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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Искусственные нейронные сети»
Тема: Многоклассовая классификация цветов

Студентка гр. 8383 _____ Максимова А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы.

Реализовать классификацию сортов растения ирис (Iris Setosa - 0, Iris Versicolour - 1, Iris Virginica - 2) по четырем признакам: размерам пестиков и тычинок его цветков.

Задачи.

- Ознакомиться с задачей классификации
- Загрузить данные
- Создать модель ИНС в Keras
- Настроить параметры обучения
- Обучить и оценить модель

Требования.

1. Изучить различные архитектуры ИНС (Разное кол-во слоев, разное кол-во нейронов на слоях)
2. Изучить обучение при различных параметрах обучения (параметры ф-ций fit)
3. Построить графики ошибок и точности в ходе обучения
4. Выбрать наилучшую модель

Основные теоретические положения.

Задача классификации: задача, в которой имеется множество объектов, где каждый объект, исходя из его свойств, параметров, можно отнести к конкретному классу. Конечное множество объектов, классы которых известны еще до решения задачи, называется выборкой. Классовая принадлежность оставшихся объектов соответственно неизвестна. Целью задачи является построение алгоритма, способного классифицировать любой объект из исходного множества.

Выполнение работы.

1. Были импортированы все необходимые для работы классы и функции; был скачан файл iris.data, содержащий набор входных данных, и переименован в iris.csv (текстовый формат, предназначенный для представления табличных данных).

```
import pandas #для работы с данными
from tensorflow.keras.layers import Dense #плотно связанный слой НС
from tensorflow.keras.models import Sequential #нейронная сеть прямого распространения
from tensorflow.keras.utils import to_categorical #для преобразования выходных атрибутов из вектора в матрицу
from sklearn.preprocessing import LabelEncoder #для преобразования текстовых данных в числа от 0 до n-1 (n – число классов)
```

2. После данные были загружены с помощью pandas и разделены на входные X (первые 4 столбца - размеры пестиков и тычинок) и выходные Y (последний столбец - сорта ирисов) данные.

```
dataframe = pandas.read_csv("iris.csv", header=None)
```

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
dataset = dataframe.values #numPy представление dataFrame
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
```

3. Так как в данной задаче 3 класса ирисов, выходные данные Y были преобразованы из вектора в матрицу, к виду, представленному ниже:

```
Iris-setosa, Iris-versicolor, Iris-virginica
1,     0,     0
0,     1,     0
0,     0,     1
```

Используемый для этого код, представлен ниже.

```
encoder = LabelEncoder()
```

```
encoded_Y = encoder.fit_transform(Y)#преобразовать в числа и вернуть результат
```

```
[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 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 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
dummy_y = to_categorical(encoded_Y)
```

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

4. После была создана модель, состоящая из двух полносвязных нейронных слоев, в первом - 4 нейрона, во втором - 3. В качестве функции активации для первого слоя выбрана Relu: max(0, x), для второго - Softmax: $\frac{e^{x_i}}{\sum_{i=0}^{k} e^{x_i}}$.

```
model = Sequential()
model.add(Dense(4, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Были определены параметры обучения сети - функция потерь, оптимизатор и метрика.

- функцию потерь, которая определяет, как сеть должна оценивать качество своей работы на обучающих данных и, соответственно, как корректировать ее в правильном направлении;
- оптимизатор — механизм, с помощью которого сеть будет обновлять себя, опираясь на наблюдаемые данные и функцию потерь;
- метрики для мониторинга на этапах обучения и тестирования — здесь нас будет интересовать только точность (доля правильно классифицированных изображений).

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

5. Было запущено обучение сети с помощью метода fit, где `epochs` - количество эпох на обучение модели, `batch_size` - количество объектов, при достижении которого корректируются веса; `validation_split` - число, указывающее какая доля данных будет использована в качестве тестовых данных, не используемых при обучении.

```
model.fit(X, dummy_y, epochs=75, batch_size=10,
validation_split=0.1)
```

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

```
Epoch 45/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9844 - accuracy: 0.4195 - val_loss: 0.9545 - val_accuracy: 0.8000
Epoch 46/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9779 - accuracy: 0.5050 - val_loss: 0.9391 - val_accuracy: 0.8667
Epoch 47/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9547 - accuracy: 0.5390 - val_loss: 0.9293 - val_accuracy: 0.8667
Epoch 48/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9528 - accuracy: 0.6173 - val_loss: 0.9271 - val_accuracy: 0.8667
Epoch 49/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9614 - accuracy: 0.5479 - val_loss: 0.9223 - val_accuracy: 0.8667
Epoch 50/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9473 - accuracy: 0.5756 - val_loss: 0.9037 - val_accuracy: 0.8667
Epoch 51/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9692 - accuracy: 0.4863 - val_loss: 0.9028 - val_accuracy: 0.8667
Epoch 52/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9246 - accuracy: 0.6162 - val_loss: 0.8893 - val_accuracy: 0.8667
Epoch 53/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9408 - accuracy: 0.5608 - val_loss: 0.8820 - val_accuracy: 0.8667
Epoch 54/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9079 - accuracy: 0.6463 - val_loss: 0.8701 - val_accuracy: 0.8667
Epoch 55/75
14/14 [==============================] - 0s 2ms/step - loss: 0.9217 - accuracy: 0.6047 - val_loss: 0.8761 - val_accuracy: 0.8667
```

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

```
#история обратного вызова - получение ошибки и точности в процессе обучения
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')
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()
```

8. После были построены различные модели, отличающиеся архитектурой и/или параметрами обучения, были построены соответствующие им графики ошибок и точности, проанализированы результаты.

Результаты запуска моделей и их оценка.

- Исходная модель

```
model = Sequential()
model.add(Dense(4, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X, dummy_y, epochs=75, batch_size=10, validation_split=0.1)
```

<u>График ошибок:</u>

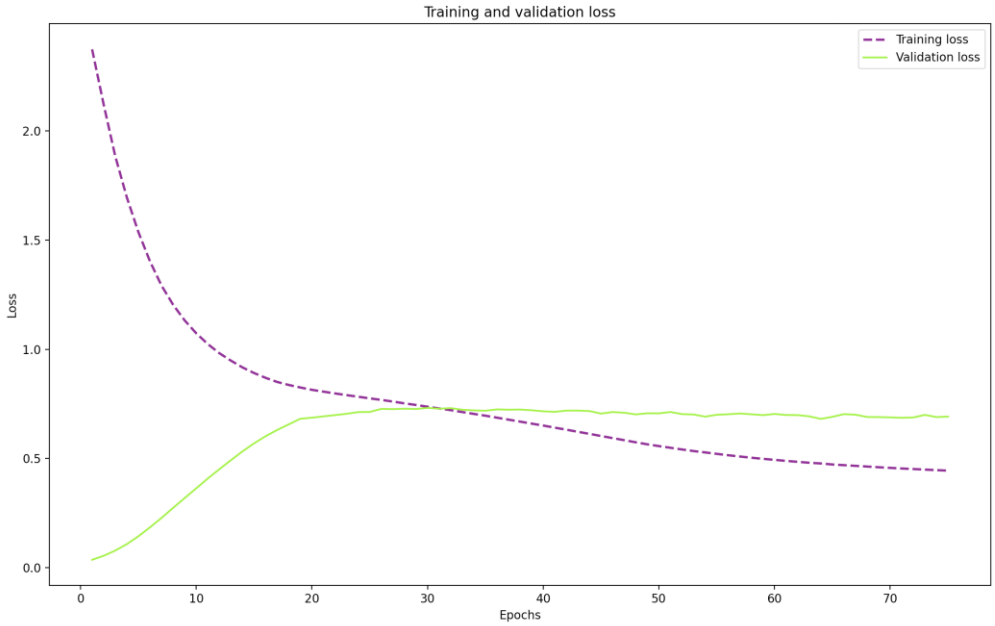
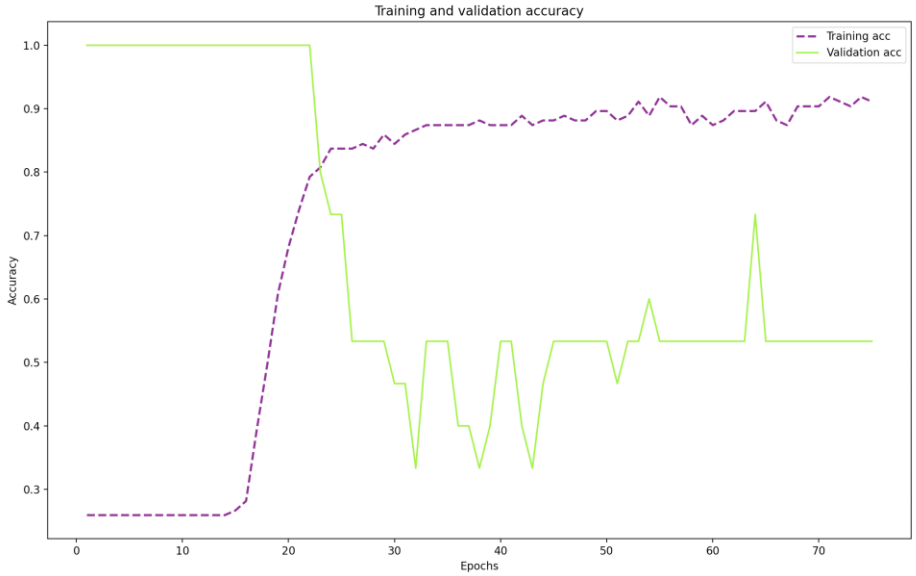


График точности:



Изменения показателей сети (10 запусков обучения):

Потери сети	
Обучающие данные	**Тестовые данные**
0.30 - 1.09	0.49 - 1.32
Точность (%)	
Обучающие данные	**Тестовые данные**
44 - 97	0 - 100

Вывод:

Из графика ошибок можно сделать вывод, что сеть практически вдвое уменьшила показатели ошибок всего лишь за 10 эпох, но дальше этот процесс затухает и полученное в итоге значение потери сети все равно остается высоким.

Из графика точности, видно, что высокие значения точности на обучающих данных достигаются относительно быстро (15 эпох), но при этом на тестовых данных стабильные результаты не достигаются.

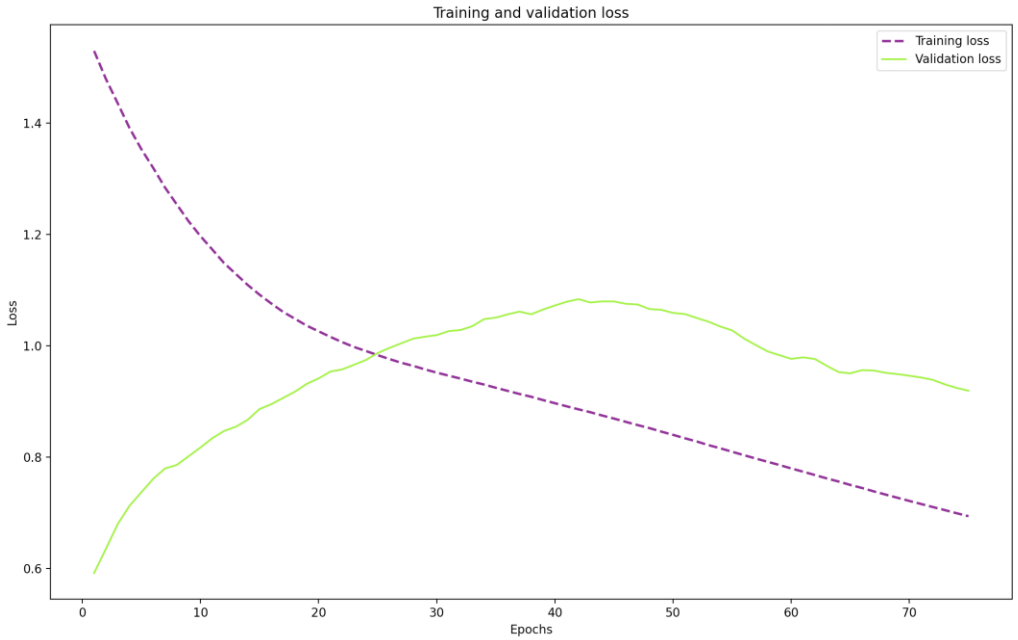
При каждом новом запуске значения потерь сети и точности сильно колеблются, что затрудняет анализ работы НС.

- Исходная модель при изменении параметра **batch_size**

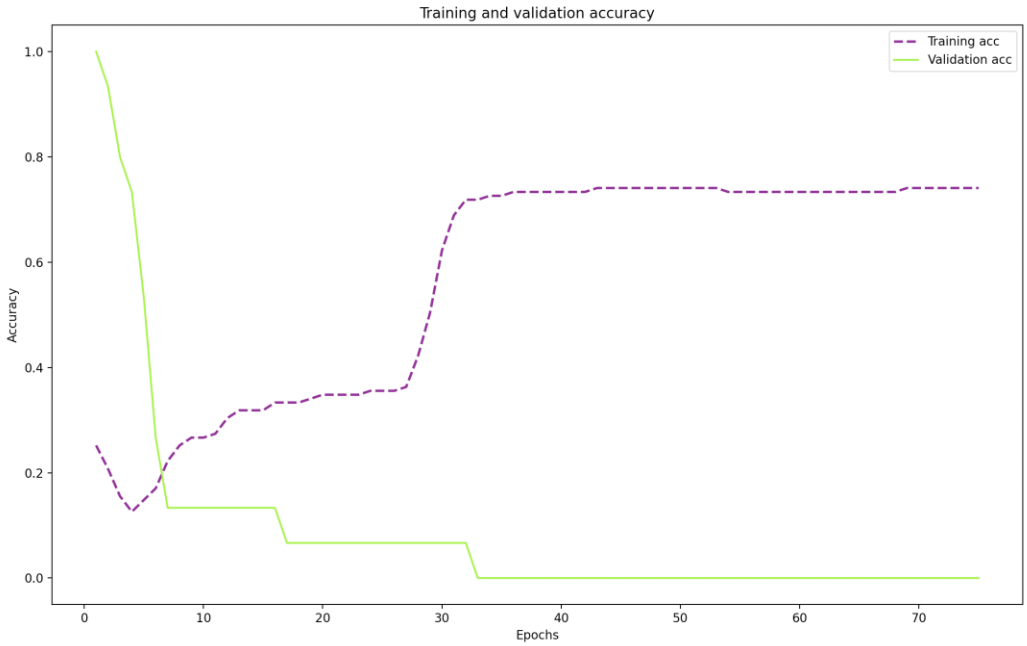
  Увеличим значение параметра batch_size с 10 до 30:

```
model = Sequential()
model.add(Dense(4, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X, dummy_y, epochs=75, batch_size=30, validation_split=0.1)
```

<u>График ошибок:</u>



<u>График точности:</u>



Изменения показателей сети (5 запусков обучения):

Потери сети	
Обучающие данные	**Тестовые данные**
0.52 - 0.91	0.63 - 1.12
Точность (%)	
Обучающие данные	**Тестовые данные**
72 - 85	0 - 86

Уменьшение параметра batch_size с 10 до 5:

```
model = Sequential()
model.add(Dense(4, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X, dummy_y, epochs=75, batch_size=5, validation_split=0.1)
```

График ошибок:

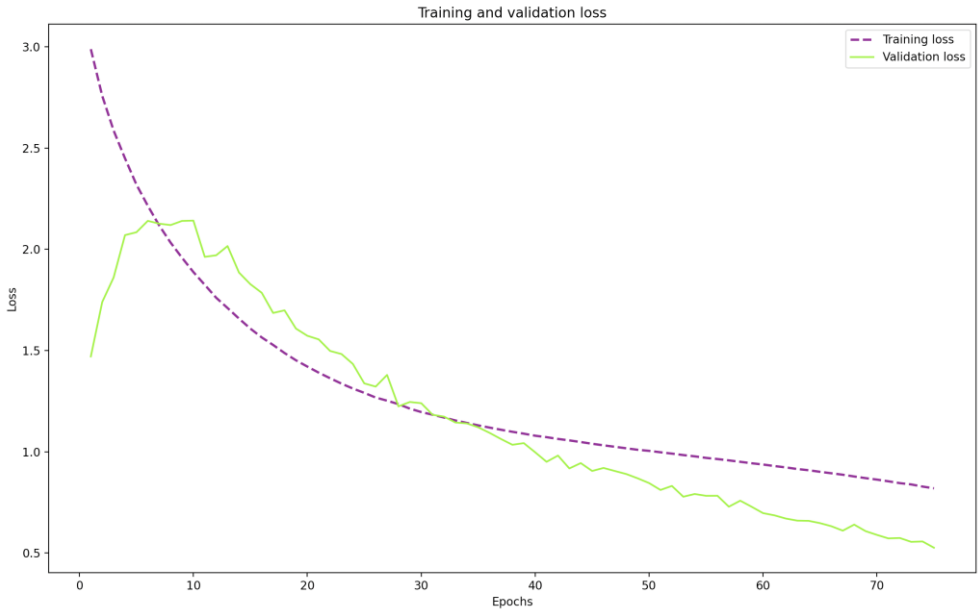
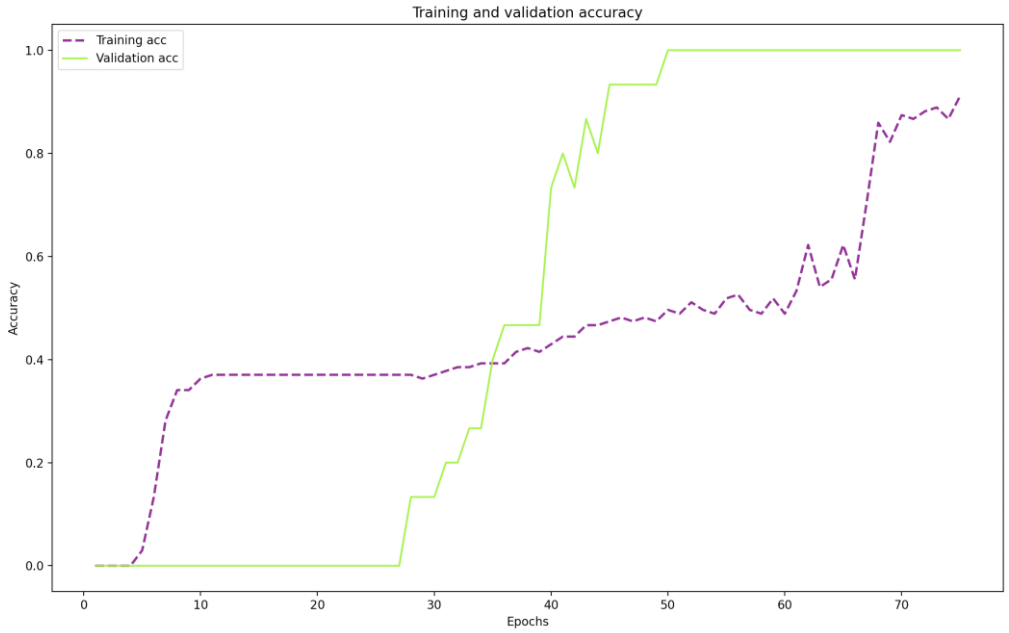


График точности:



Изменения показателей сети (5 запусков обучения):

Потери сети	
Обучающие данные	**Тестовые данные**
0.16 - 0.8	0.25 - 0.91
Точность (%)	
Обучающие данные	**Тестовые данные**
73- 98	0 - 100

Вывод:

Как видно из графиков, уменьшение значения параметра batch_size приводит к уменьшению потерь и увеличению точности, что связано с более частой корректировкой весов. Увеличение же этого параметра не улучшило работу нейронной сети по сравнению с исходной моделью.

Так как уменьшение batch_size привело к улучшению работы нейронной сети, оставим его равным 5.

- Исходная модель при изменении параметра **epochs**, который отвечает за количество повторов обучения сети на обучающих данных.

Уменьшим количество повторов обучения в 2 раза.

```
model = Sequential()
model.add(Dense(4, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X, dummy_y, epochs=35, batch_size=5, validation_split=0.1)
```

График ошибок:

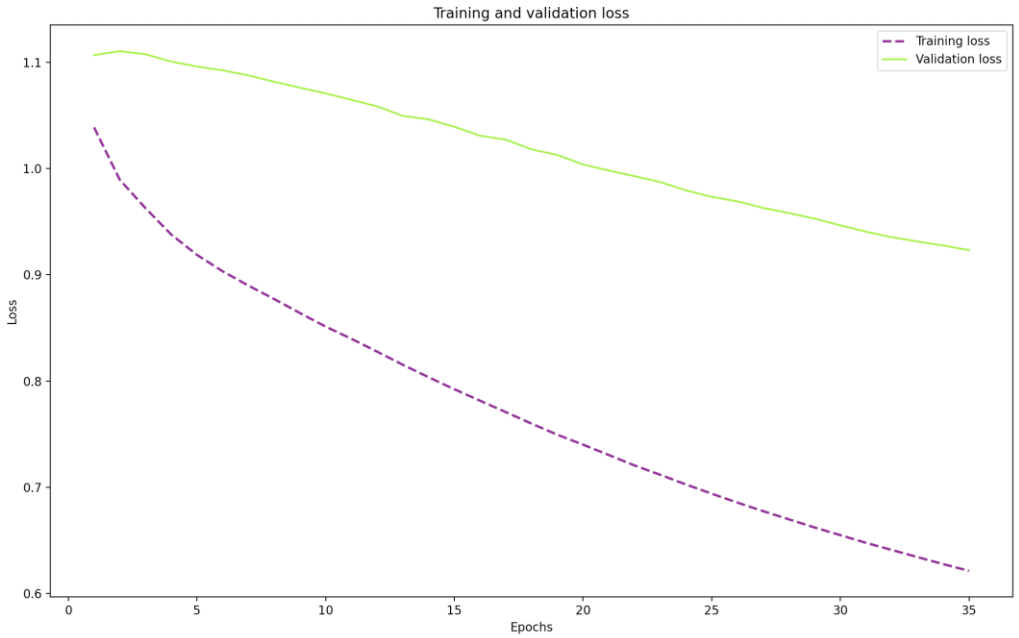
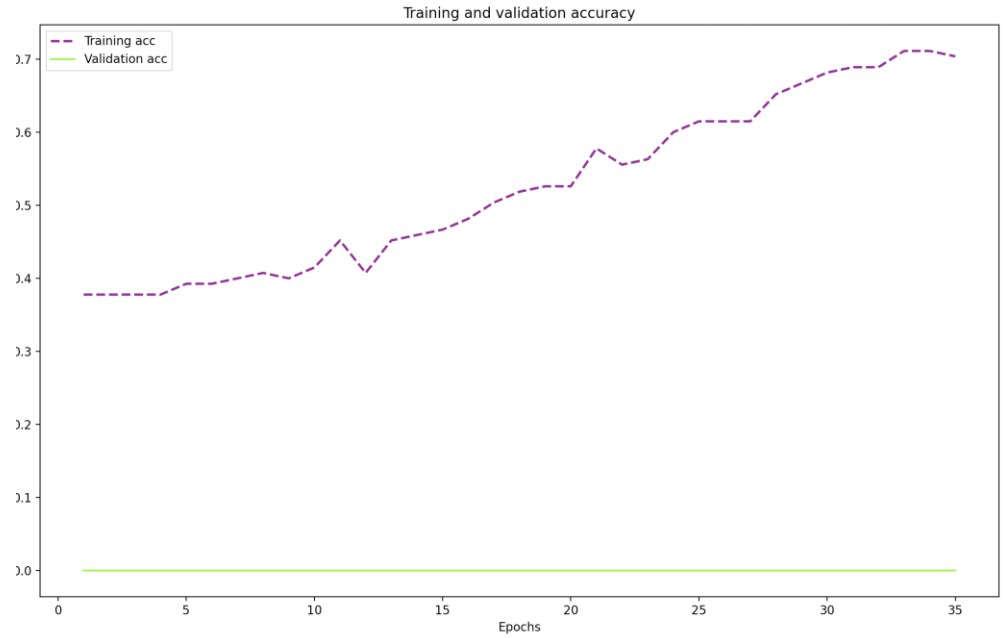


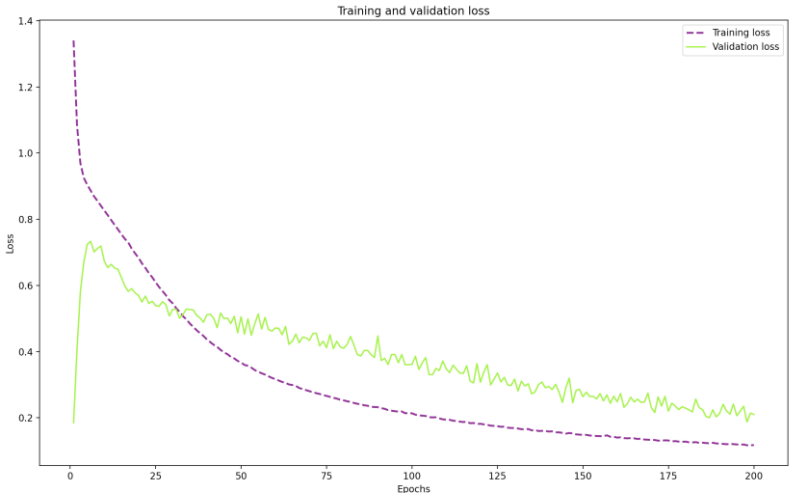
График точности:



Уменьшение числа повторов обучения, как видно из графиков, сильно отразилось на тестовых данных: потеря сети стремится к единице, а ее точность равна 0. Таким образом, уменьшение количества повторов обучения привело к ухудшению работы сети.

Увеличение количества повторов должно повысить точность сети и уменьшить ее потери. При достижении некоторого большого значения сеть может переобучиться, то есть запомнить все обучающие данные, поэтому необходимо подобрать такое значение, при которым будут наблюдаться заметные улучшения работы нейронной сети, но графики, тестовых и обучающих данных, не будут расходиться.

<u>График ошибок (epochs = 200):</u>



<u>График точности (epochs = 200):</u>

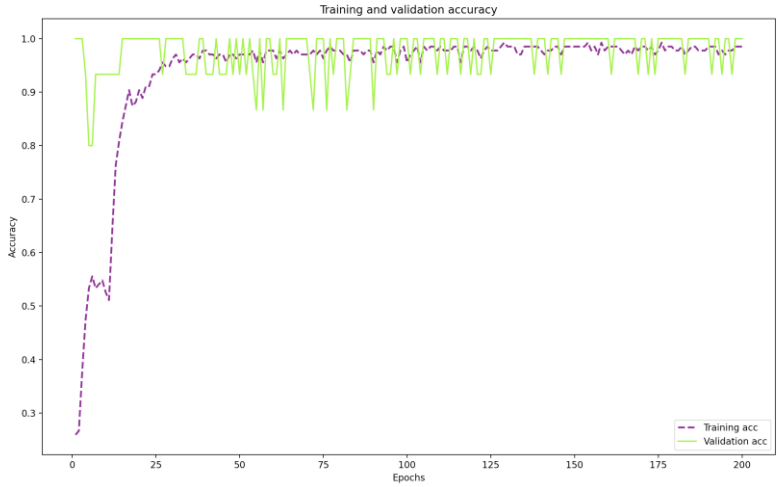


График ошибок (epochs = 600):

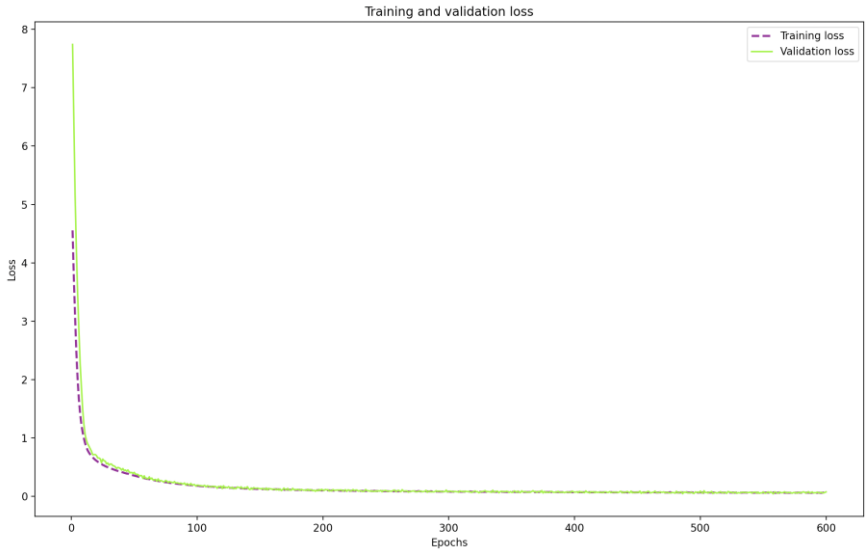
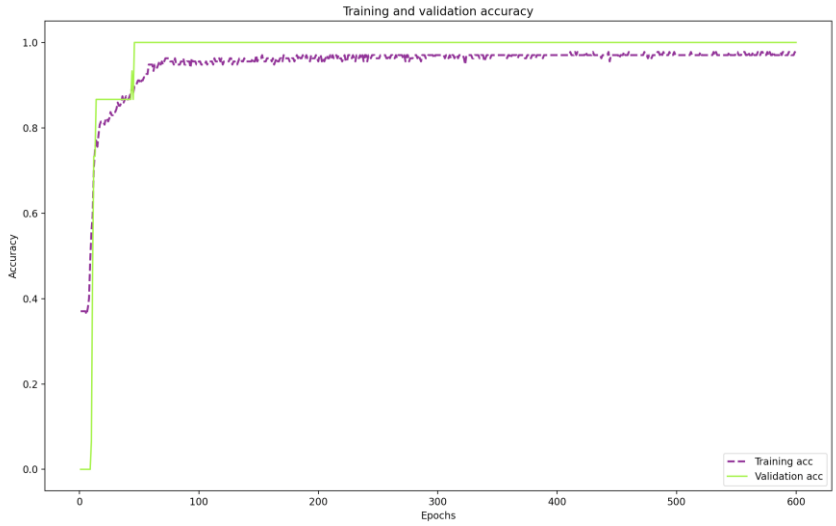


График точности (epochs = 600):

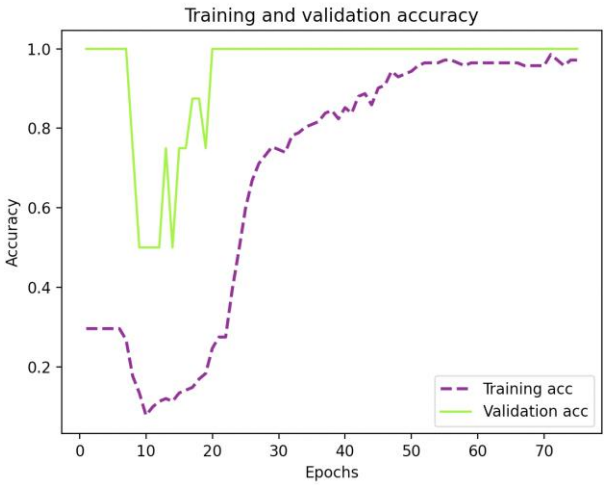


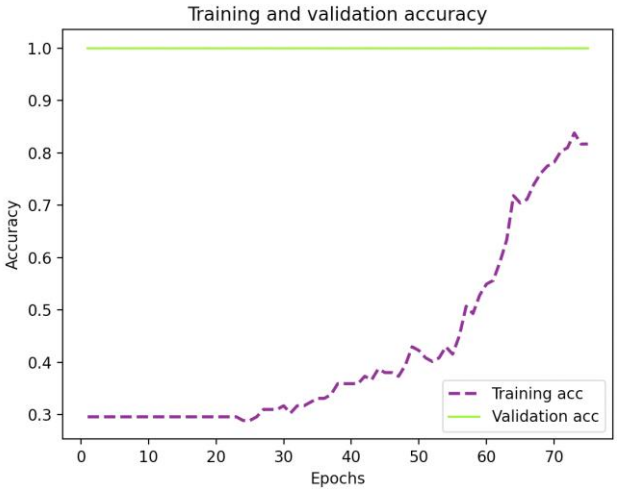
Как видно из графиков, значение epochs = 600 является более оптимальным среди перечисленных: величина потери сети уменьшилась, точность повысилась, на графиках нет шума, при этом переобучения сети не произошло.

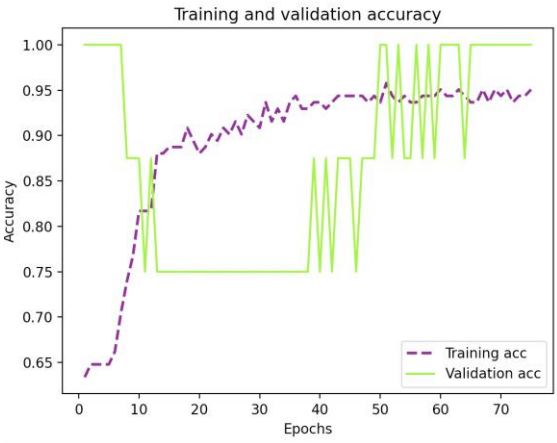
- Исходная модель при изменении параметра **validation_split** - вещественное число, обозначающее какая доля данных будет использована в качестве тестовых данных, не используемых при обучении.

  Пусть validation_split = 0.05. Тогда:

  <u>Графики точности:</u>



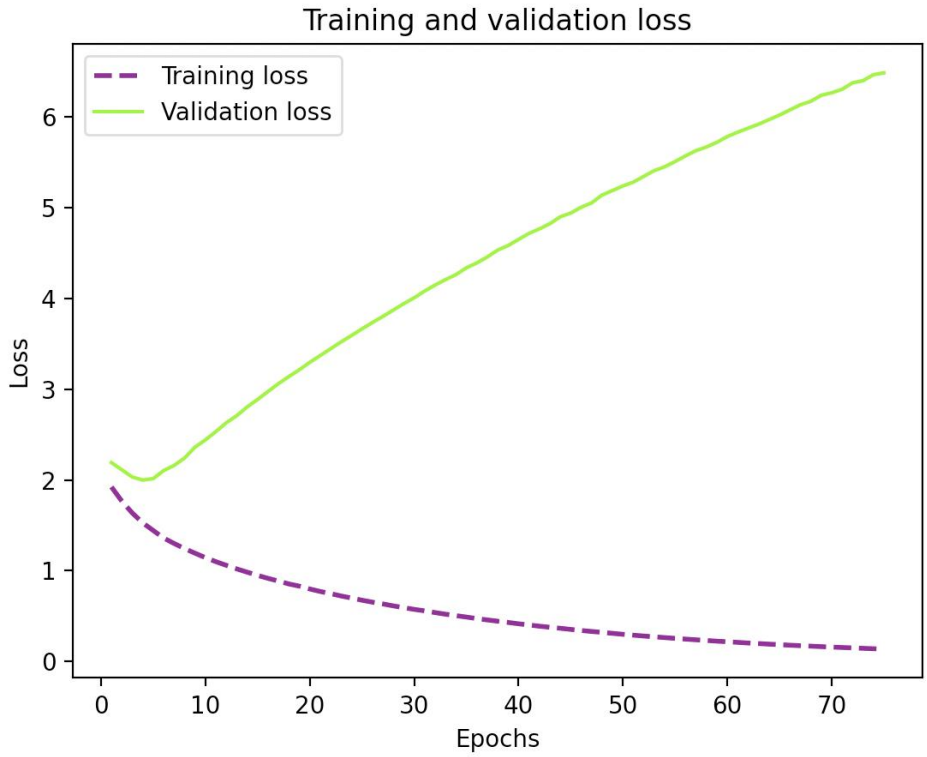




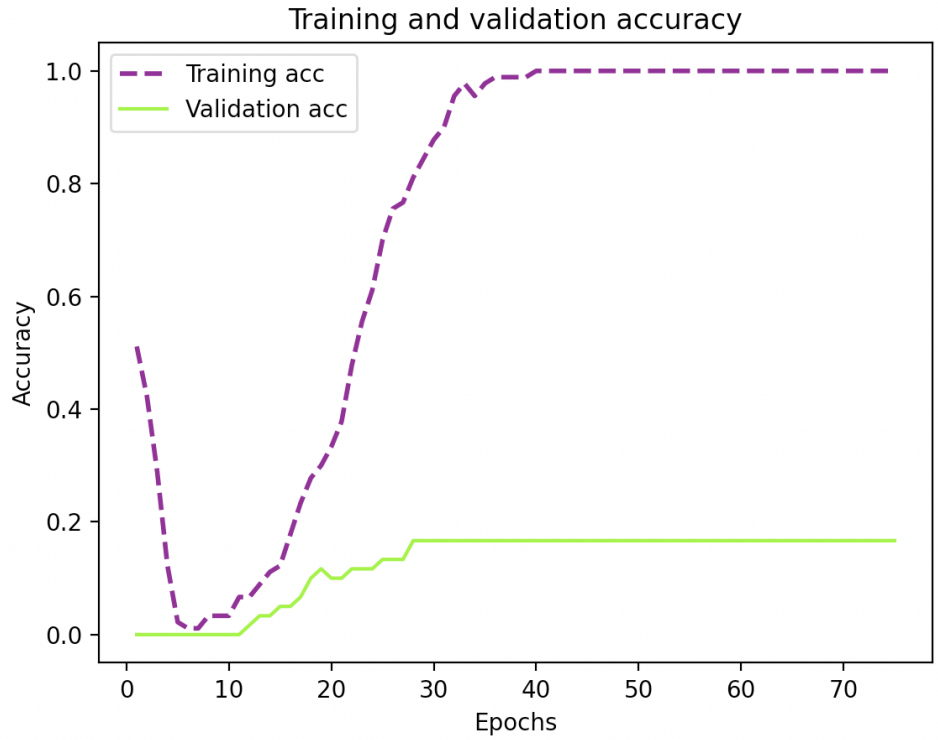
Как видно из графиков, при низкой доле тестовых данных, что усугубляется изначально небольшим объемом входных данных, значения точности достаточно сильно колеблются, поэтому оценить такие результаты сложно.

Пусть validation_split = 0.4. Тогда:

<u>График ошибок:</u>



<u>График точности:</u>



Вывод:

Как видно из графиков, в данном случае, значение потери сети на тестовых данных будет большим, а значение точности на тех же данных - очень маленьким, что связано с уменьшением объема обучающих и увеличением объема тестовых данных.

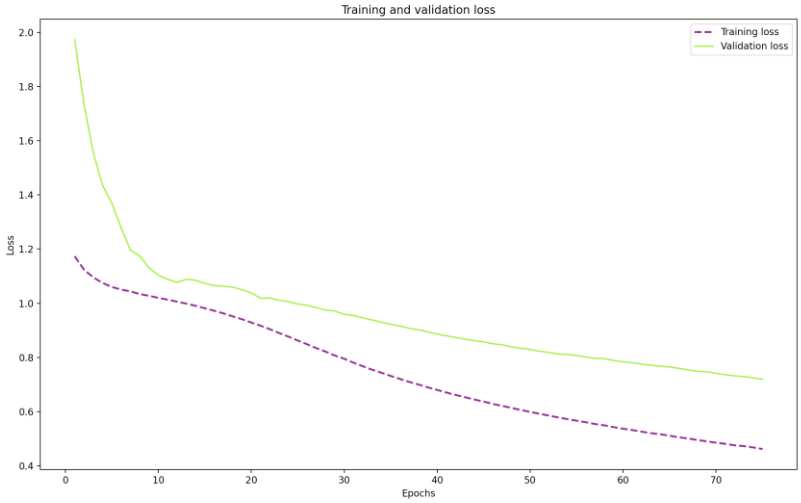
При запусках обучения с параметром validation_split, заключенным в интервале от 0.1 до 0.4, однозначно более оптимального значения выявлено не было, поэтому в дальнейшем было использовано исходное значение = 0.1.

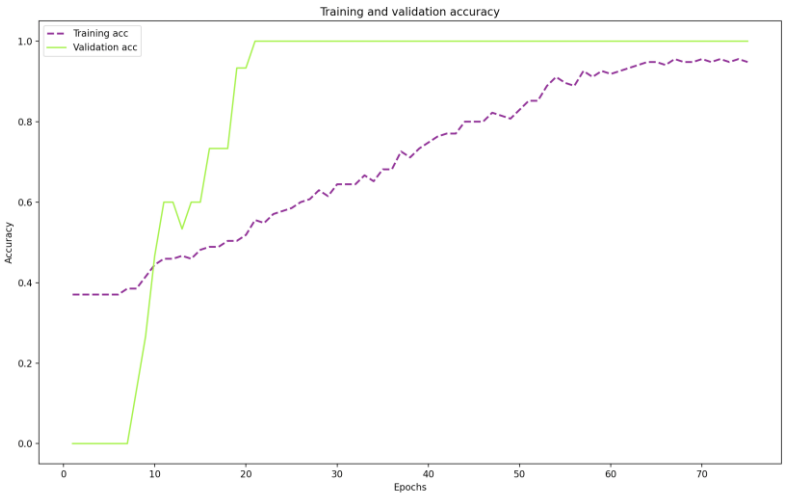
- Увеличение **слоев** ИНС

  Добавим еще один полносвязанный слой с 4 нейронами:

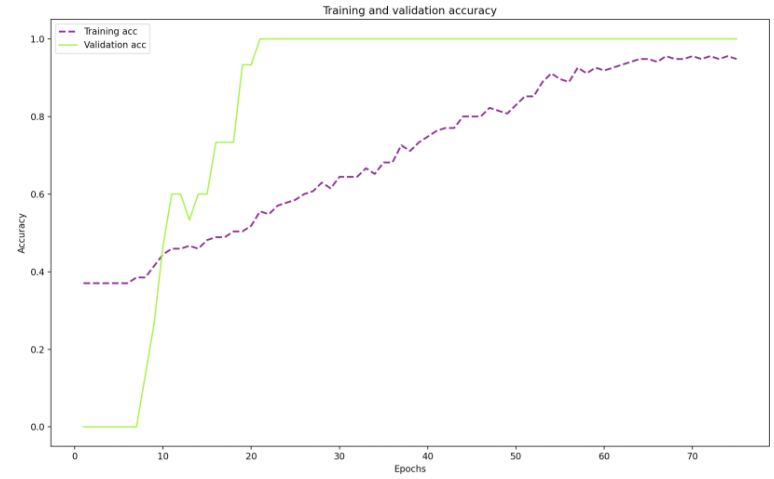
```
model = Sequential()
model.add(Dense(4, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X, dummy_y, epochs=75, batch_size=5, validation_split=0.1)
```

Графики ошибок:





Графики точности:



Вывод:

Значения показателей точности и потери сети, как и в исходной модели, приобретают совершенно различные и непостоянные значения во время нового запуска обучения.

- Увеличение **нейронов** в исходной модели

  Увеличим кол-во нейронов в первом слое вдвое:

```
model = Sequential()
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X, dummy_y, epochs=75, batch_size=5, validation_split=0.1)
```

  График ошибок:

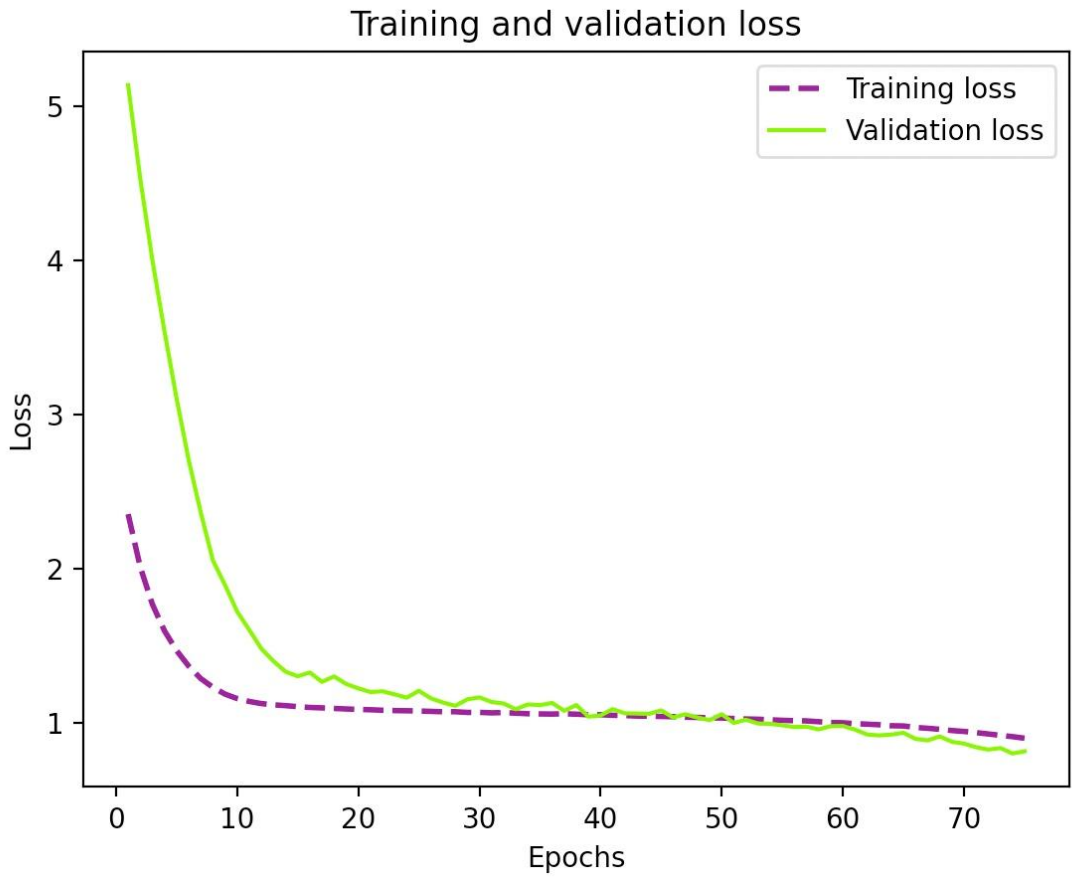
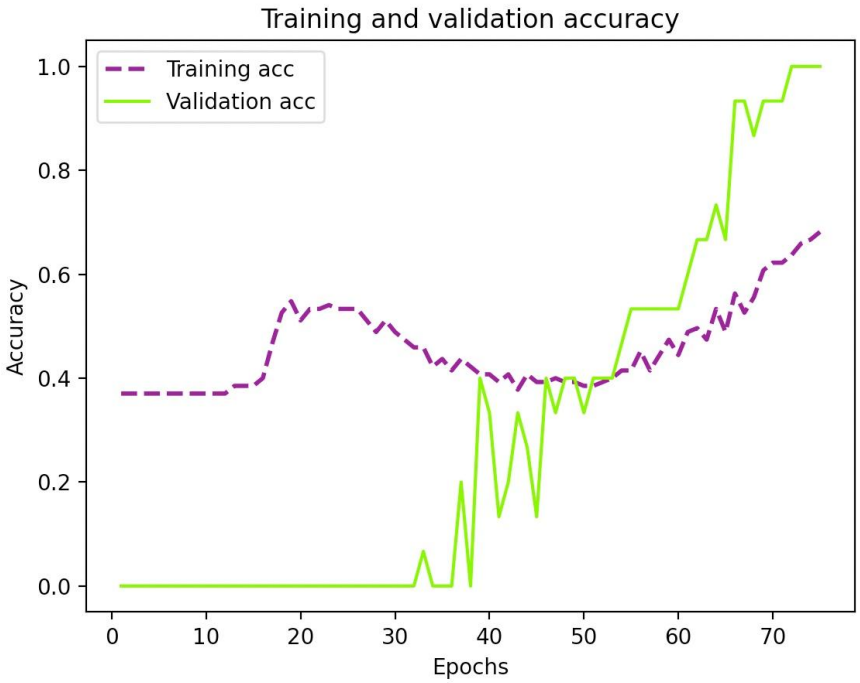


График точности:



Увеличим кол-во нейронов в первом слое до значения 16:

```
model = Sequential()
model.add(Dense(16, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X, dummy_y, epochs=75, batch_size=5, validation_split=0.1)
```

График ошибок:

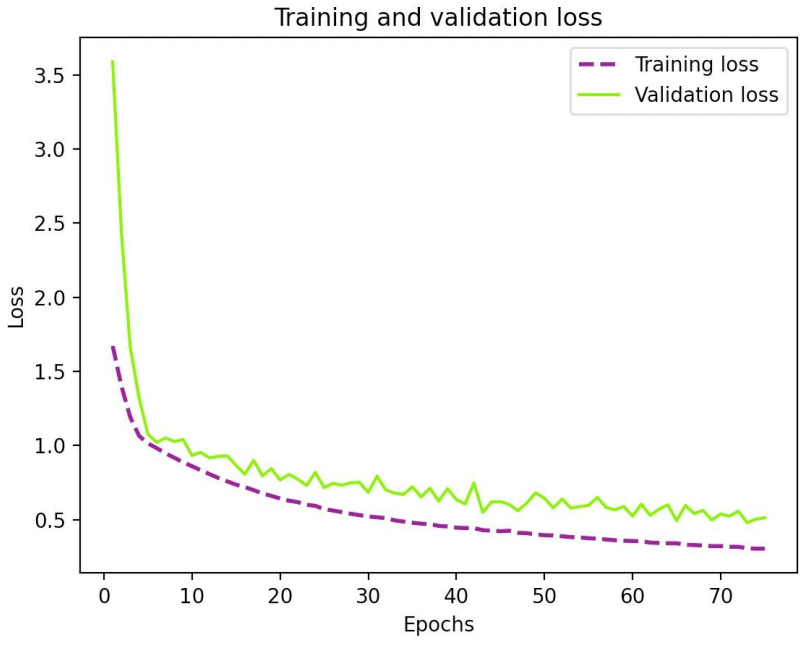
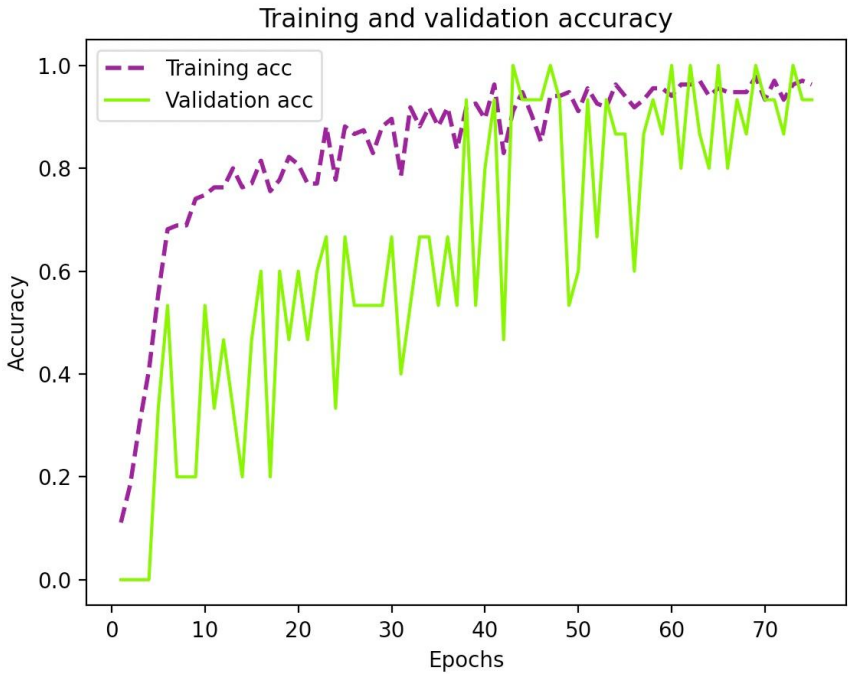


График точности:



Вывод:

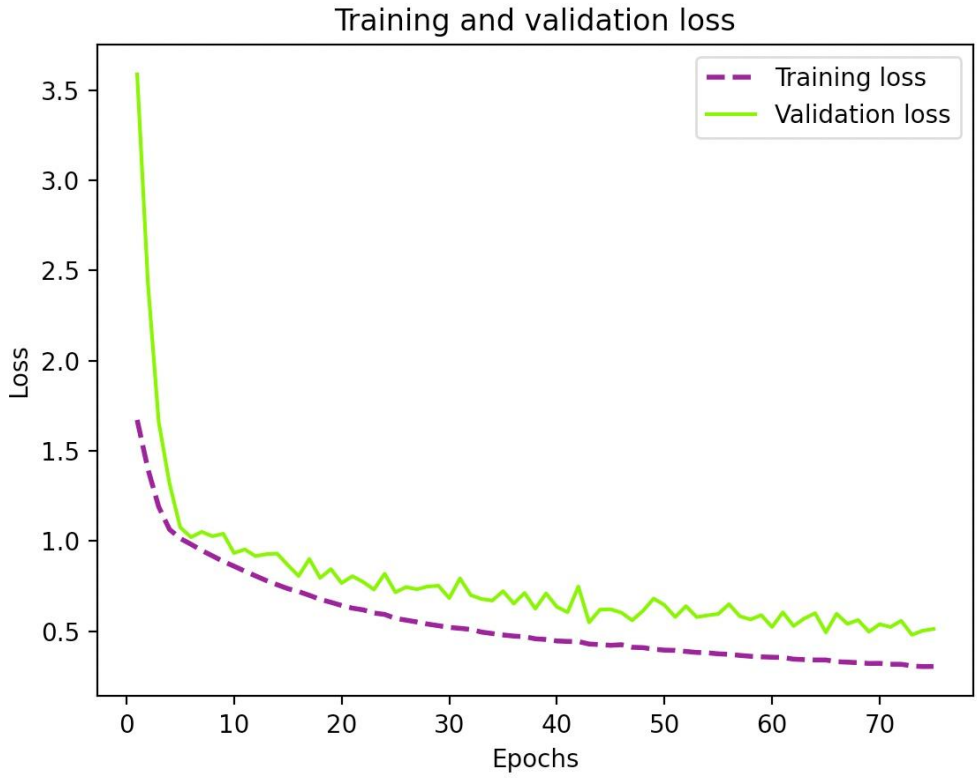
Из графиков видно, что при увеличение числа нейронов в слое показатель потери сети уменьшается, а ее точность увеличивается (в случае 16 нейронов графики точности тестовых и обучающих данных приближены друг к другу).

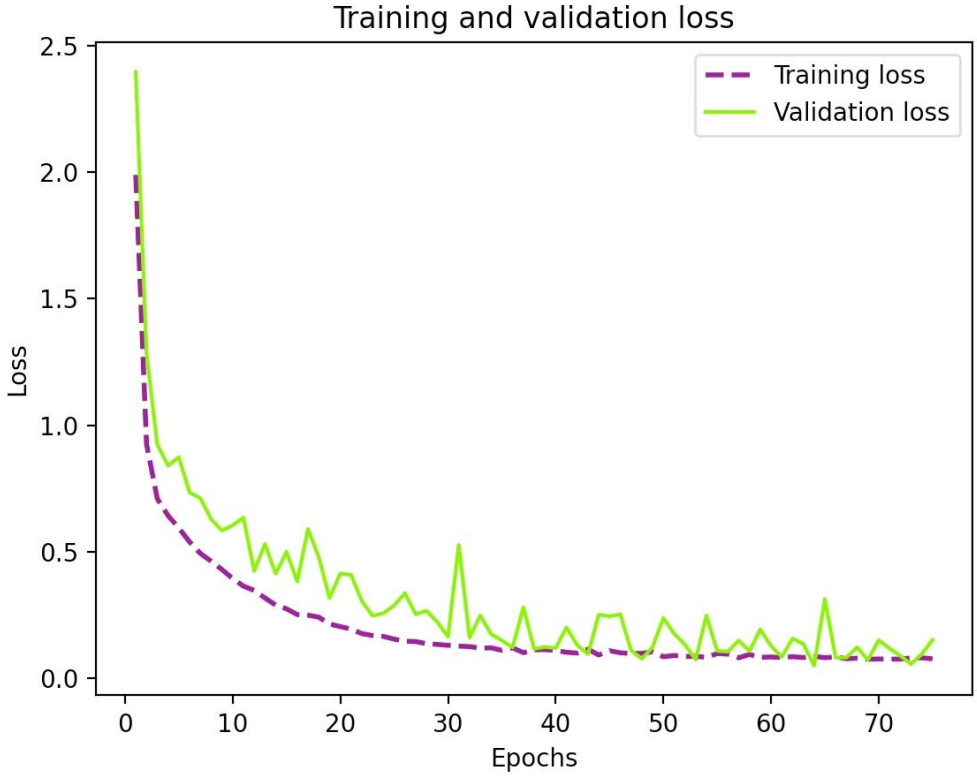
- Увеличение количества слоев и количества нейронов, на них

  Имеем следующую модель:

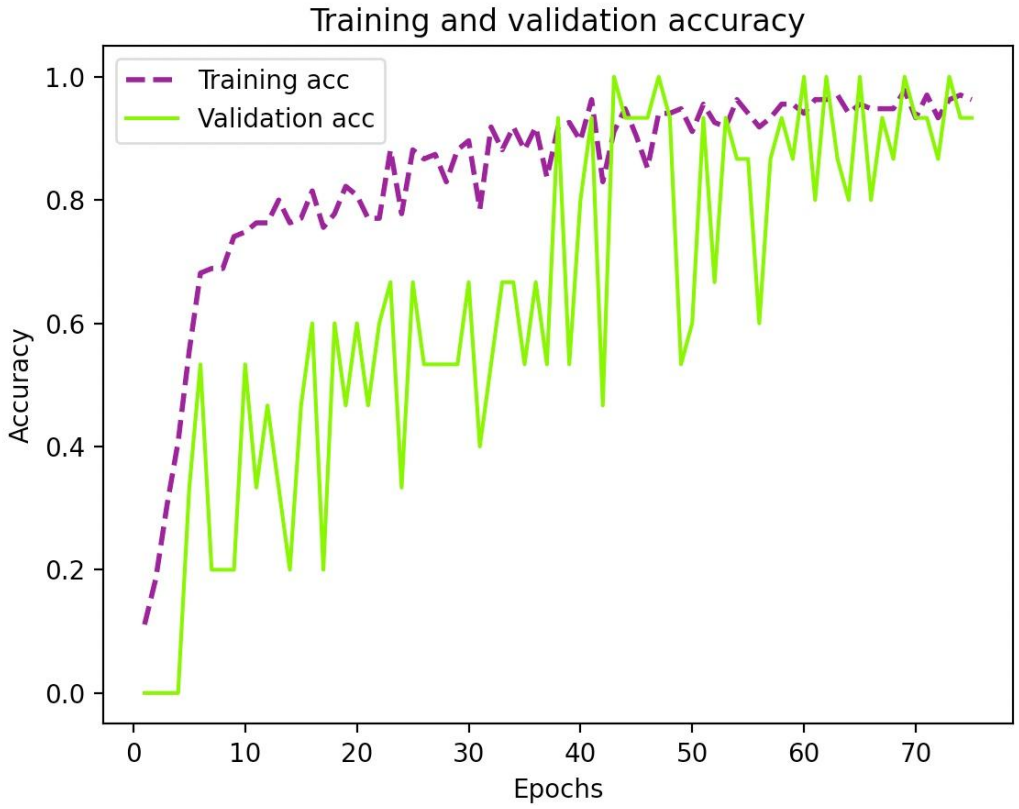
```
model = Sequential()
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X, dummy_y, epochs=75, batch_size=5, validation_split=0.1)
```

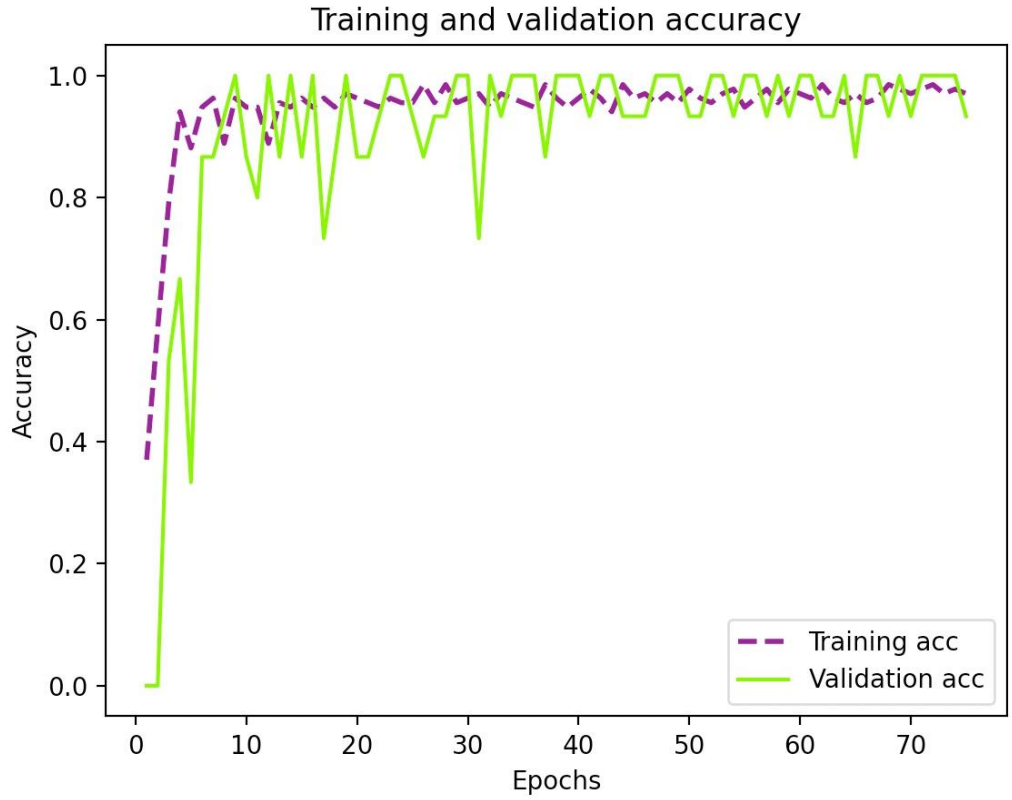
Графики ошибок:





Графики точности:





Вывод:

Как видно из графиков, значения точности сети являются высокими, а ее потери стремятся к нули. При этом, в отличие от предыдущих моделей данная сеть стабильна - при новых запусках обучения значения точности и потери сети практически не изменяются.

9. Наилучшая модель

Оптимальные значения параметров (определенные ранее): *batch_size = 5; epochs = 600; validation_split = 0.1*.

Архитектура с добавлением слоев и увеличением количества нейронов.

```
model = Sequential()
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(X, dummy_y, epochs=600, batch_size=5, validation_split=0.1)
```

График ошибок:

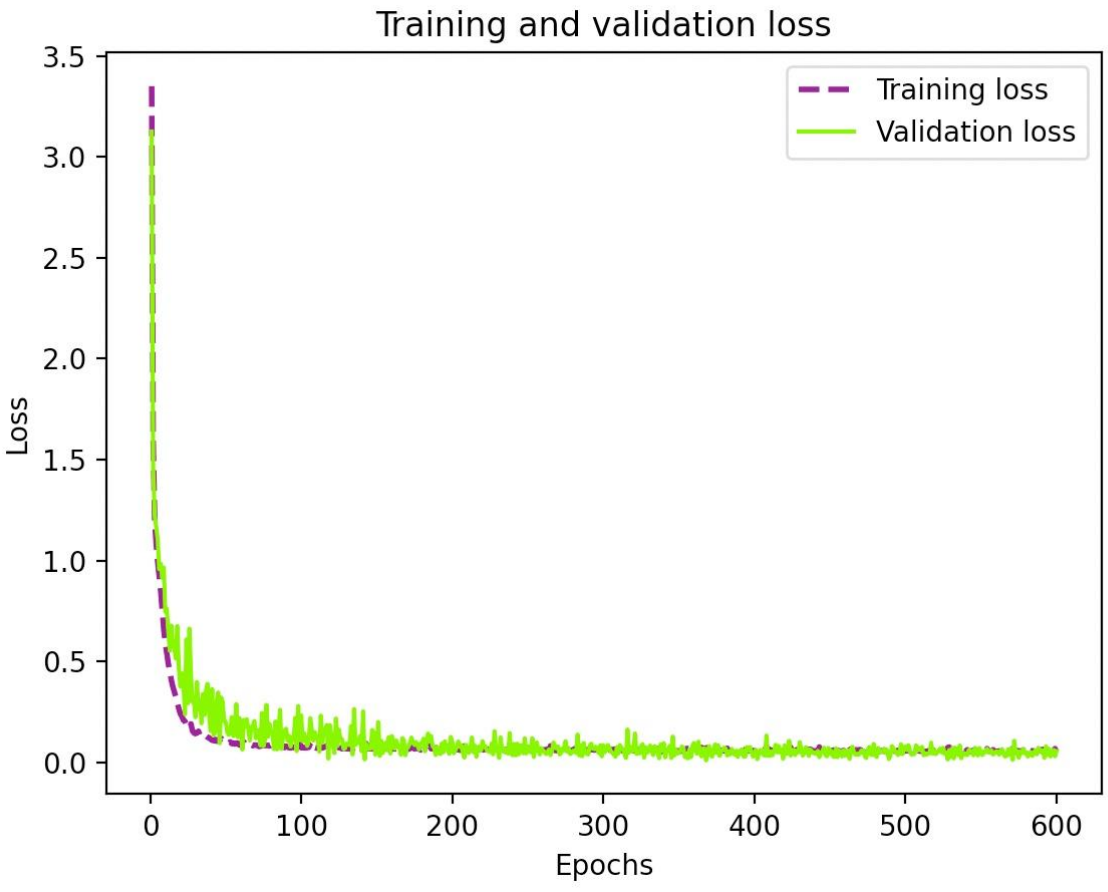
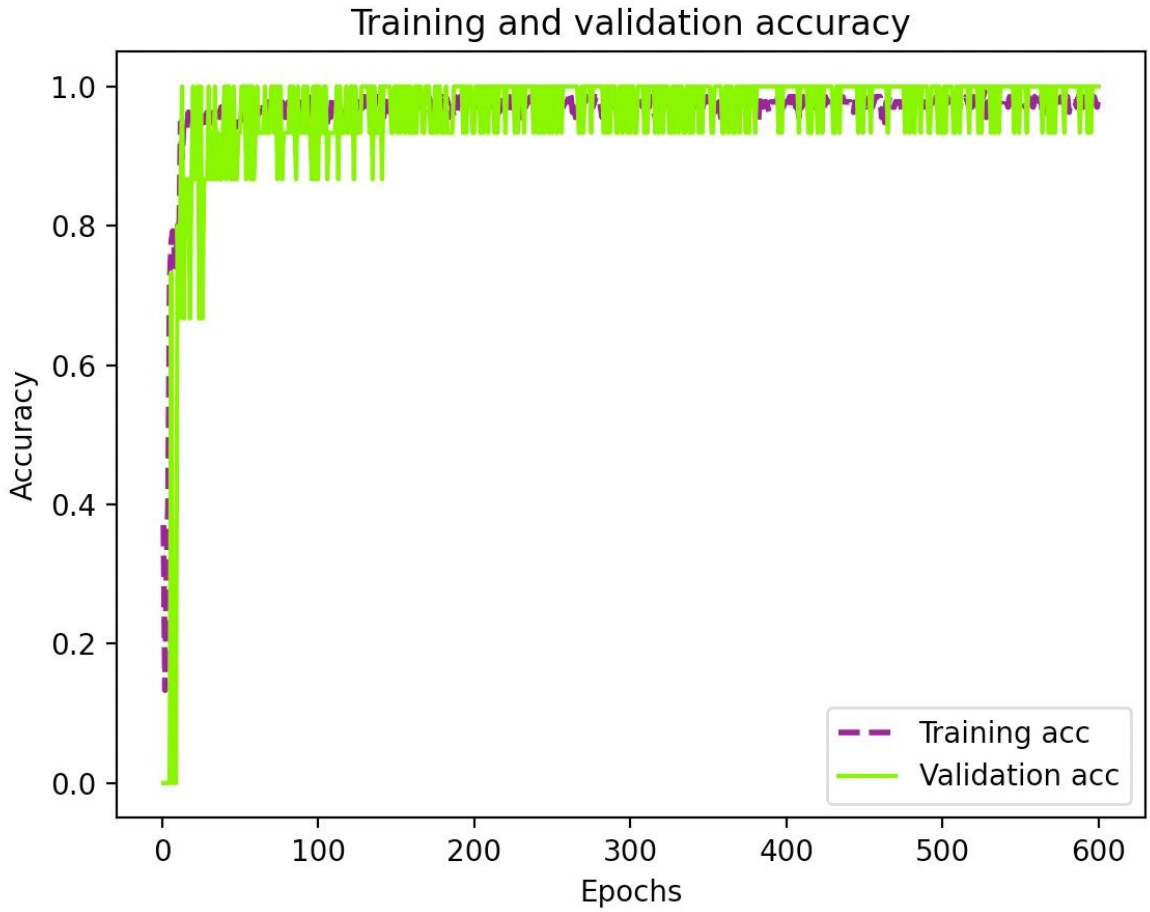


График точности:



Выводы.

В результате выполнения лабораторной работы были изучены различные архитектура ИНС (с разным кол-вом слоев и/или увеличением числа нейронов), было выяснено влияние параметров обучения на работу сети. Для всех моделей были построены графики ошибок и точности в ходе изучения, была выбрана наилучшая модель, сформированная в результате оценки предшествующих моделей и способная решить задачу классификации ирисов с точностью от 90-100%.