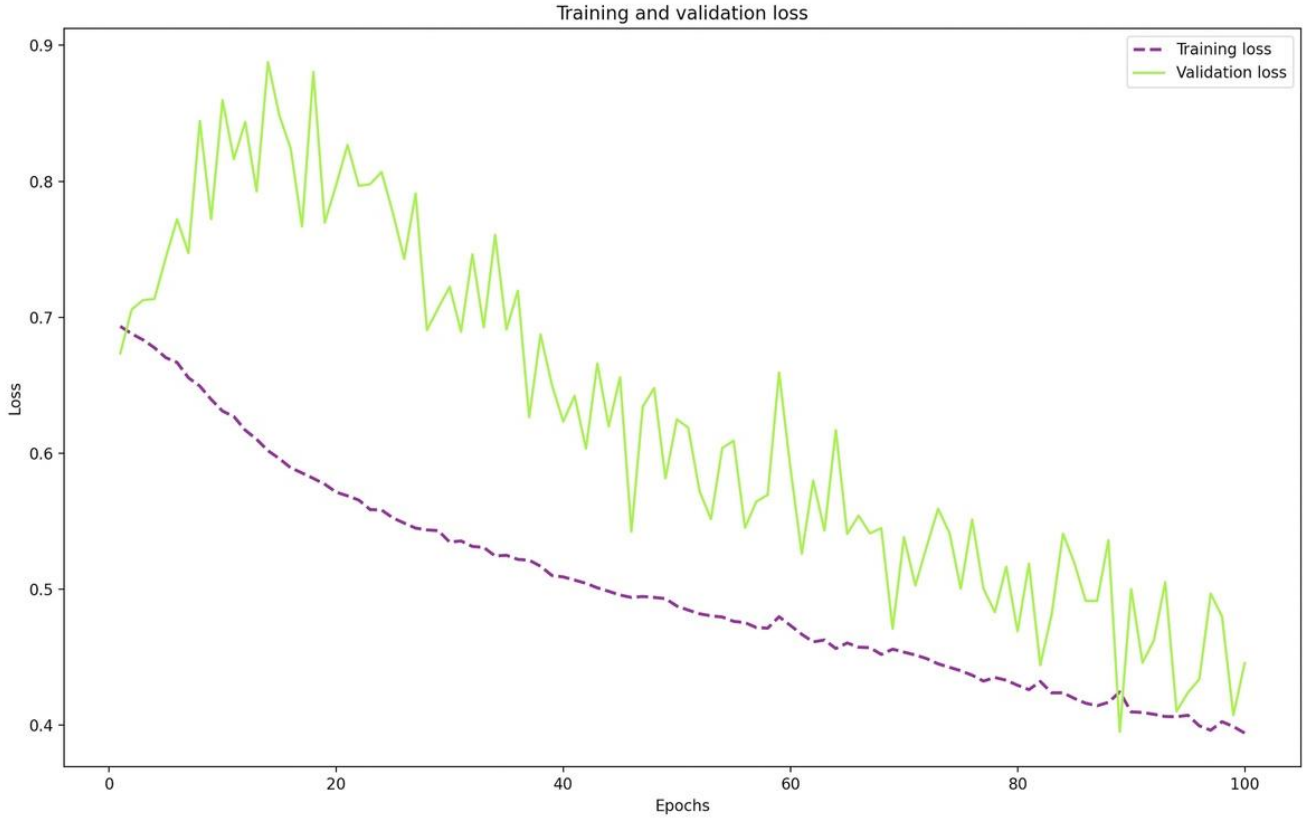
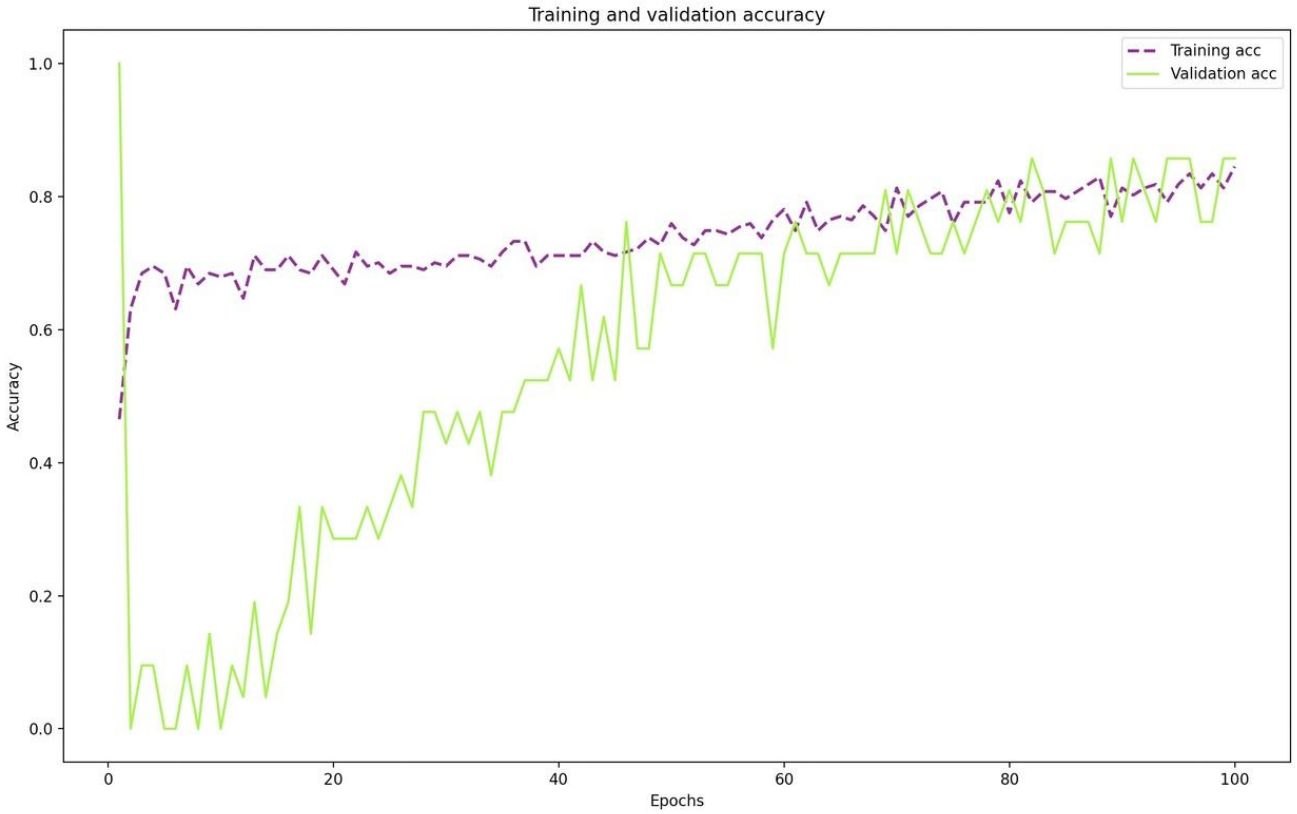


График точности:



Изменения показателей сети:

1 запуск		**2 запуск**		**3 запуск**	
Потери сети					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
0.3826	0.4979	0.4129	0.4457	0.3978	0.4573
Точность (%)					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
82	76	82	86	83	86

Данные:

```
X = dataset[:, 30:60].astype(float)
Y = dataset[:, 60]
```

Изменения показателей сети:

1 запуск		**2 запуск**		**3 запуск**	
Потери сети					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
0.4862	0.6943	0.3735	0.6952	0.4853	0.6632
Точность (%)					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
81	47	88	47	80	48

График ошибок:

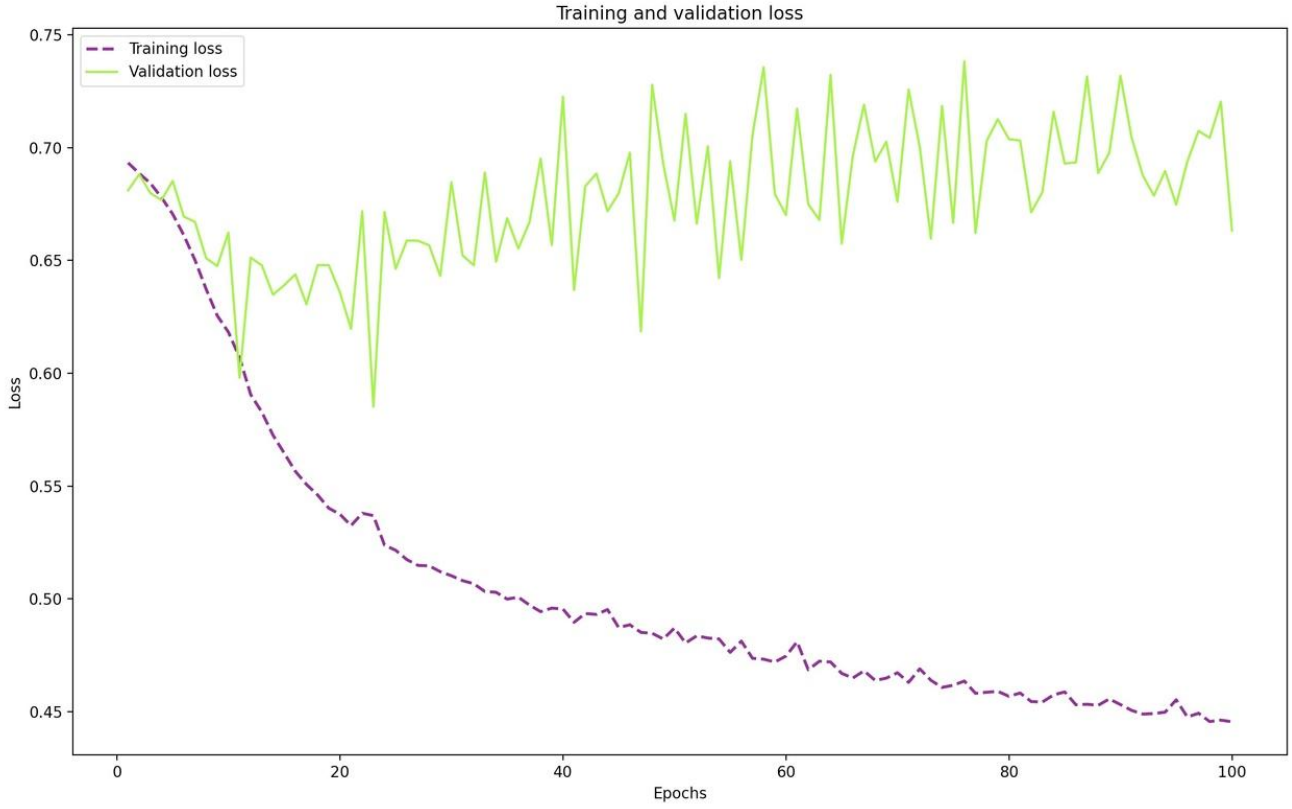
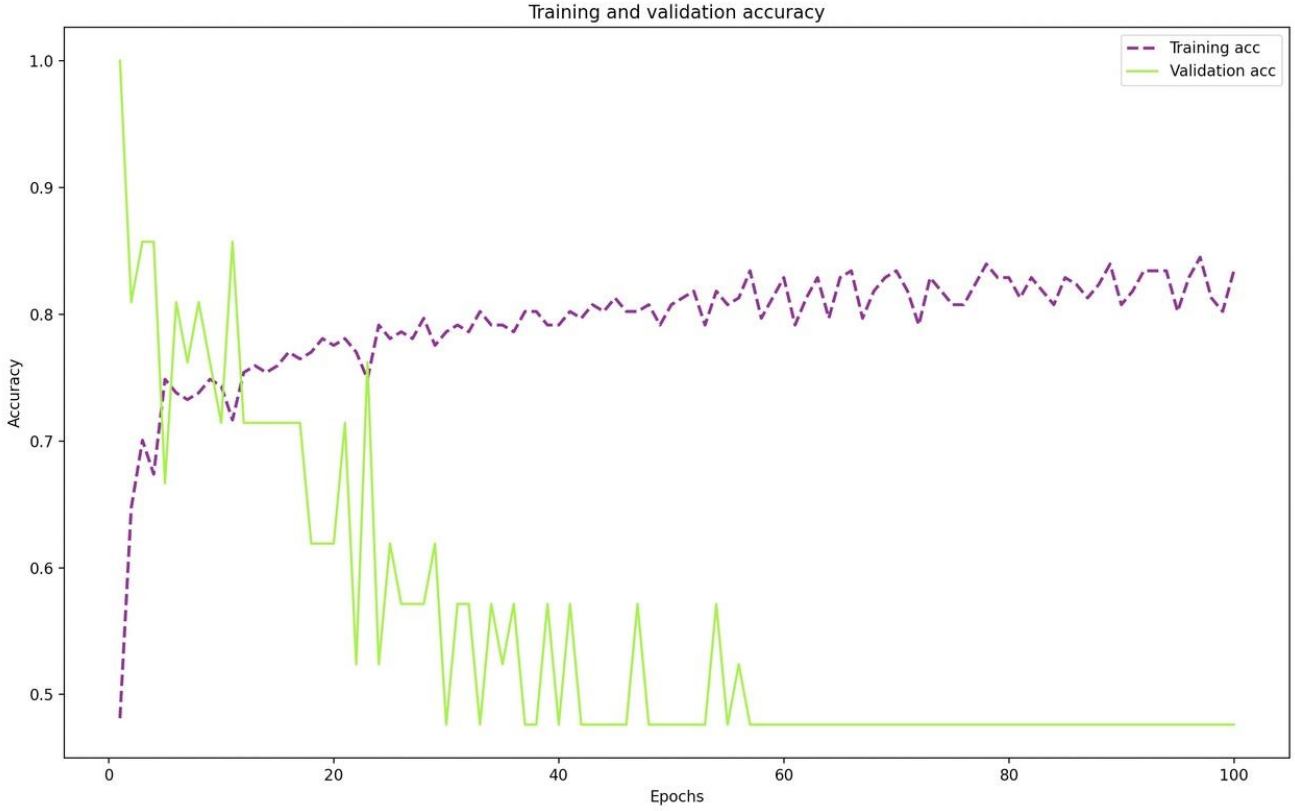


График точности:



Вывод: из таблиц "Изменения показателей сети" можно сделать вывод, что для обучения ИНС лучше использовать первую половину данных, которая, вероятно имеет большую значимость, так как при обучении на второй половине данных вероятность правильного ответа на тестовых данных равна всего лишь 50%.

Сравнивая модели 1 и 2 (модель 2 на первой половине данных) можно сделать вывод, что точность сети и ее потери практически не изменились, но стоит отметить, что значения этих показателей на тестовых и обучающих данных при разных запусках ближе по значениям, чем в модели 1.

Так, можно сделать вывод, что уменьшение нейронов не привело к негативным последствиям в работе ИНС, а, следовательно, нет никакого смысла использовать избыточные входных данные.

Был рассмотрен вариант уменьшения количества нейронов вдвое на скрытом слое, что привело к небольшому увеличению потери сети и уменьшению точности, поэтому эту модель подробно рассматривать не будем.

Модель 3 (увеличим количество нейронов на скрытом слое с 60 до 120)

```
model.add(Dense(120, input_dim=30, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist = model.fit(X, encoder_Y, epochs=100, batch_size=10, validation_split=0.1)
```

Изменения показателей сети:

1 запуск		**2 запуск**		**3 запуск**	
Потери сети					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
0.3285	0.3280	0.3377	0.3590	0.3612	0.3946
Точность (%)					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
89	86	89	91	88	85

График ошибок:

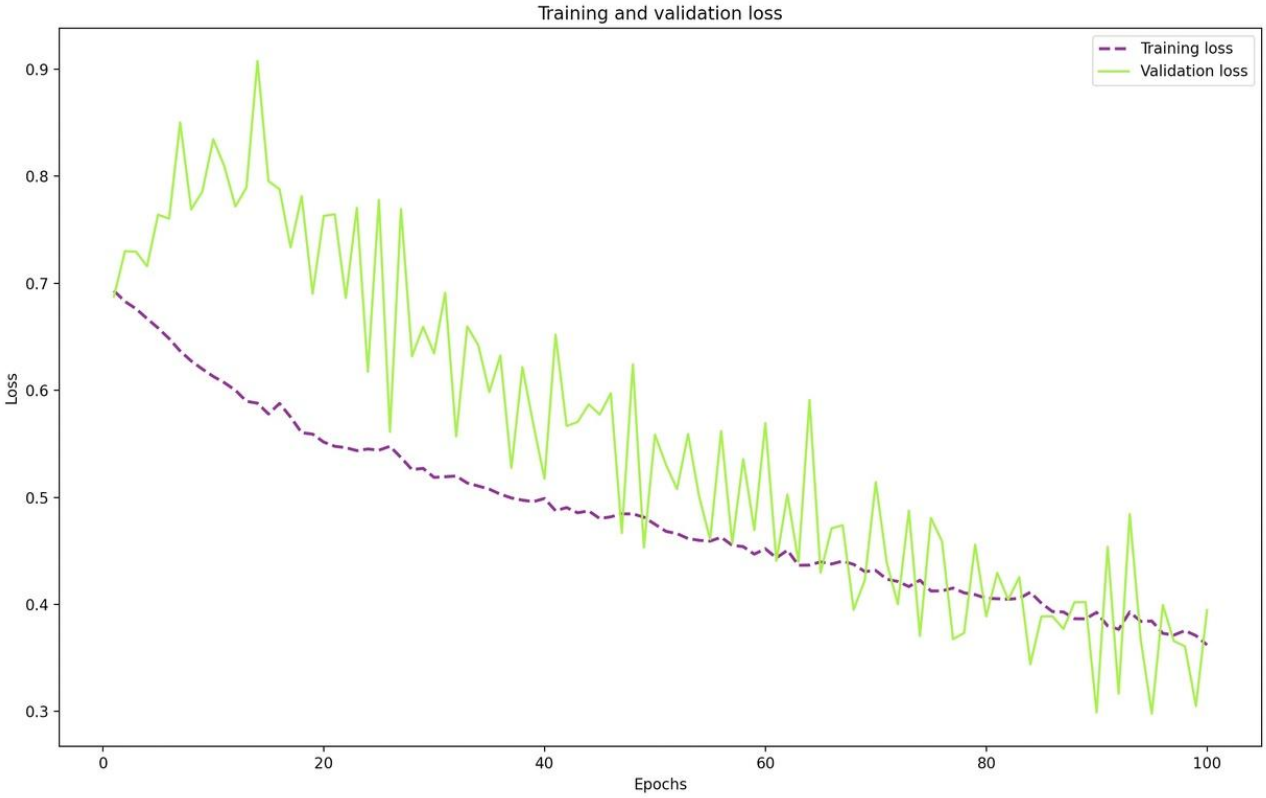
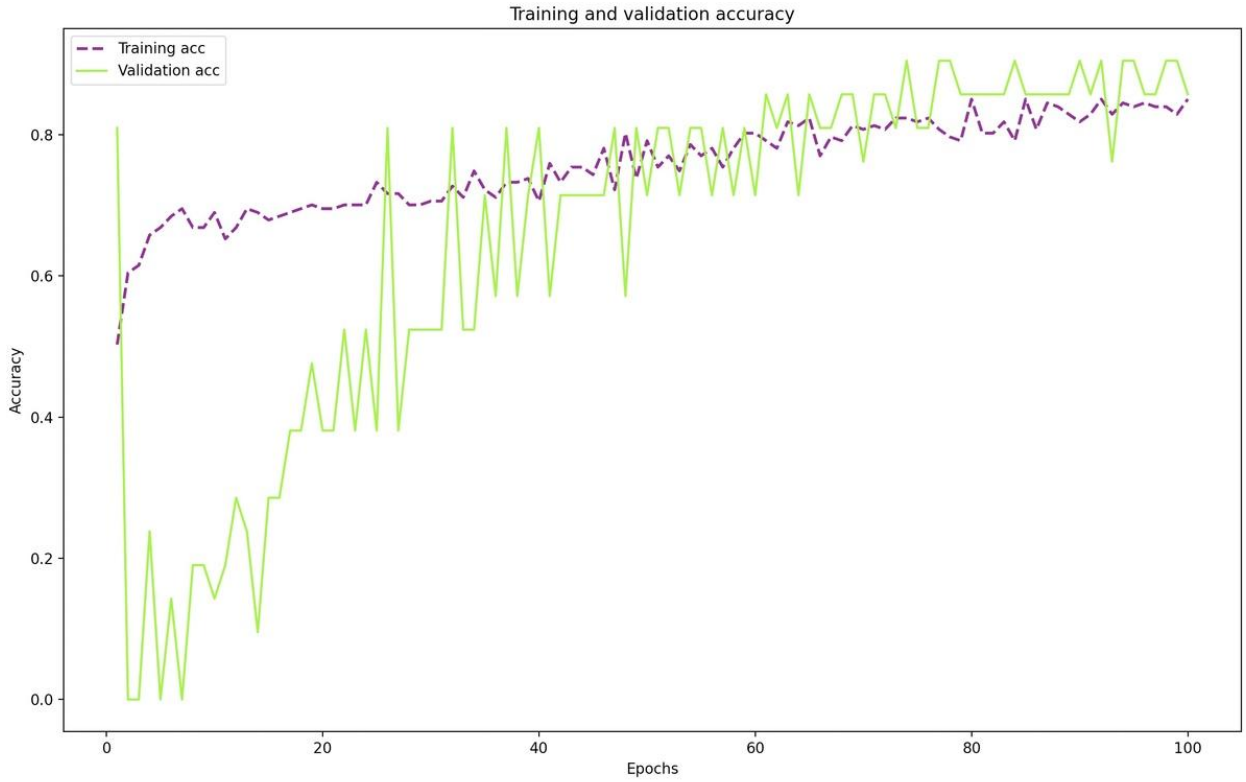


График точности:



Вывод: как видно из графиков и таблицы "Изменения показателей сети" параметр точности сети вырос, а потери - уменьшился, но значения этих изменений практически незначительны. При этом колебания графиков точности и ошибок остались такими же резкими. Оставим прежнее (60) количество нейронов на скрытом слое.

Нейронная сеть с несколькими слоями позволяет находить закономерности не только во входных данных, но и в их комбинациях. Также, дополнительные слои позволяют ввести нелинейность в сеть, что позволяет получать более высокую точность.

Модель 4 (добавление скрытого слоя, содержащего 15 нейронов)

```
model = Sequential()
model.add(Dense(60, input_dim=30, kernel_initializer='normal', activation='relu'))
model.add(Dense(15, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist = model.fit(X, encoder_Y, epochs=100, batch_size=10, validation_split=0.1)
```

Изменения показателей сети:

1 запуск		**2 запуск**		**3 запуск**	
Потери сети					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
0.2491	0.2859	0.2147	0.2038	0.2703	0.1452
Точность (%)					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
91	91	94	95	90	95

График ошибок:

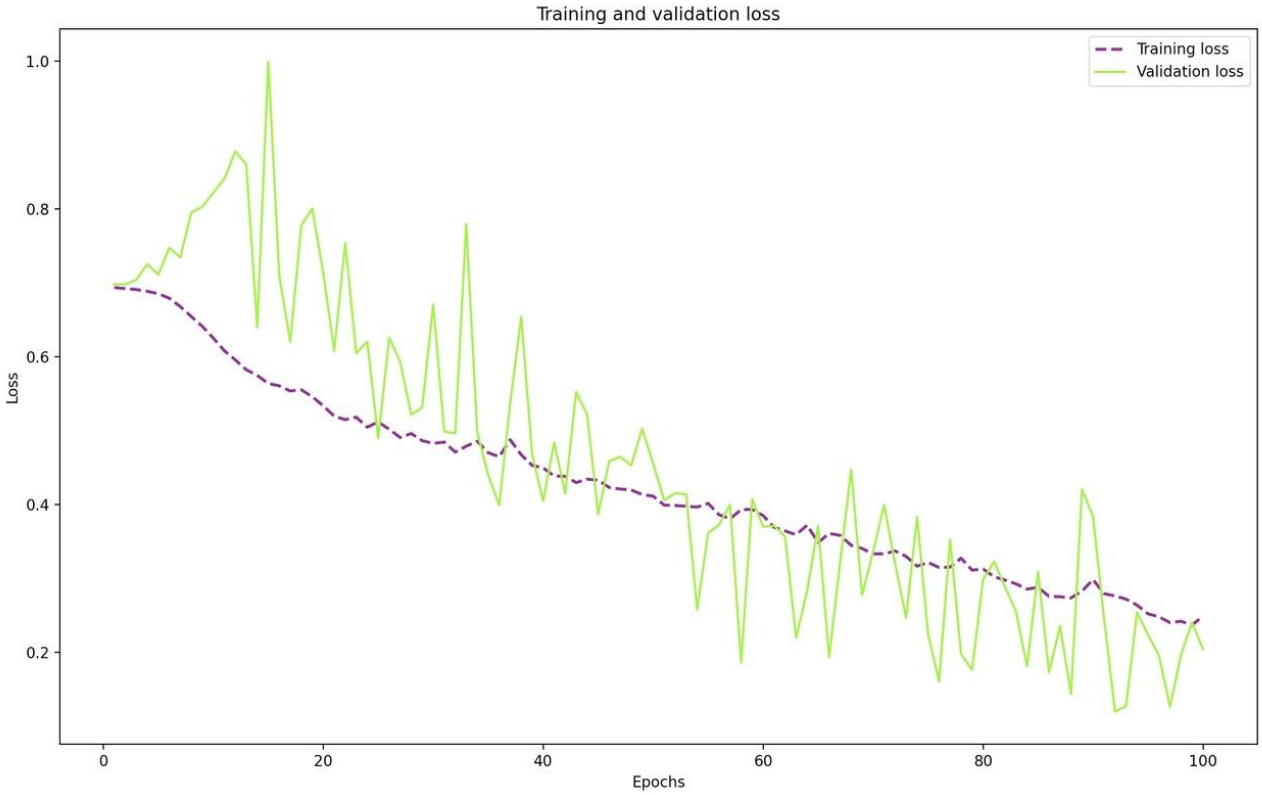
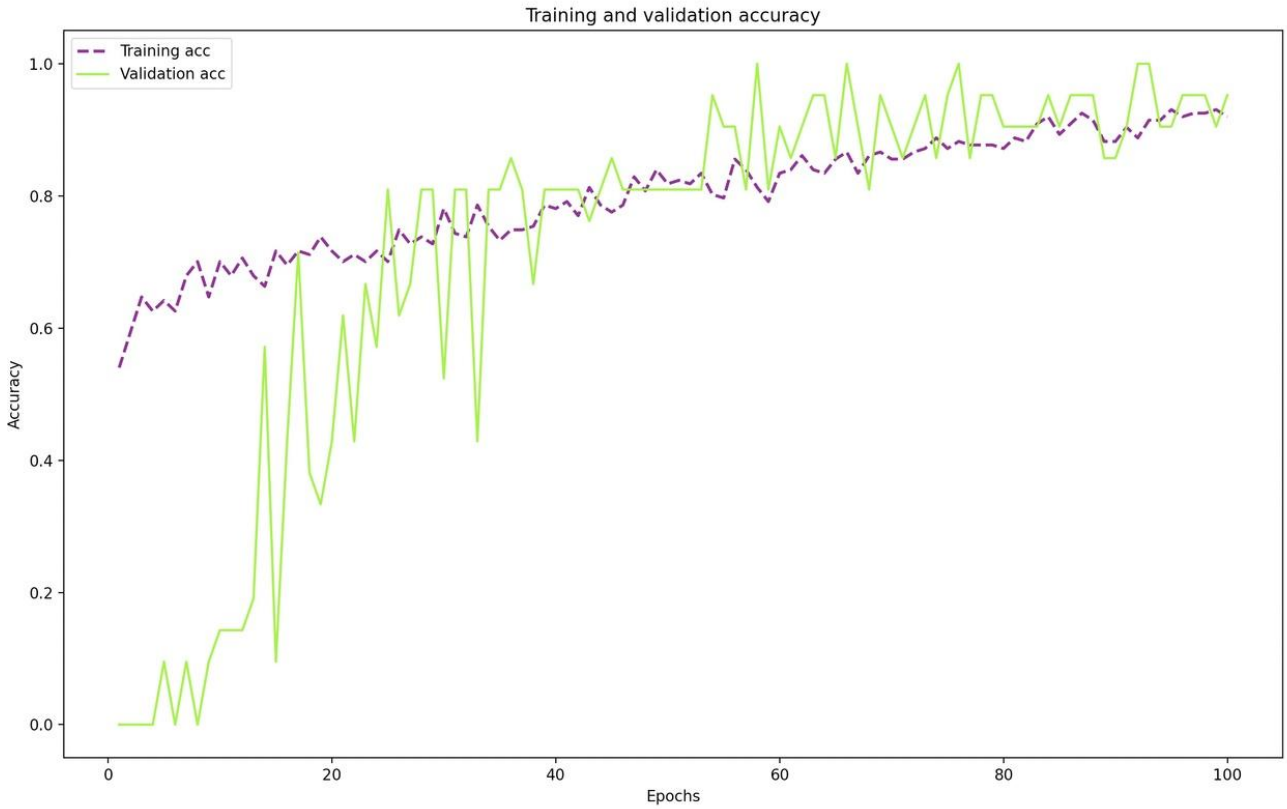


График точности:



Вывод: как видно из таблицы "Изменения показателей сети" точность сети на обучающих и тестовых данных возросла, по сравнению с предыдущими моделями (1, 2 и 3) с 67% до 95%. Ошибки сети, уменьшились вдвоем, по сравнению с моделью 3, значения потерь которой были ниже всех рассмотренных ранее моделей. Кроме того, можно отметить стабильность получаемых данных при разных запусках. Таким образом, можно сделать вывод, что добавление дополнительного скрытого слоя положительно повлияло на работу нейронной сети.

Модель 5 (добавление еще одного скрытого слоя, содержащего 15 нейронов)

```
model = Sequential()
model.add(Dense(60, input_dim=30, kernel_initializer='normal', activation='relu'))
model.add(Dense(15, kernel_initializer='normal', activation='relu'))
model.add(Dense(15, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
hist = model.fit(X, encoder_Y, epochs=100, batch_size=10, validation_split=0.1)
```

Изменения показателей сети:

1 запуск		**2 запуск**		**3 запуск**	
Потери сети					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
0.2619	0.1317	0.1295	0.1620	0.2159	0.0863
Точность (%)					
Обучающие	Тестовые	Обучающие	Тестовые	Обучающие	Тестовые
93	95	98	91	91	100

График ошибок:

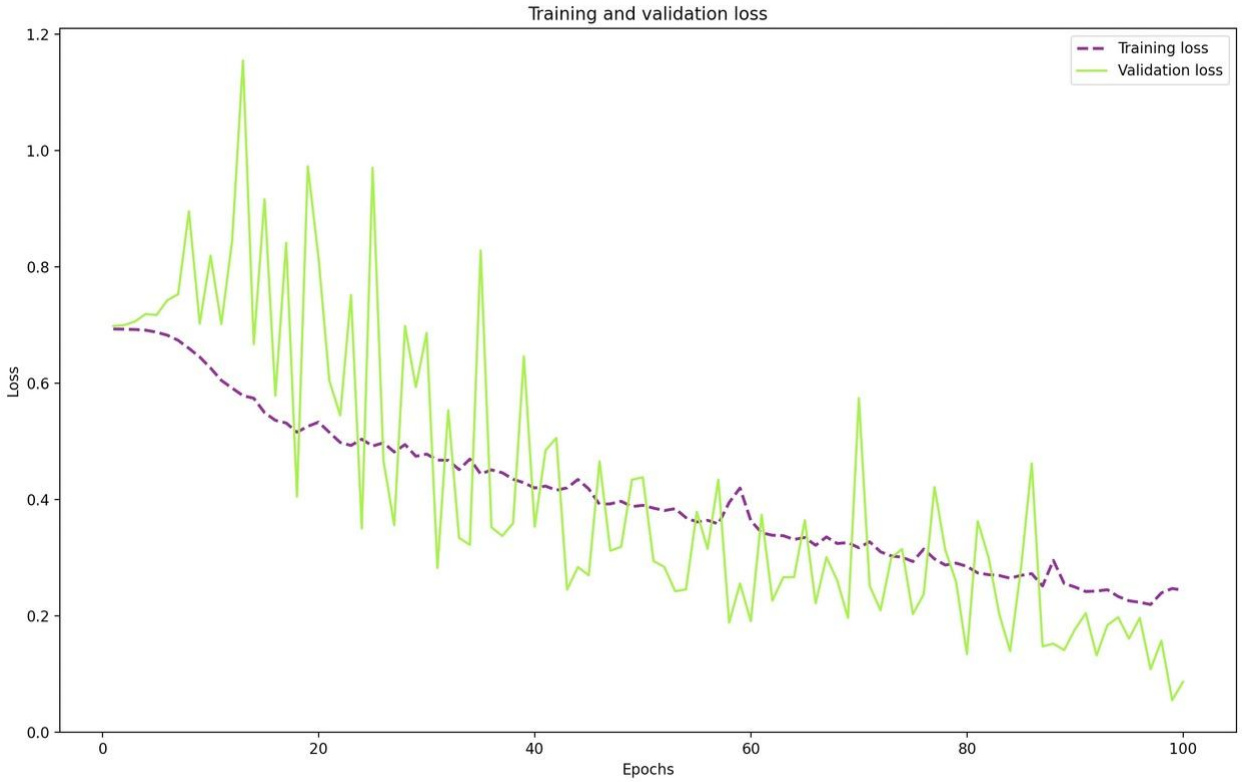
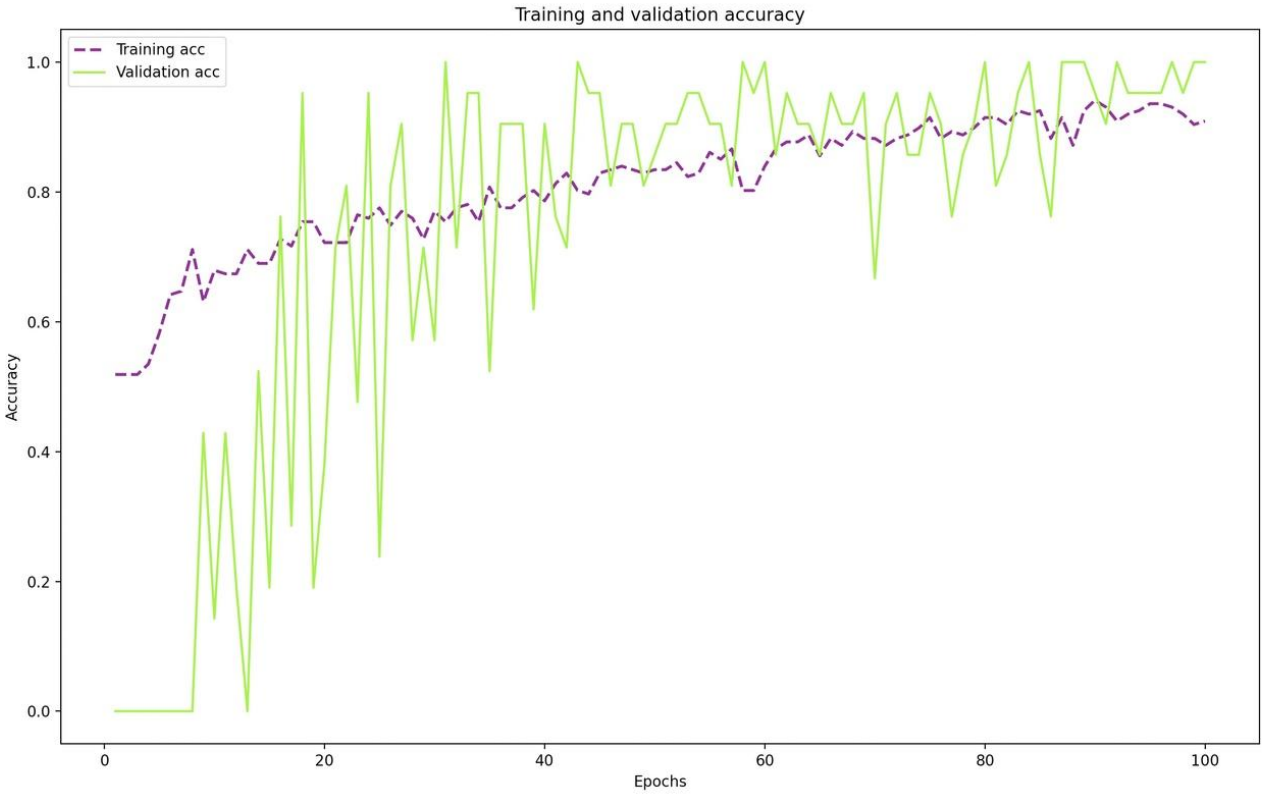


График точности:



Вывод: сравнивая модели 5 и 4, можно заметить, что при данной архитектуре получилось добиться 100% (впервые) точности на тестовых данных. Кроме того, было получено наименьшее из имеющихся раньше, значение потери сети равное 0.0863.

Дальнейшее увеличение слоев не приводит к более стабильным и высоким по точности результатам, а даже напротив приводит к ухудшению показателей (на тестовых данных), что, вероятно, связано с переобучением нейронной сети. По этой причине на данном шаге остановимся и примем в качестве окончательной и дающей лучшие результаты модель под номером 5.

Выводы.

В результате выполнения лабораторной работы были построены различные по архитектуре модели ИНС, решающие задачу бинарной классификации отраженных сигналов радара. Было изучено влияние увеличения/уменьшения количества нейронов в слое, а также влияние добавления дополнительных скрытых слоев. Для всех рассматриваемых моделей были построены соответствующие им графики ошибок и точности. Была выбрана модель, дающая высокую точность и минимальные из получаемых ранее потери сети.