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

ОТЧЕТ

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

Студентка гр. 8383 _____ Ишанина Л.Н.

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.

Задание.

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

Изучить различные архитектуры ИНС (Разное кол-во слоев, разное кол-во нейронов на слоях)

Изучить обучение при различных параметрах обучения (параметры ф-ций fit)

Построить графики ошибок и точности в ходе обучения

Выбрать наилучшую модель

Выполнение работы.

Был скачен набор данных (файл iris.data). Скачанный файл был переименован в “iris.csv” и помещен в директорию проекта лабораторной работы. Файл “iris.csv” содержит в себе строки, в каждой из которых 4 числа (размеры пестиков и тычинок) и 1 строка (название сорта растения ирис).

В листинге 1 представлен пример данных:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

Затем были импортированы необходимые для работы классы и функции. Листинг подключения модулей представлен ниже:

```
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
```

Далее были загружены данные из файла и разделены атрибуты (столбцы) на входные данные (X) и выходные данные (Y). Листинг представлен ниже:

```
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
```

В первой строке кода происходит чтение из файла “iris.csv”, а также указан параметр `header=None` который означает, что заголовки в файле отсутствуют. Во второй строке происходит преобразование `dataframe` в удобный для работы формат (массив). В третьей строке происходит отделение входных данных (4 числа от каждой строки) и приведение их к типу float. В четвертой строке происходит отделение выходных данных, то есть четвертого столбца каждой строки, в котором содержится название сорта растения ирис.

Так как данная задача является задачей многоклассовой классификации, то следующим шагом было сделано преобразование выходных атрибутов из вектора в матрицу к виду представленных в листинге ниже:

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

Для этого был совершен переход от текстовых меток к категориальному вектору, листинг представлен ниже:

```
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
dummy_y = to_categorical(encoded_Y)
```

Функция `to_categorical()` совершает преобразование выходных атрибутов к виду, который упоминался в крайнем представленном листинге.

Затем была создана простая модель, состоящая из последовательности двух слоев Dense, которые являются тесно связанными нейронными слоями. Первый слой – это 4-перменный слой, обрабатывающий входные данные функцией relu. Последний слой — это 3-переменный слой потерь (softmax layer), возвращающий массив с 3 оценками вероятностей (в сумме дающих 1). Каждая оценка определяет вероятность принадлежности текущего изображения к одному из 3 классов цветов. Листинг представлен ниже:

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

Далее были инициализированы параметры обучения и проведено сам процесс обучения сети. Листинг приведен ниже:

```
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])
H = model.fit(X, dummy_y, epochs=75, batch_size=10,
validation_split=0.1)
```

В процессе обучения отображаются четыре величины: потери сети на обучающих данных и точность сети на обучающих данных, а также потери и точность на данных, не участвовавших в обучении.

Для построения графика ошибок и точности в ходе обучения была подключена библиотека matplotlib. Листинг подключения представлен ниже:

```
import matplotlib.pyplot as plt
```

Затем были реализованы сами графики. Листинг приведен ниже:

```
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, 'm*', 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()
plt.plot(epochs, acc, 'm*', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Анализ различных искусственных нейронных сетей.

Изначально, была простая модель, состоящая из последовательности двух слоев.

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

Лучший результат тестирования представлен на рис.1-2.

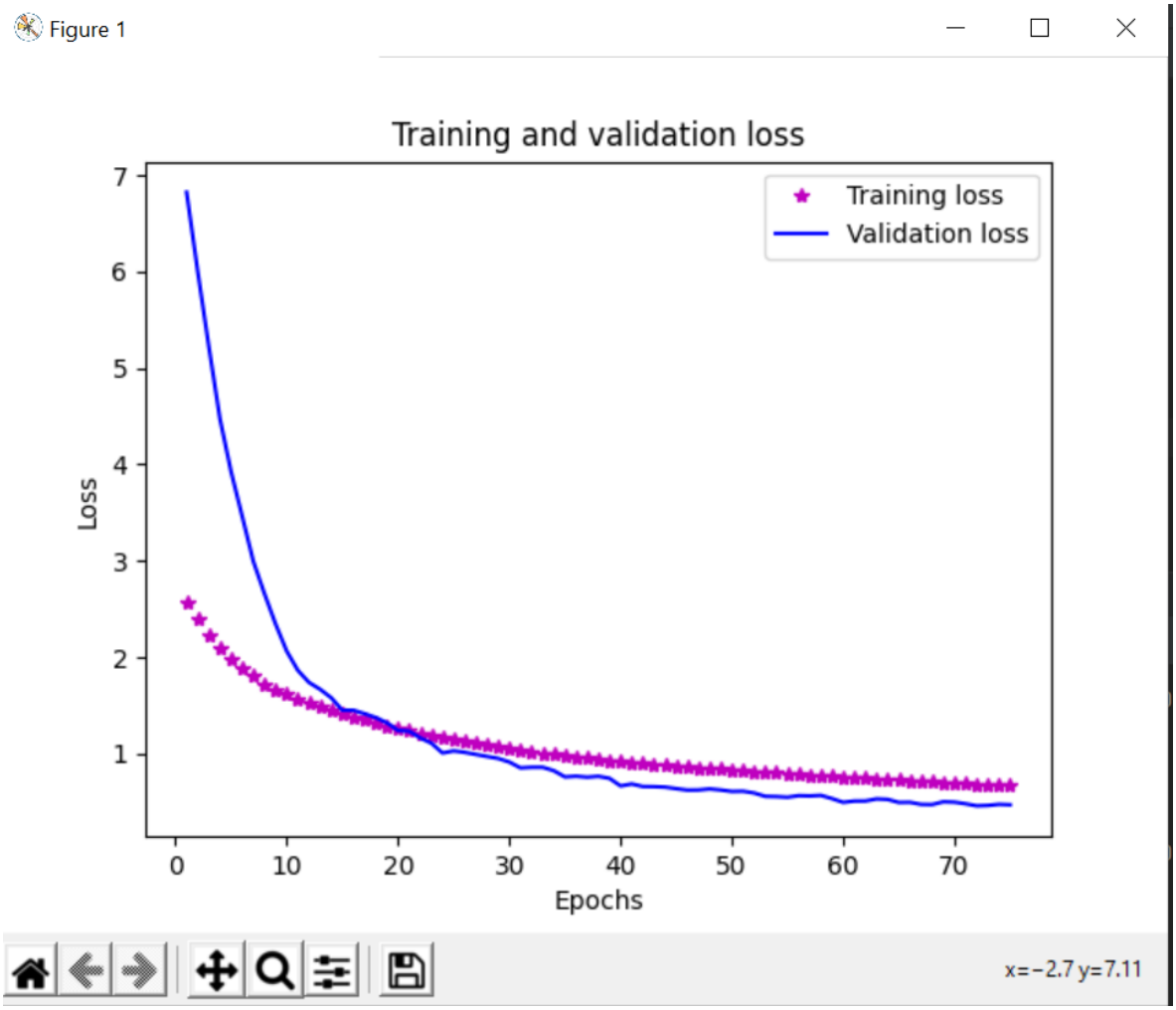

Рисунок 1 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

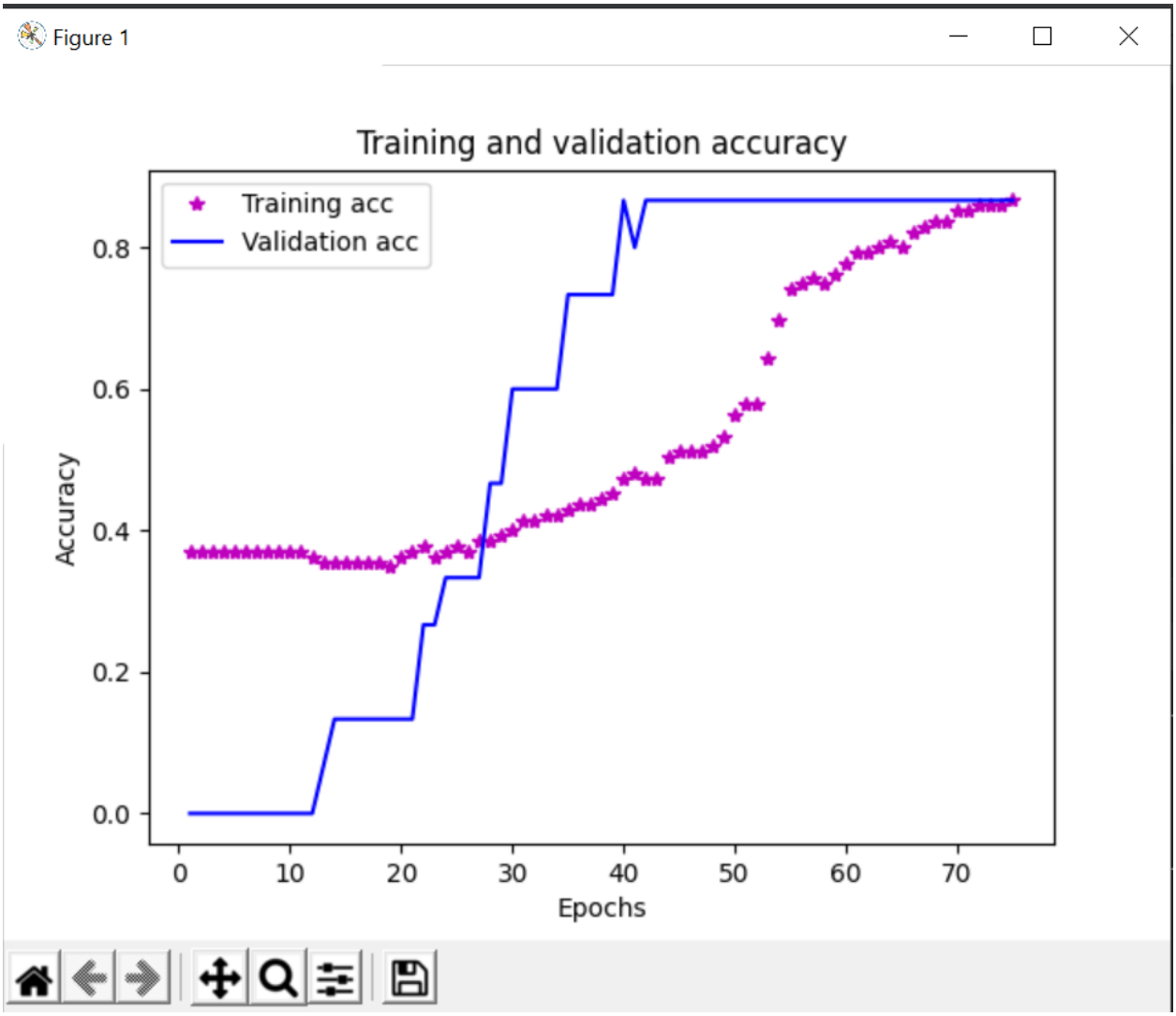

Рисунок 2 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из полученных результатов, видно, что и на обучающих данных, и на данных, не участвовавших в обучении, точность невысокая (~ 0.85) и потери довольно высоки. Следовательно, обучение не очень эффективное.

Далее был добавлен один скрытый слой. Листинг приведен ниже:

```
model.add(Dense(4, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Лучший результат тестирования представлен на рис.3-4.

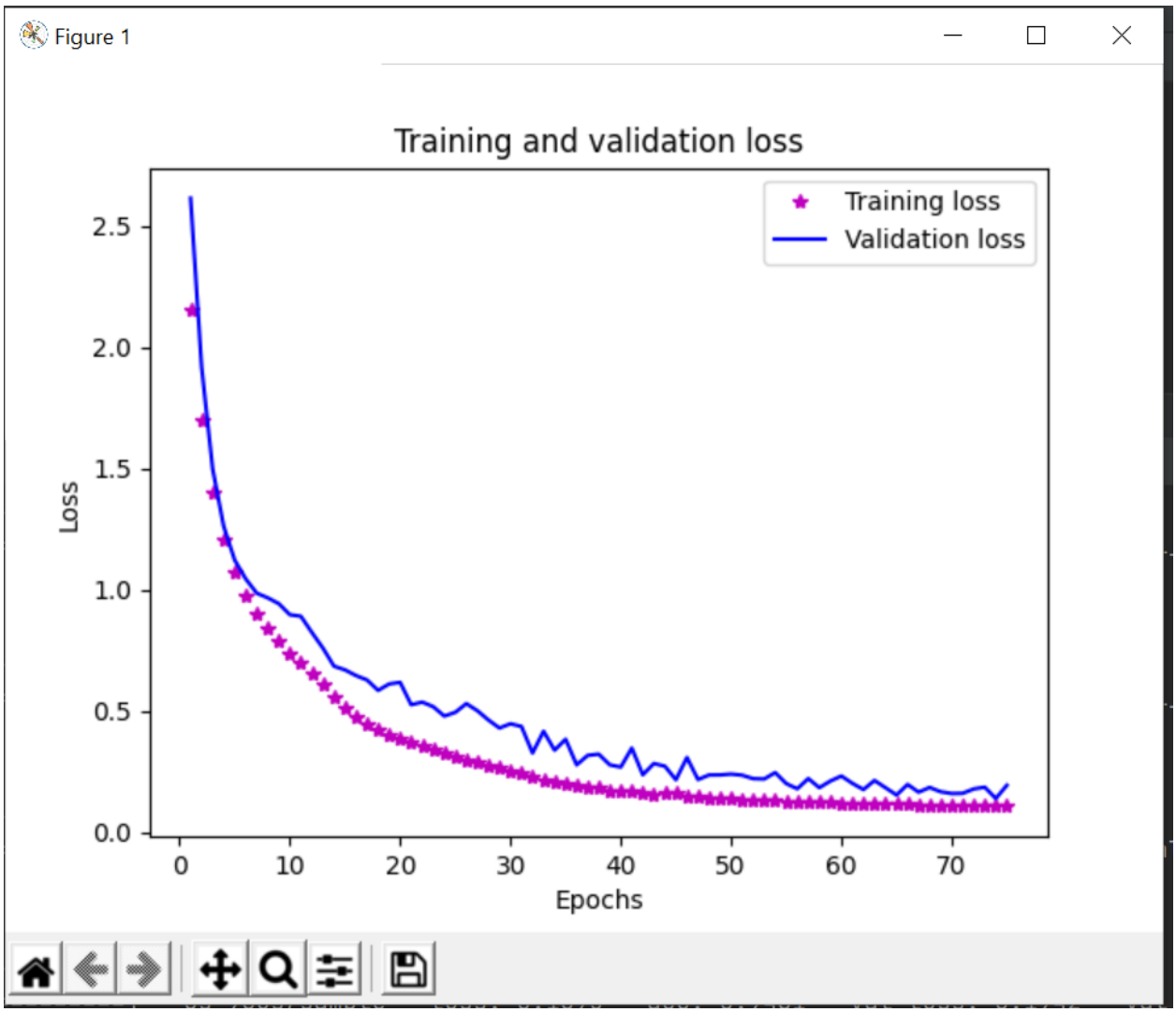

Рисунок 3 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

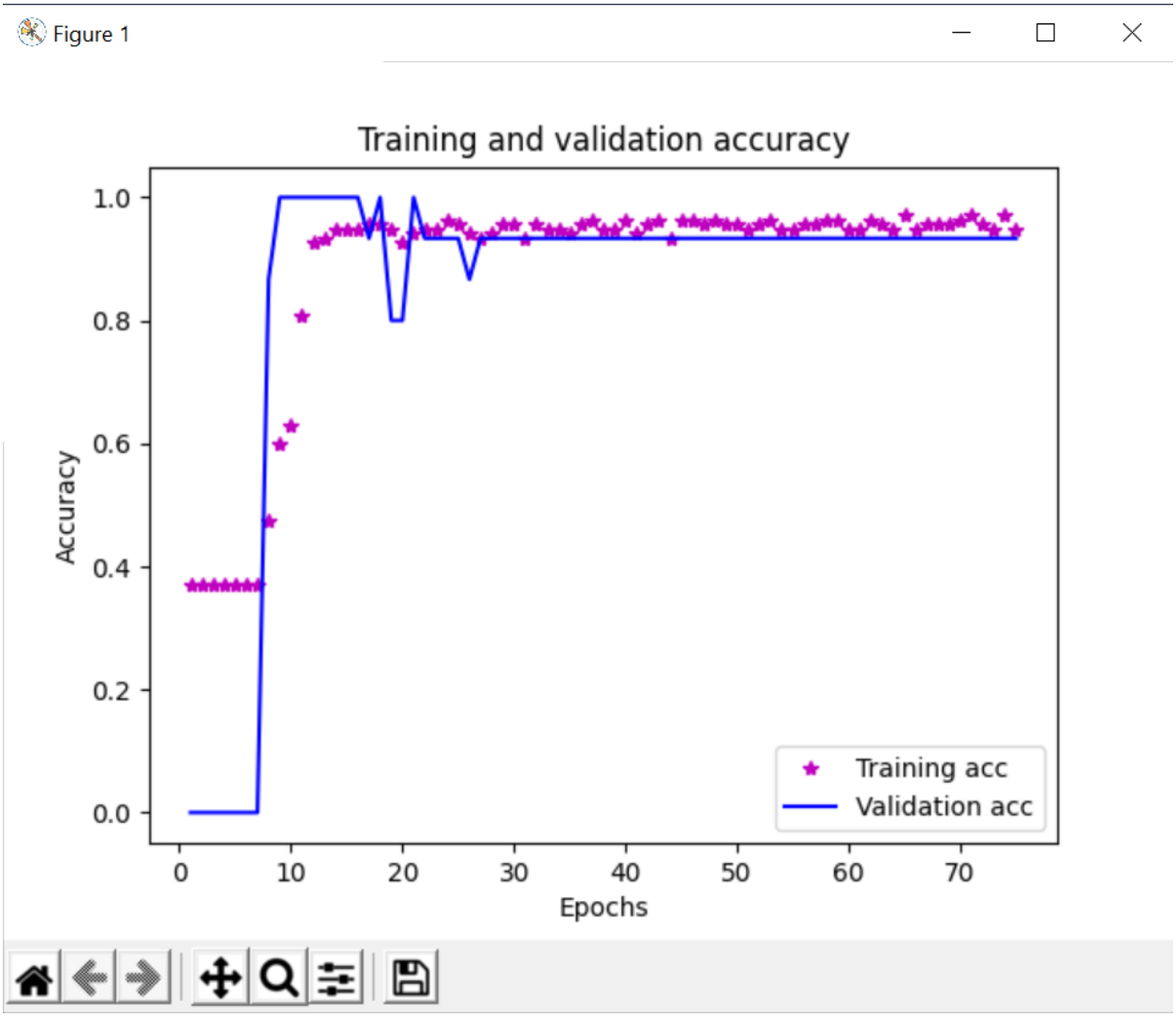

Рисунок 4 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

ИНС достигла следующих показателей на 26 эпохе:

loss: 0.3120 - acc: 0.9556 - val_loss: 0.4950 - val_acc: 0.9333

Следовательно, можно сделать вывод, что добавление одного скрытого слоя с количеством нейронов 30 привело к тому, что точность сети значительно улучшилась. Показатели потерь тоже уменьшились, но всё равно достаточно велики.

Далее, в ходе анализа ИНС, количество слоев остается прежним, но количество нейронов будет изменяться, чтобы понять, как количество нейронов в скрытом слое влияет на качество обучаемости ИНС.

Изменим количество нейронов в скрытом слое с 30 на 100. Листинг приведен ниже:

```
model.add(Dense(4, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Лучший результат тестирования представлен на рис.5-6.

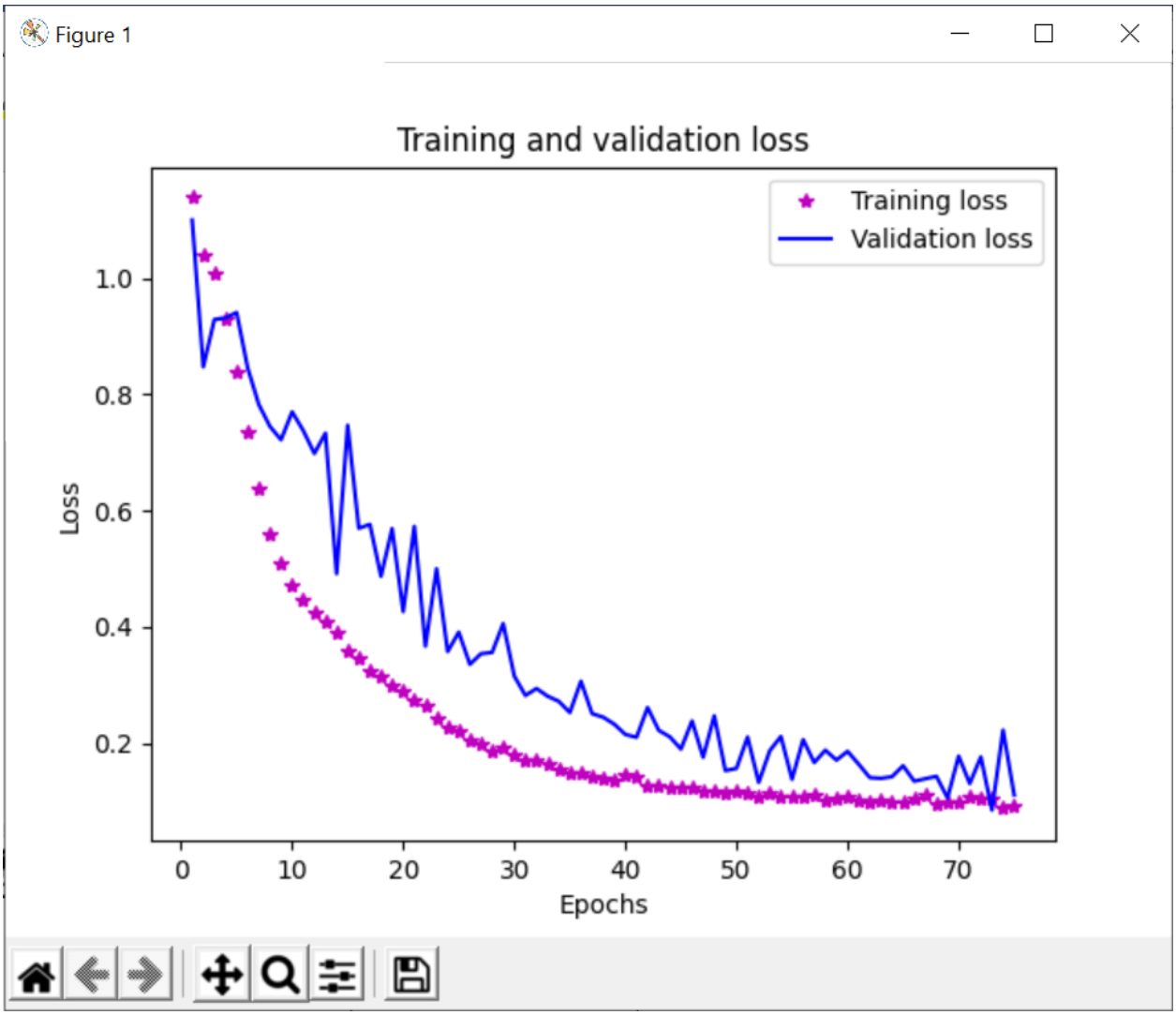

Рисунок 5 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

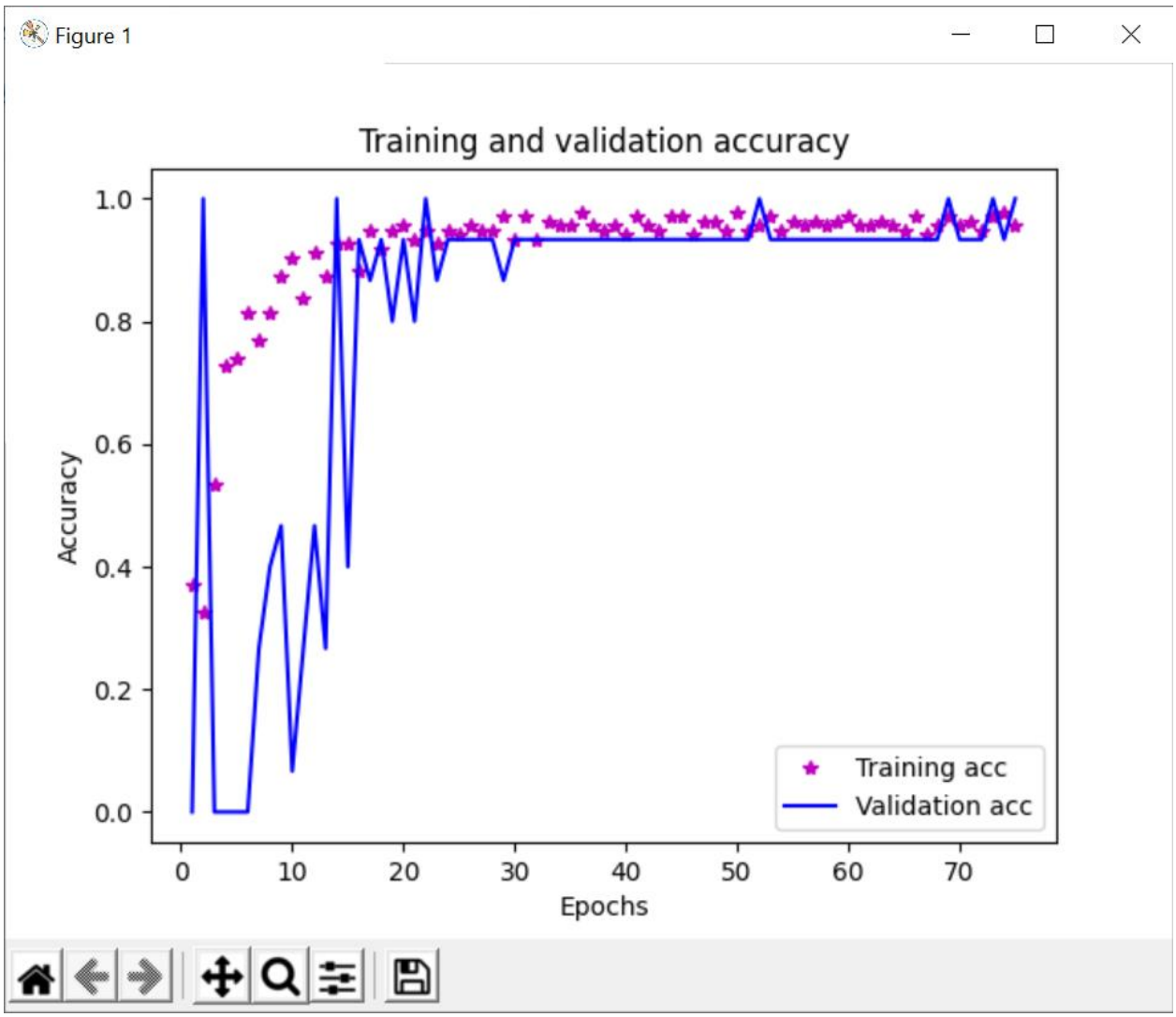

Рисунок 6 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из графиков, видно, что ИНС обучается уже примерно к 25 эпохе. На 25 эпохе её показатели достигают:

loss: 0.2266 - acc: 0.9481 - val_loss: 0.3574 - val_acc: 0.9333

Сравнив, с предыдущими значениями, можно сделать вывод что все показатели ещё немного улучшились. А значит, увеличение числа нейронов улучшает как значение потери, так и точность ИНС. Что касаемо скорости обучения, то она совсем немного увеличилась, практически не изменилась.

Изменим количество нейронов в скрытом слое со 100 до 200. Листинг приведен ниже:

```
model.add(Dense(4, activation='relu'))
model.add(Dense(200, activation='relu'))
```

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

Лучший результат тестирования представлен на рис.7-8.

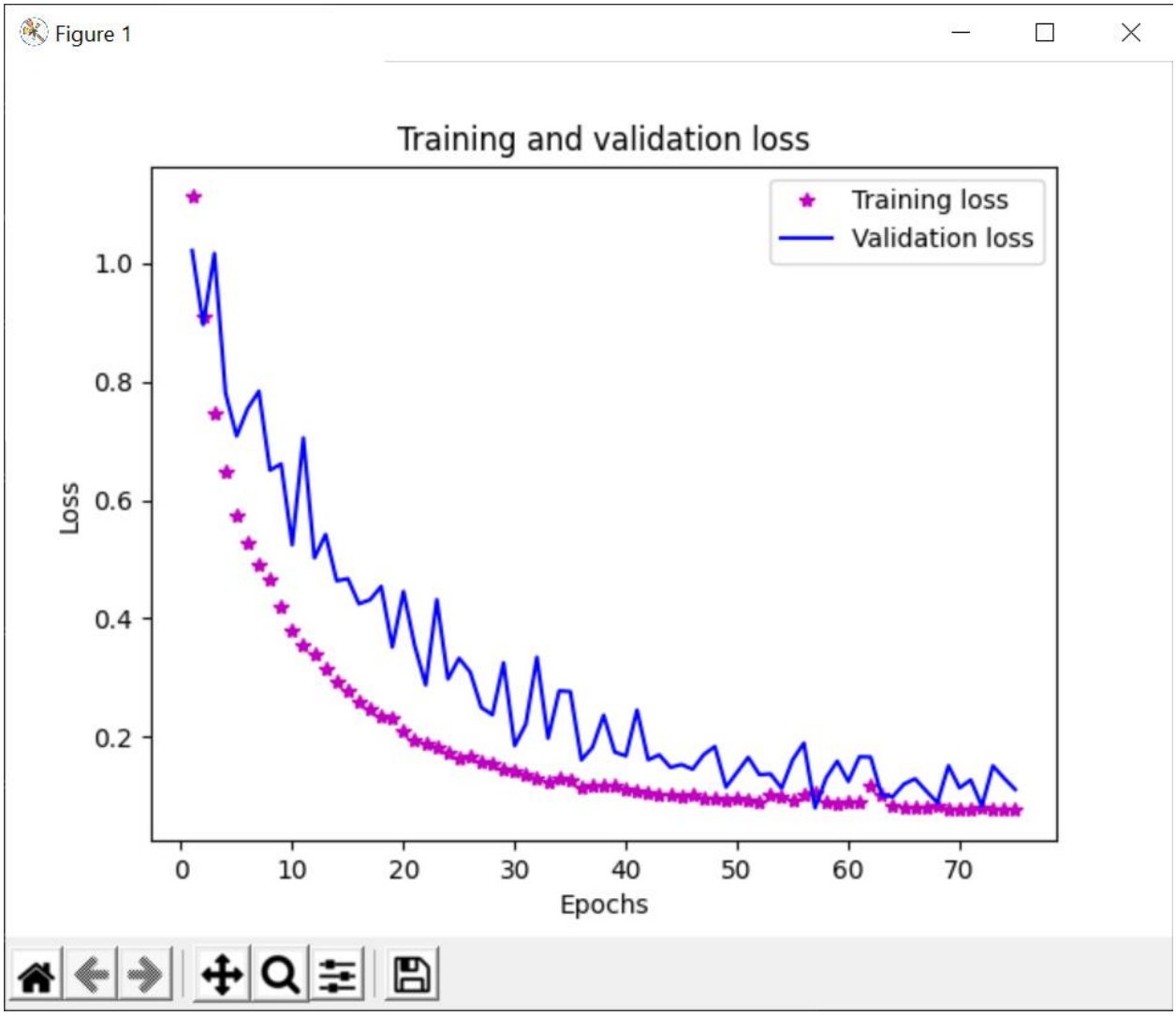

Рисунок 7 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

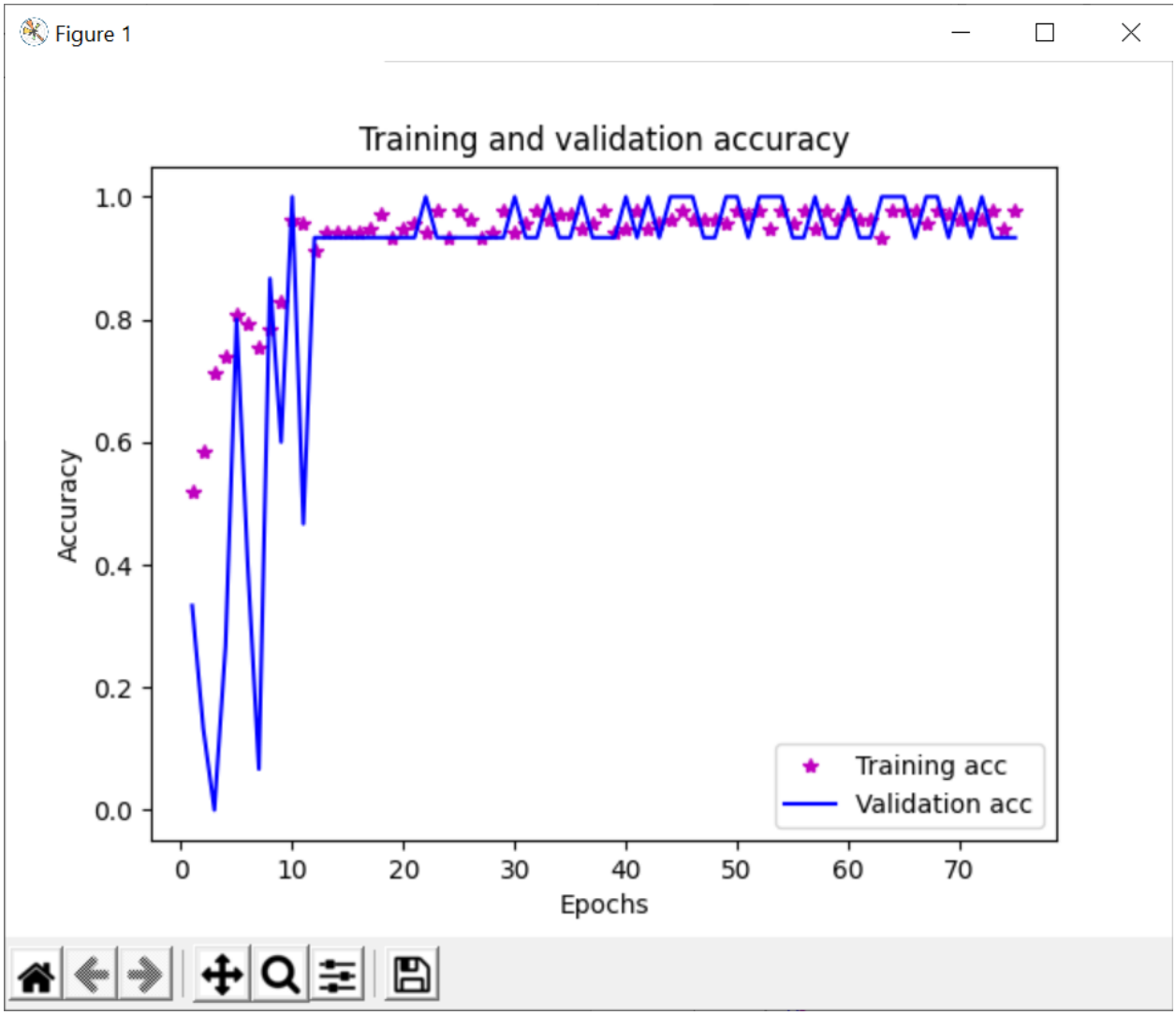

Рисунок 8 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Как видно из рис.8, точность увеличилась, кроме того, значение validation acc стало ещё чаще достигать 100%. А на рис.7 видно, что значение потерь также стало быстрее уменьшаться.

Исходя из графиков, видно, что ИНС обучается уже примерно к 23 эпохе. На 23 эпохе её показатели достигают:

loss: 0.1881 - acc: 0.9407 - val_loss: 0.2877 - val_acc: 1.0000

По сравнению с предыдущими значениями, видно, что действительно показатели улучшились.

Далее было изменено количество нейронов в скрытом слое с 200 до 500. Листинг приведен ниже:

```
model.add(Dense(4, activation='relu'))
model.add(Dense(500, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Лучший результат тестирования представлен на рис.9-10.

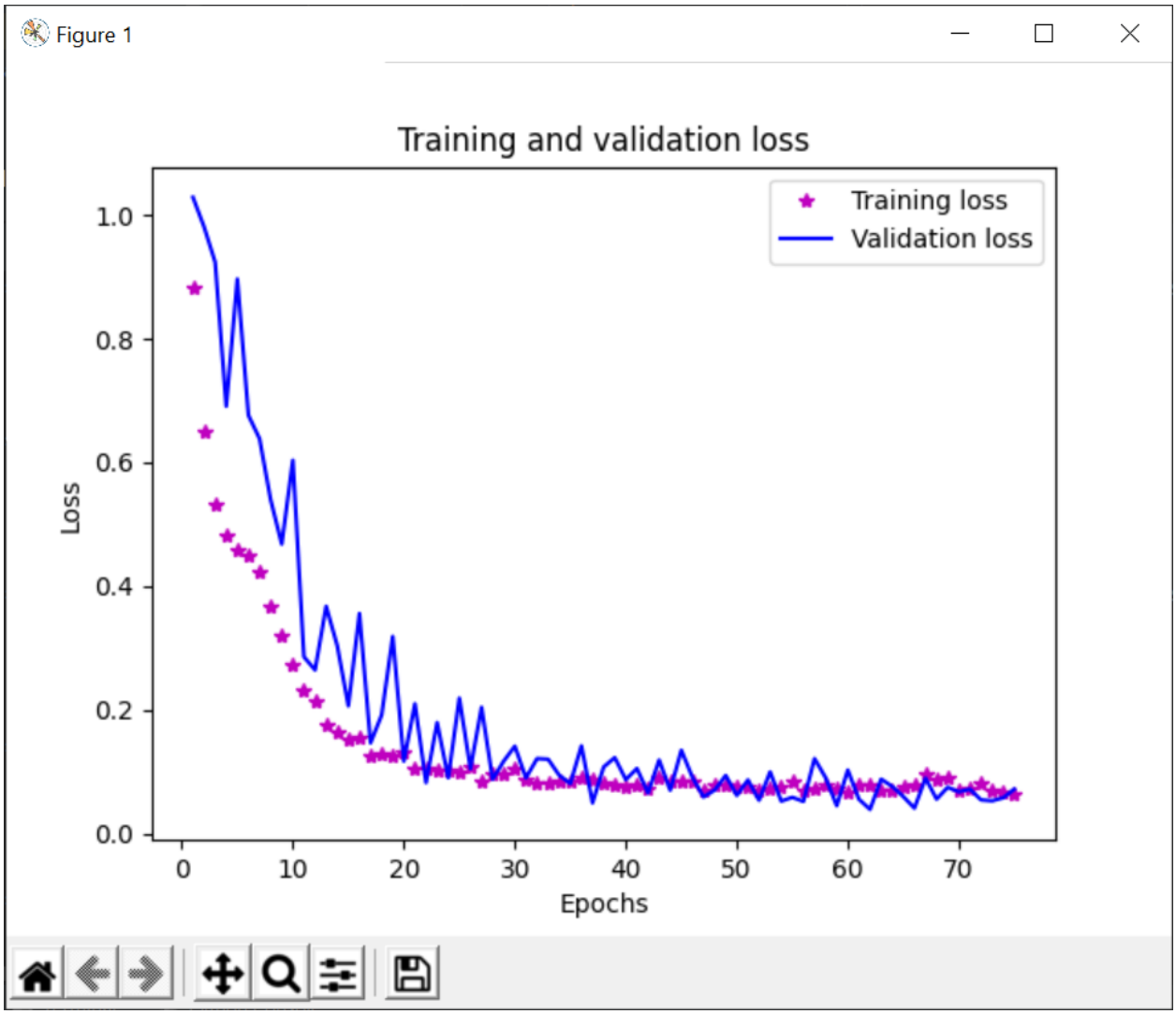

Рисунок 9 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

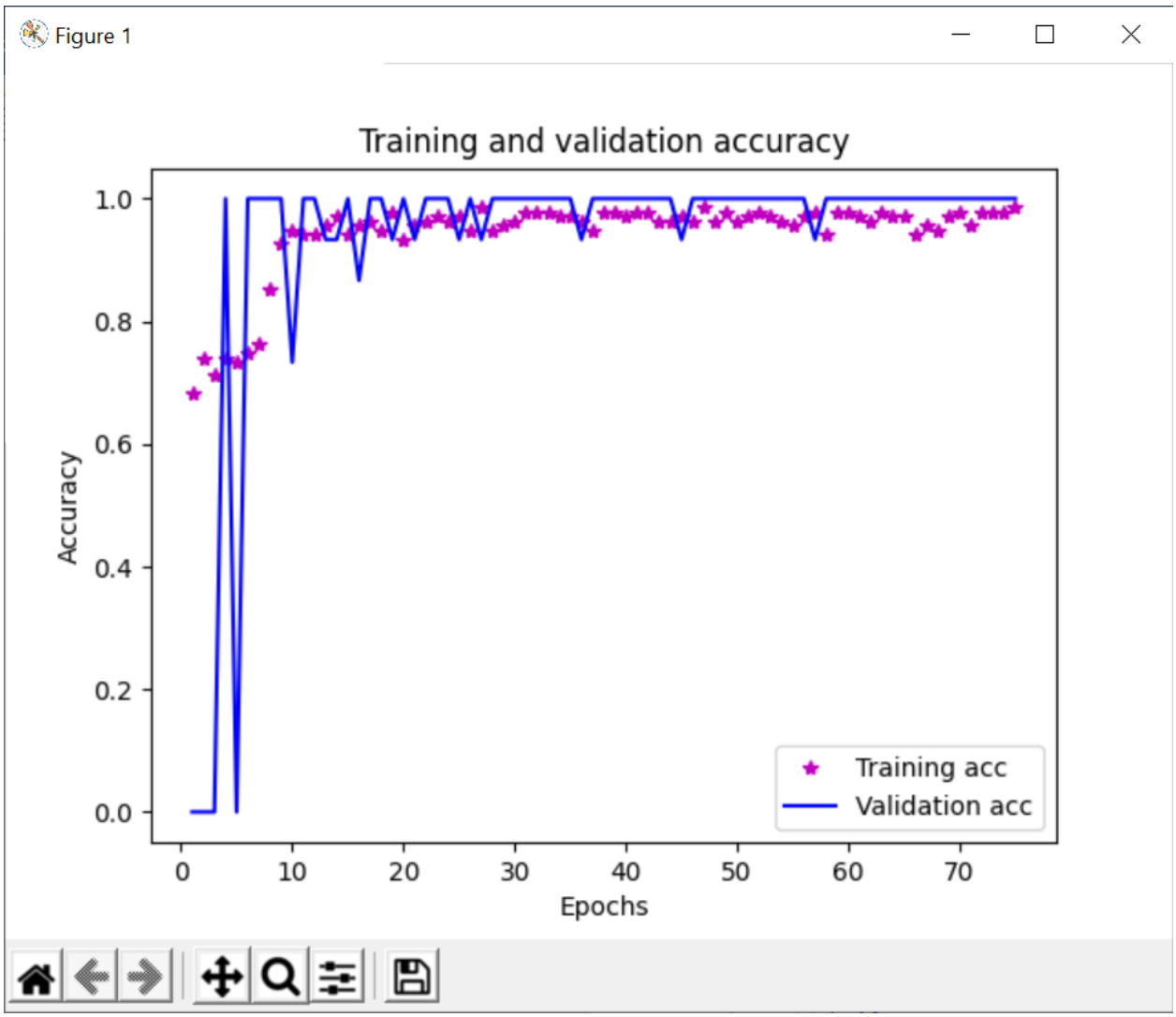

Рисунок 10 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из графиков, видно, что ИНС обучается уже примерно к 22 эпохе. На 22 эпохе её показатели достигают:

loss: 0.1060 - acc: 0.9556 - val_loss: 0.2098 - val_acc: 0.9333

По сравнению с предыдущими значениями показатели очень схожи. А также исходя из графиков видно, что при 500 нейронах результат не сильно отличается от 200. А значит, дальнейшее увеличение нейронов не дает более хороших результатов.

В таком случае можно оставить количество нейронов равное 200.

Далее был добавлен ещё один скрытый слой. Листинг приведен ниже:

```
model.add(Dense(4, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Лучший результат тестирования представлен на рис.11-12.

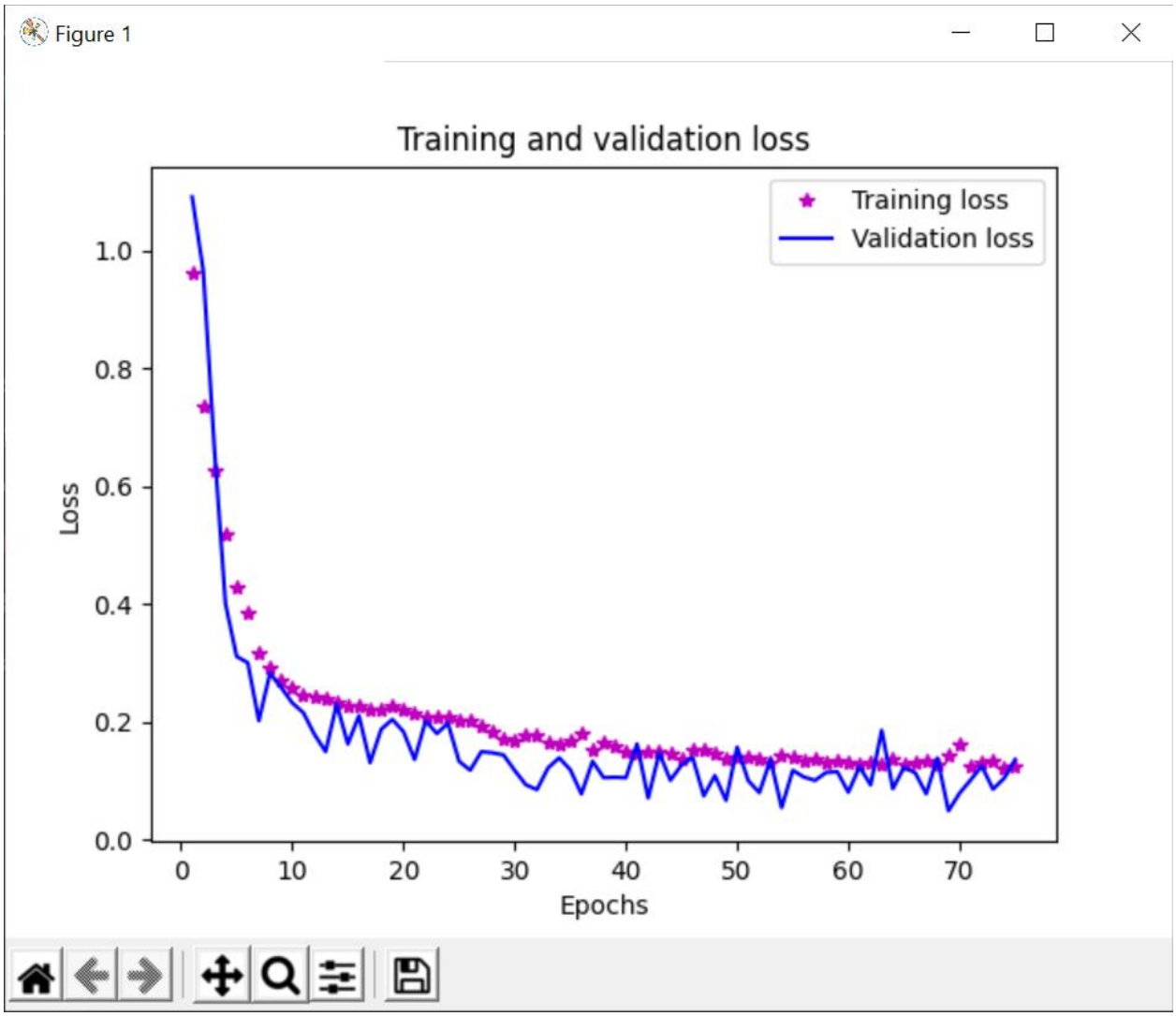

Рисунок 11 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

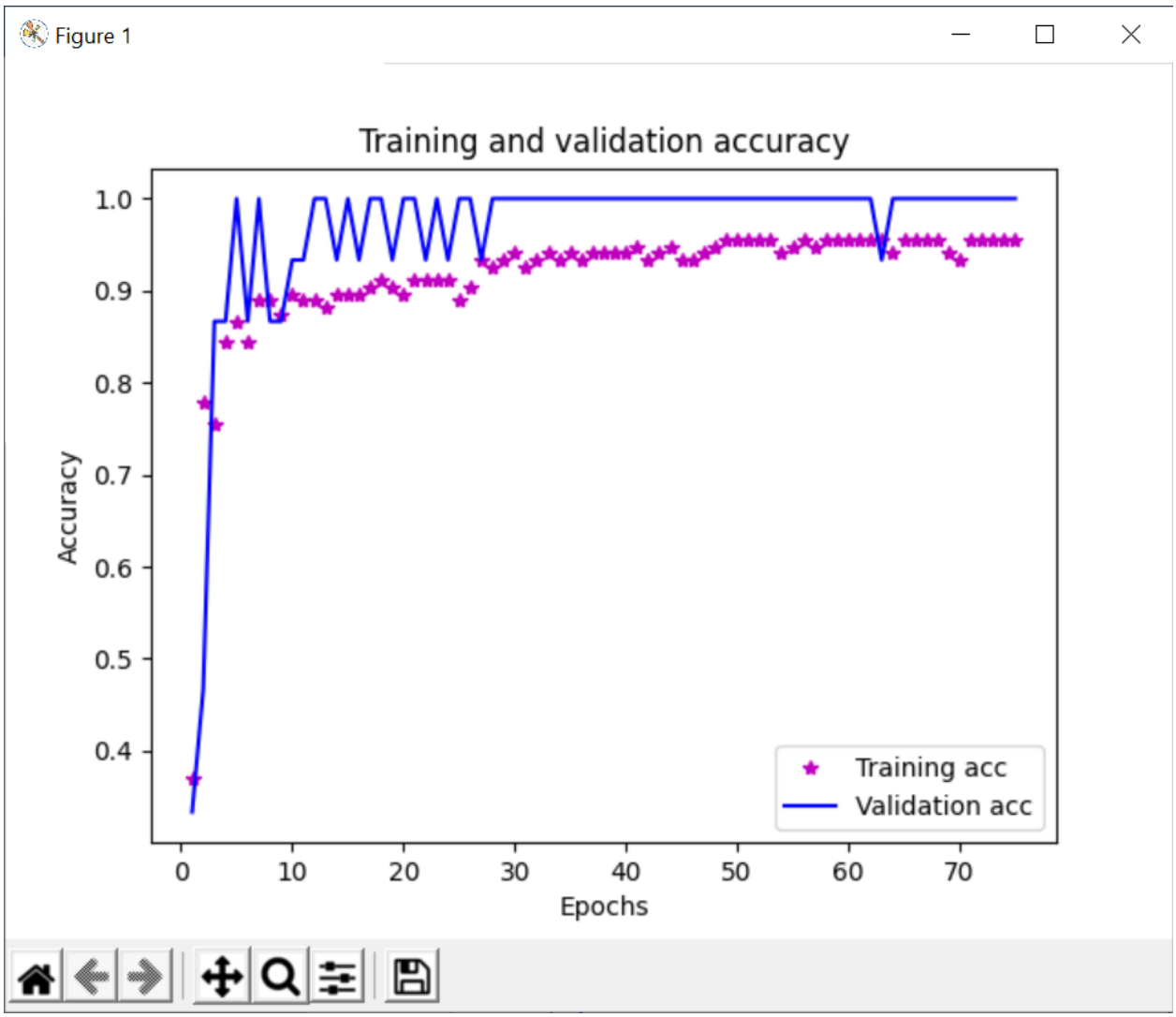

Рисунок 12 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из графиков, видно, что ИНС обучается уже примерно к 16 эпохе. На 16 эпохе её показатели достигают:

loss: 0.2270 - acc: 0.8963 - val_loss: 0.1624 - val_acc: 1.0000

По сравнению с одним слоем два слоя приводят к более быстрому процессу обучения ИНС.

Далее был добавлен ещё один скрытый слой. Листинг приведен ниже:

```
model.add(Dense(4, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(200, activation='relu'))
```

```
model.add(Dense(200, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Лучший результат тестирования представлен на рис.13-14.

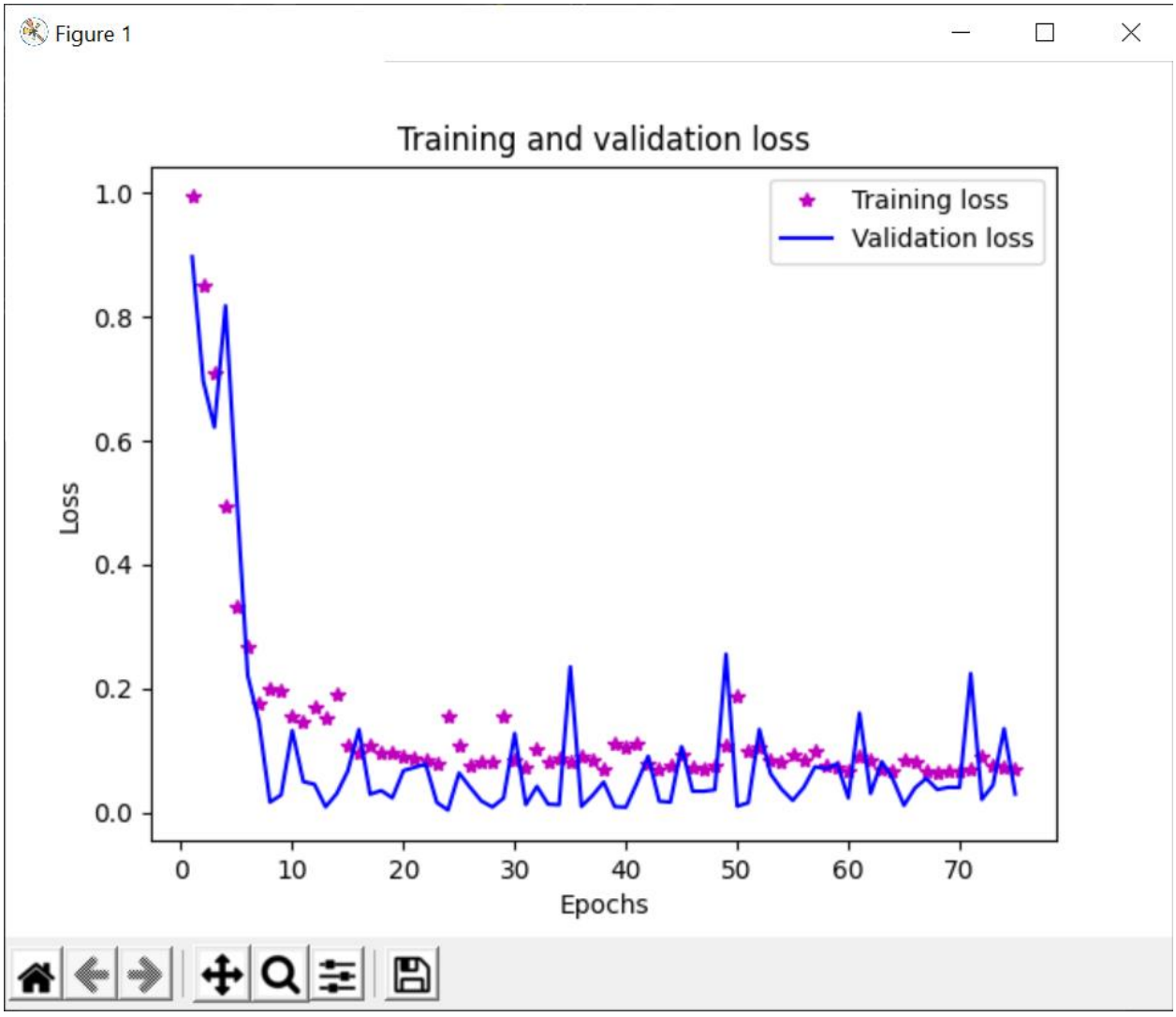

Рисунок 13 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

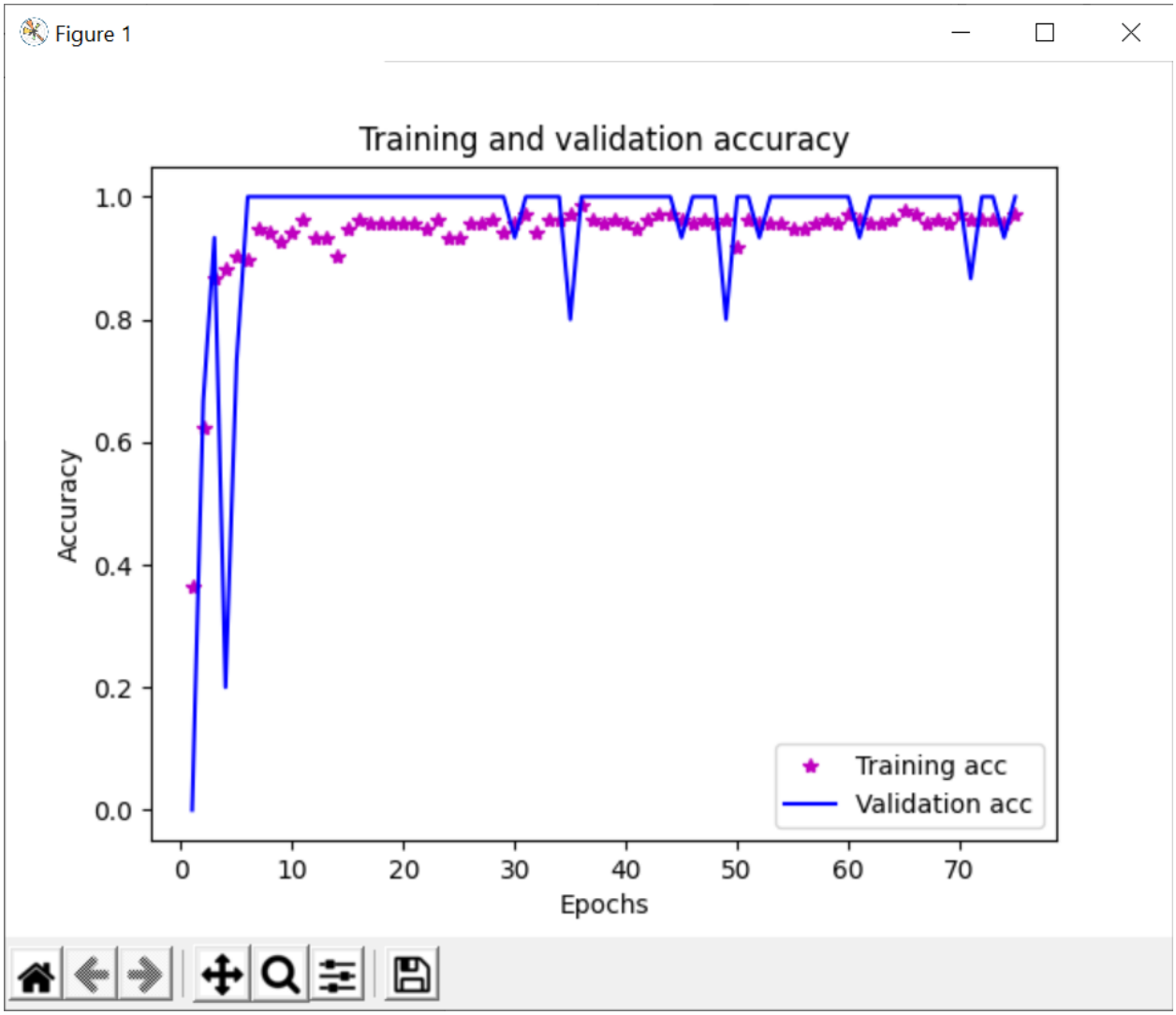

Рисунок 14 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из графиков, видно, что ИНС обучается уже примерно к 9 эпохе. На 9 эпохе её показатели достигают:

loss: 0.1997 - acc: 0.9407 - val_loss: 0.0163 - val_acc: 1.0000

Полученные результаты ещё немного лучше, чем результаты с двумя слоями, а также процесс обучения стал быстрее.

Далее был добавлен ещё один скрытый слой. Листинг приведен ниже:

```
model.add(Dense(4, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(200, activation='relu'))
```

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

Лучший результат тестирования представлен на рис.15-16.

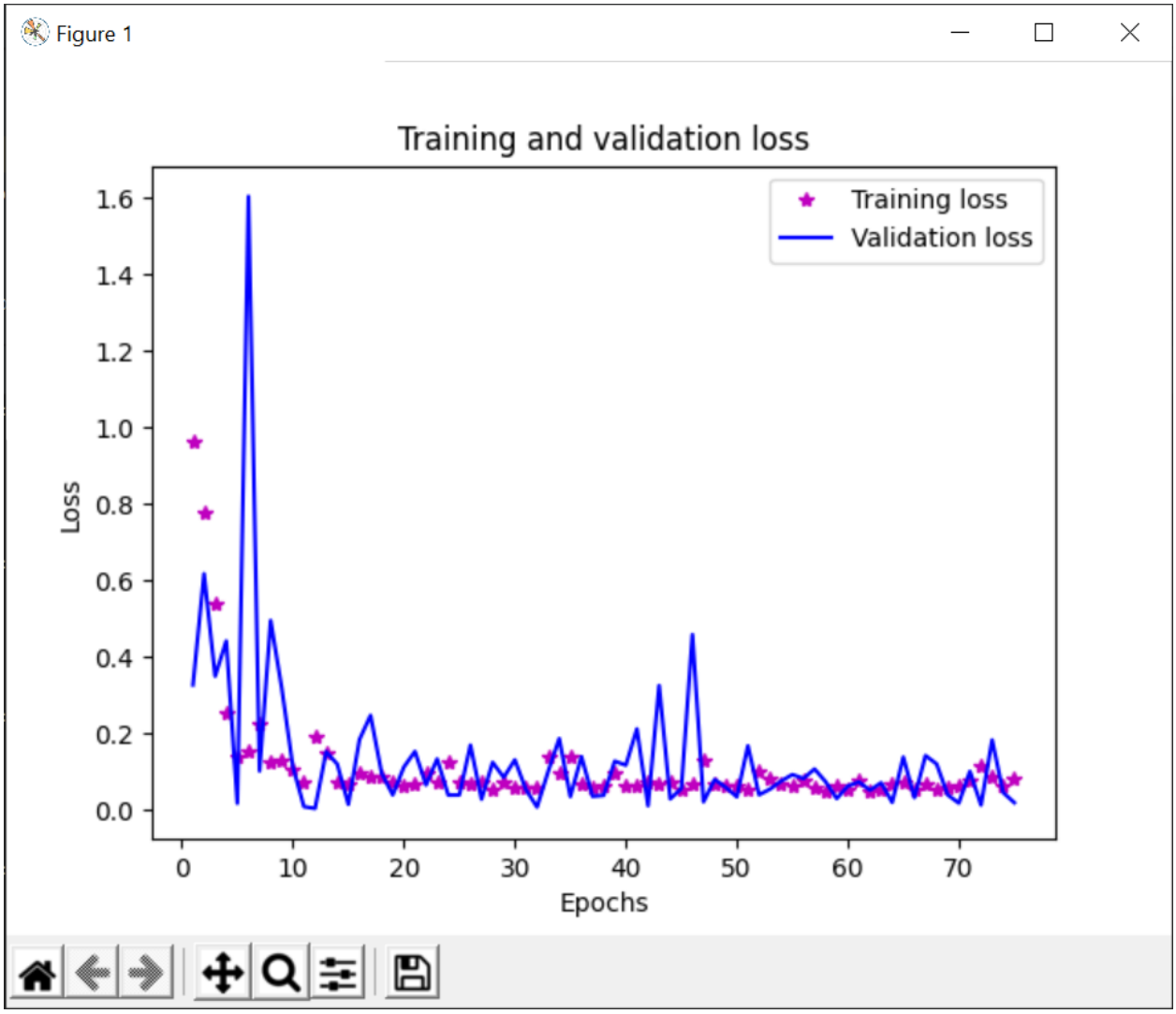

Рисунок 15 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

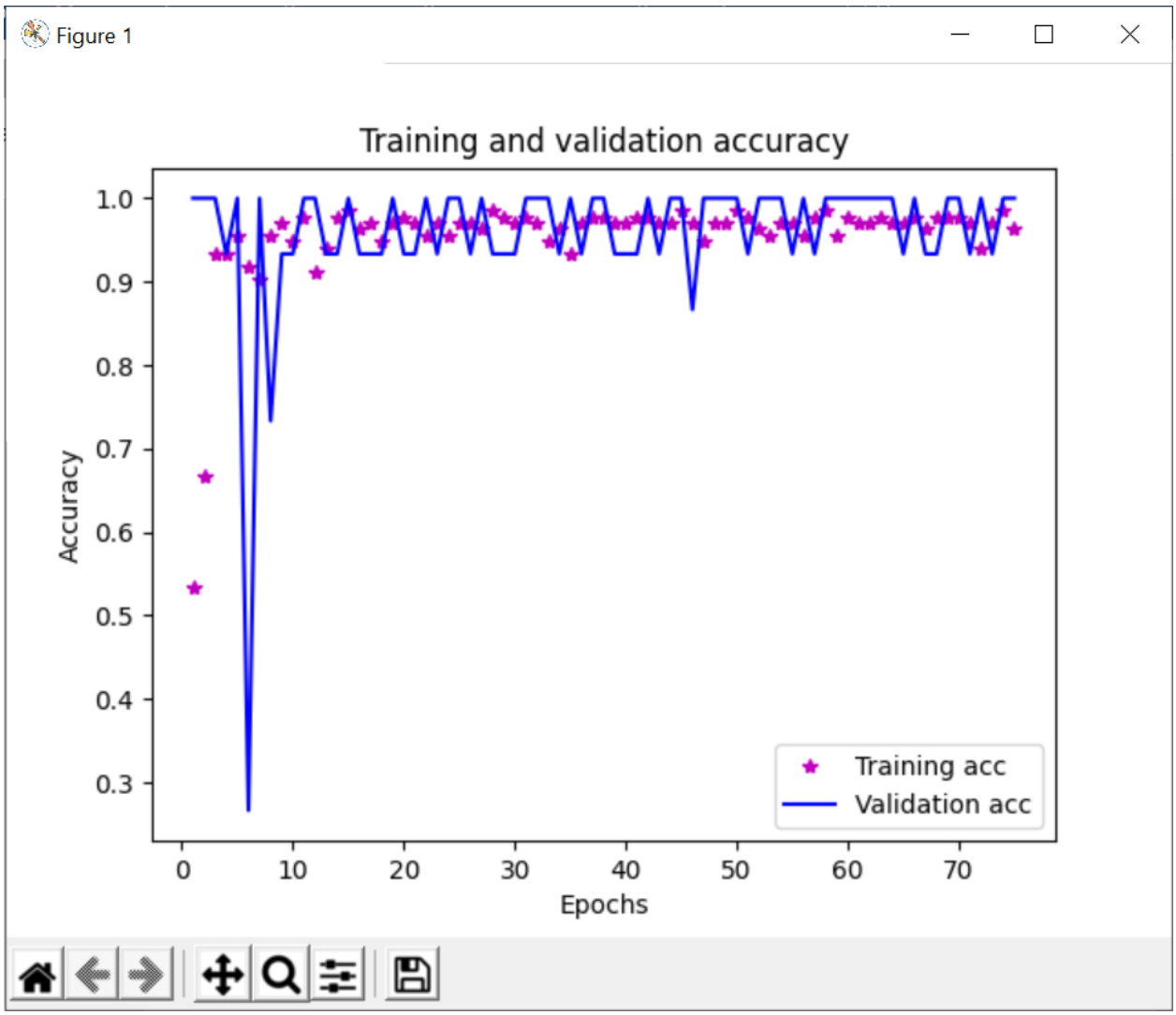

Рисунок 16 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из графиков, видно, что ИНС обучается уже примерно к 11 эпохе. На 11 эпохе её показатели достигают:

loss: 0.1080 - acc: 0.9481 - val_loss: 0.1139 - val_acc: 0.9333

Данные показатели немного хуже, чем в предыдущем тесте, но в целом они схожи. Обучаемость сети немного ухудшилась. Следовательно, можно сделать вывод, что самое оптимальное количество слоев для сети – 3 или 2.

В таком случае было оставлено количество слоев равное 3.

Далее было изучено обучение при различных параметрах обучения (параметры ф-ций `fit`).

Параметр batch_size был изменен на значение 30. Листинг приведен ниже:

```
H = model.fit(X, dummy_y, epochs=75, batch_size=30, validation_split=0.1)
```

Лучший результат тестирования представлен на рис.17-18.

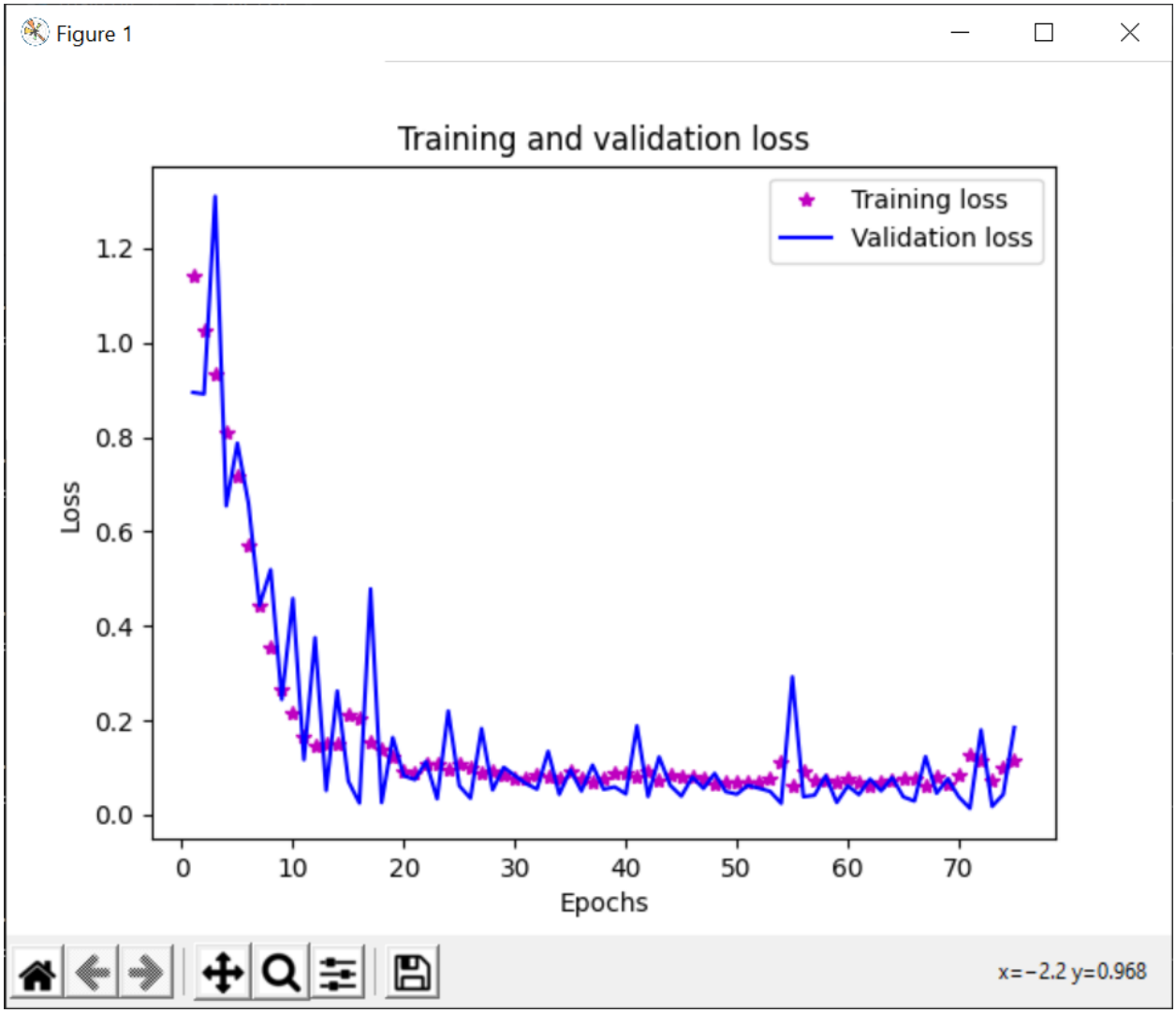

Рисунок 17 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

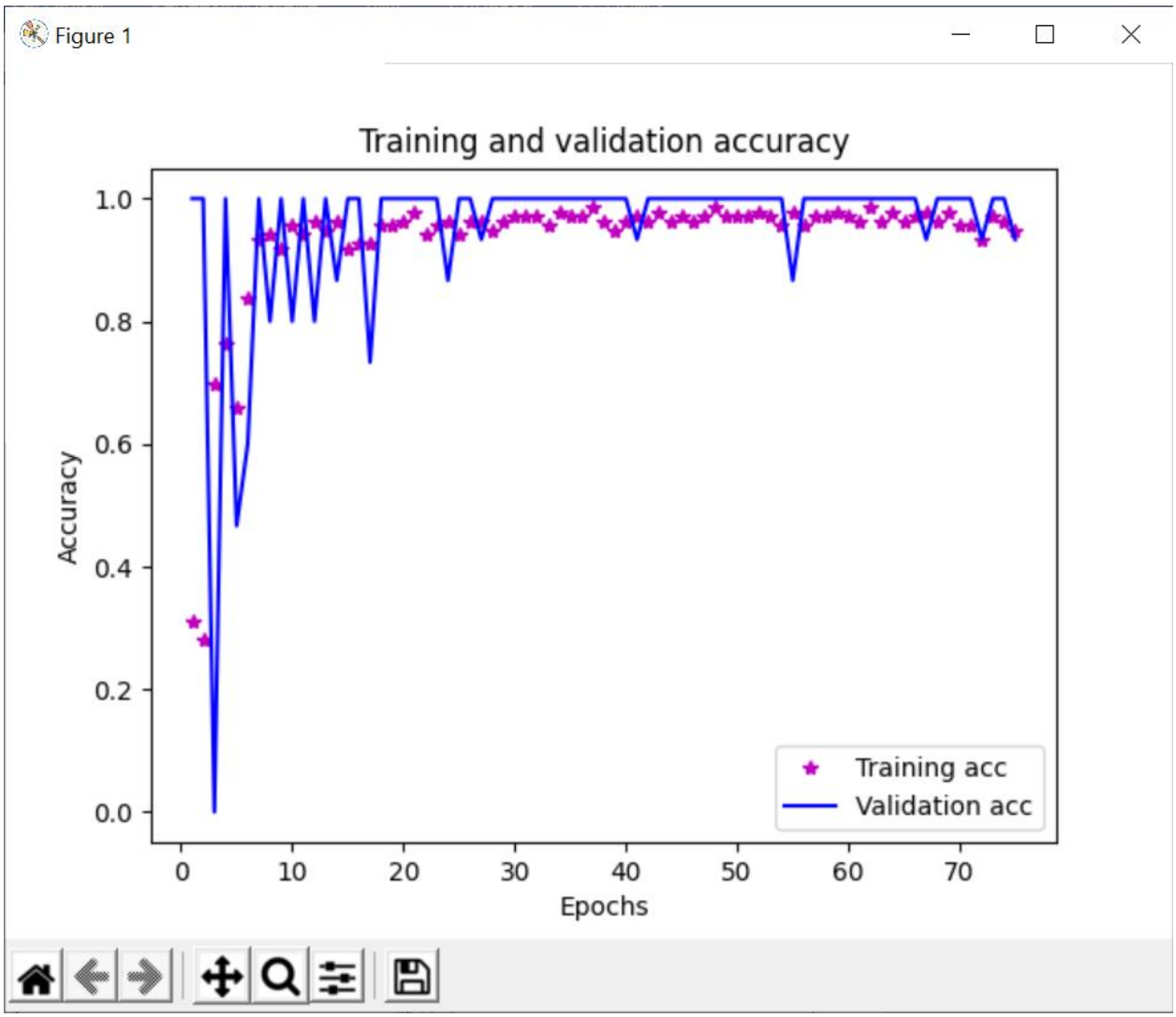

Рисунок 18 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из графиков, видно, что ИНС обучается уже примерно к 19 эпохе. На 19 эпохе её показатели достигают:

loss: 0.1367 - acc: 0.9556 - val_loss: 0.0259 - val_acc: 1.0000

Из результатов можно сделать выводы, что обучаемость ИНС ухудшилась. Следовательно, увеличение параметра batch_size приводит к замедлению обучения ИНС.

Параметр batch_size был изменен на значение 5. Листинг приведен ниже:

```
H = model.fit(X, dummy_y, epochs=75, batch_size=5, validation_split=0.1)
```

Лучший результат тестирования представлен на рис.19-20.

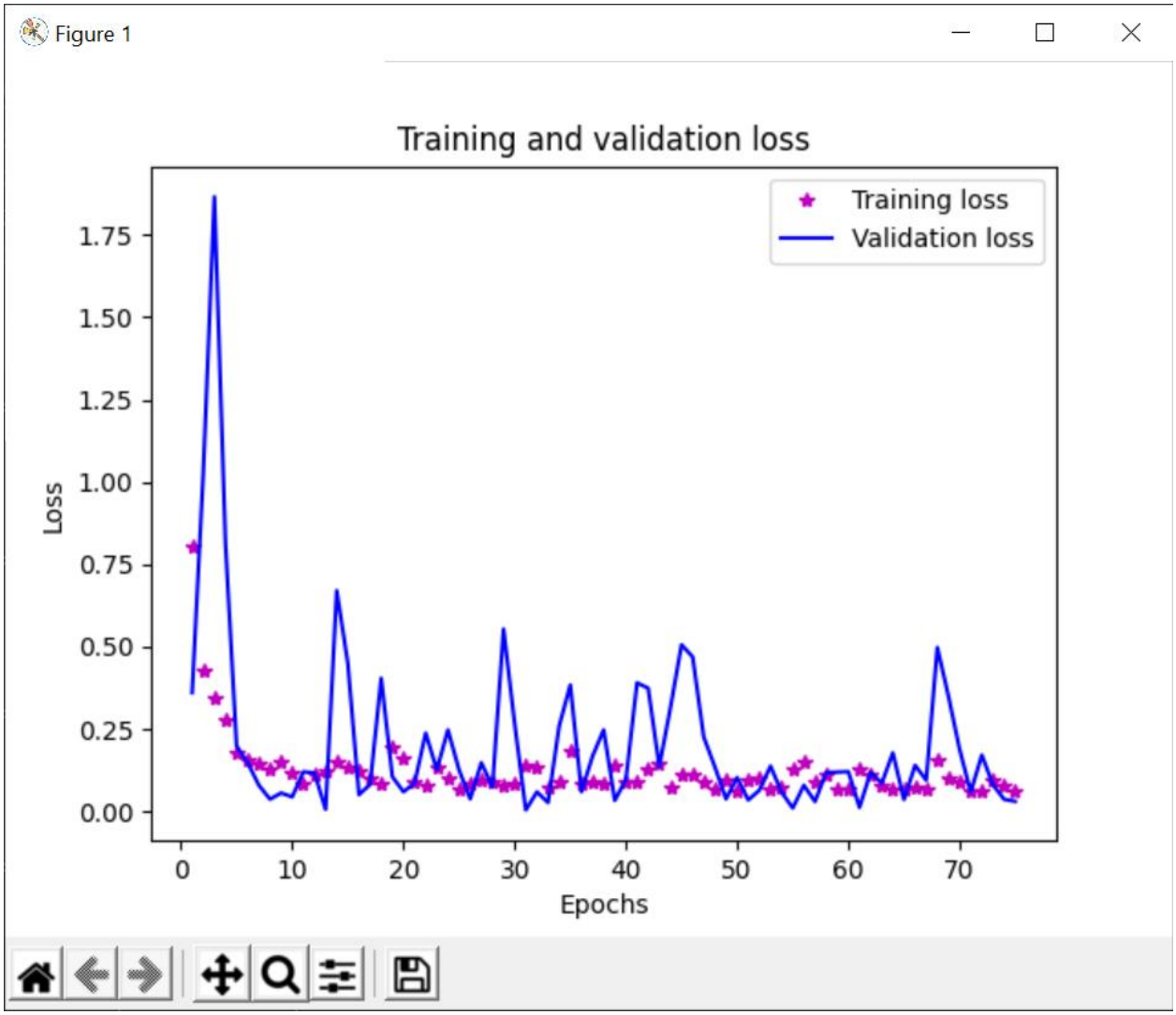

Рисунок 19 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

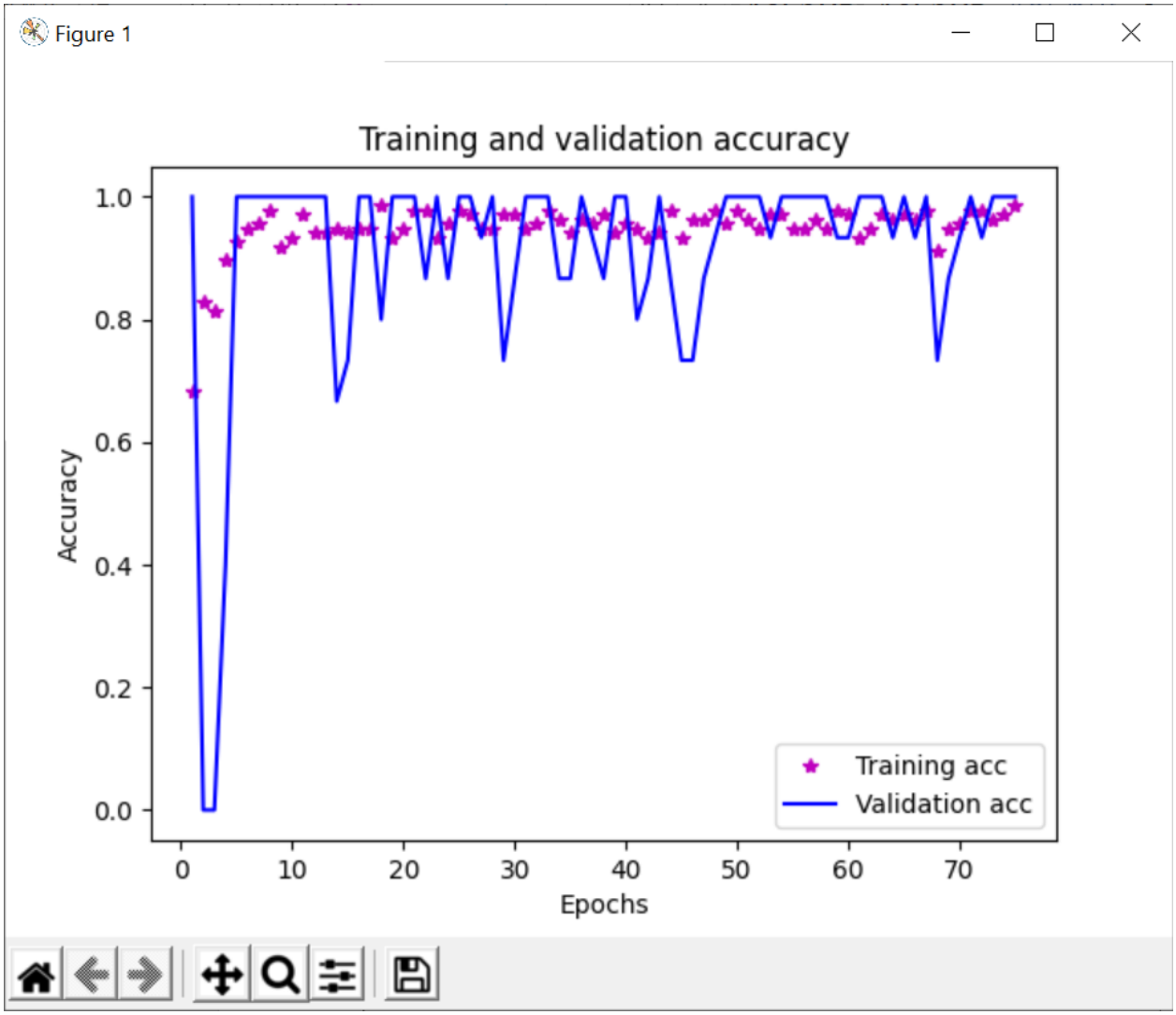

Рисунок 20 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из графиков видно, что обучаемость сети ухудшилось. Это можно объяснить тем, что, когда модель меняет веса через каждые 5 наборов данных, стабильность точности падает. Это может быть связано с тем, что 5 наборов данных недостаточно, чтобы проанализировать, как нужно изменить веса. Следовательно, изначальный вариант параметра batch_size вполне подходил для данной задачи.

Параметр validation_split был изменен на значение 0,4. Листинг приведен ниже:

```
H = model.fit(X, dummy_y, epochs=75, batch_size=10, validation_split=0.4)
```

Лучший результат тестирования представлен на рис.21-22.

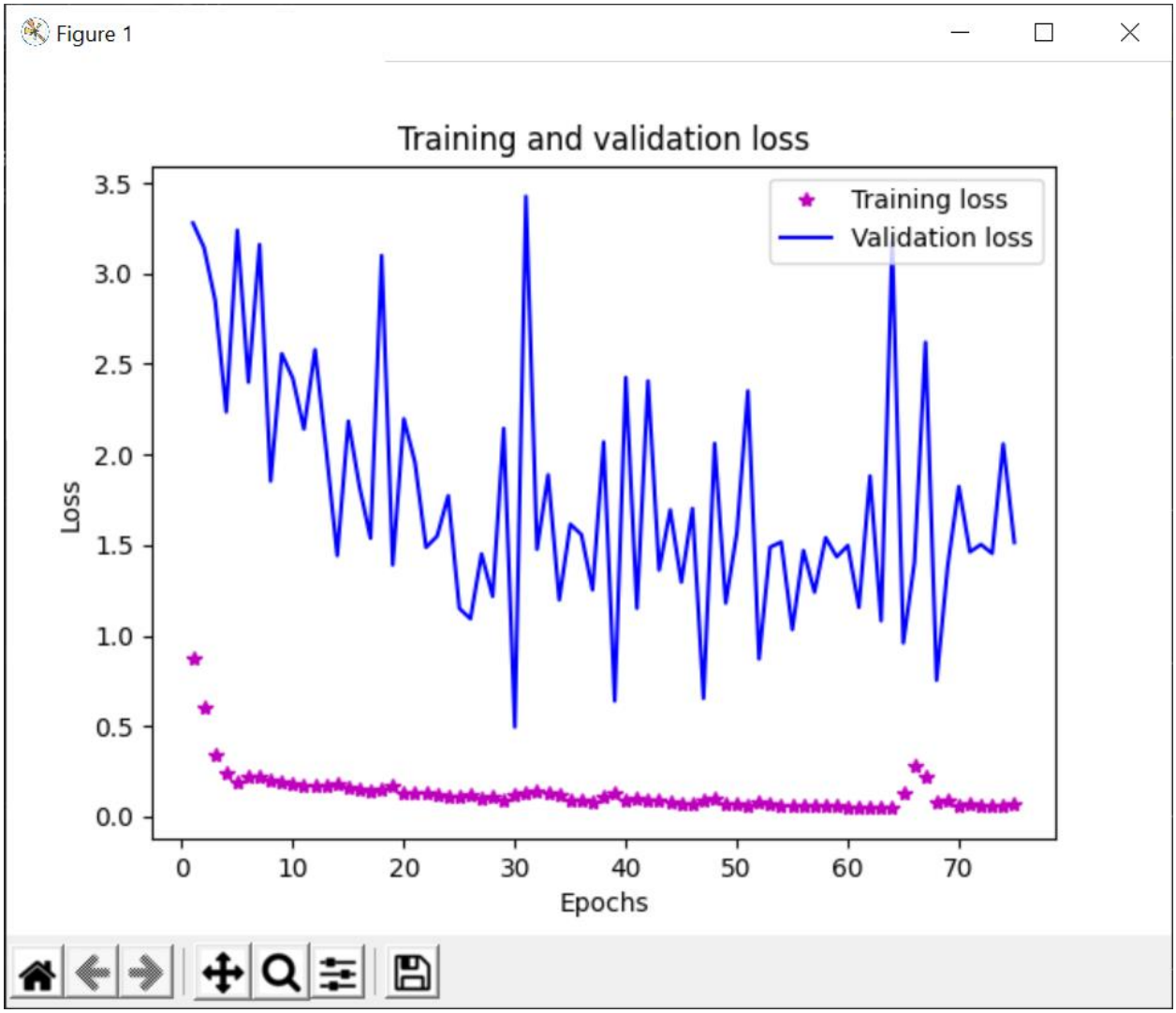

Рисунок 21 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

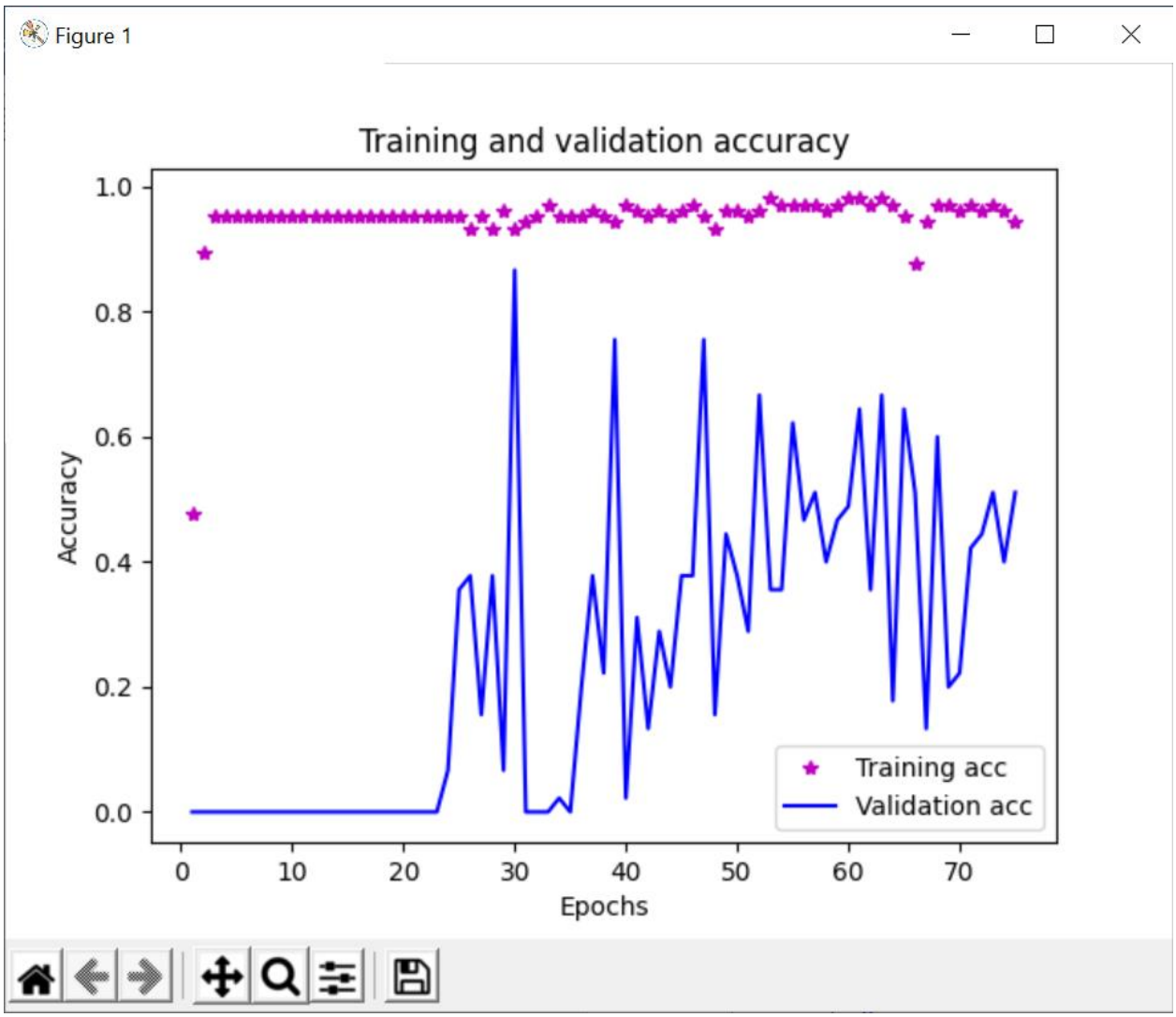

Рисунок 22 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Исходя из графиков, можно сделать вывод, что при validation_split = 0.4, сети не хватает данных для обучения, так как на данных для обучения точность хорошая и потери небольшие, а проверке всё плохо. Поэтому увеличение параметра validation_split ведет к нехватке данных для обучения.

Далее параметр epochs был изменен на значение 50. Листинг приведен ниже:

```
H = model.fit(X, dummy_y, epochs=50, batch_size=10, validation_split=0.1)
```

Лучший результат тестирования представлен на рис.23-24.

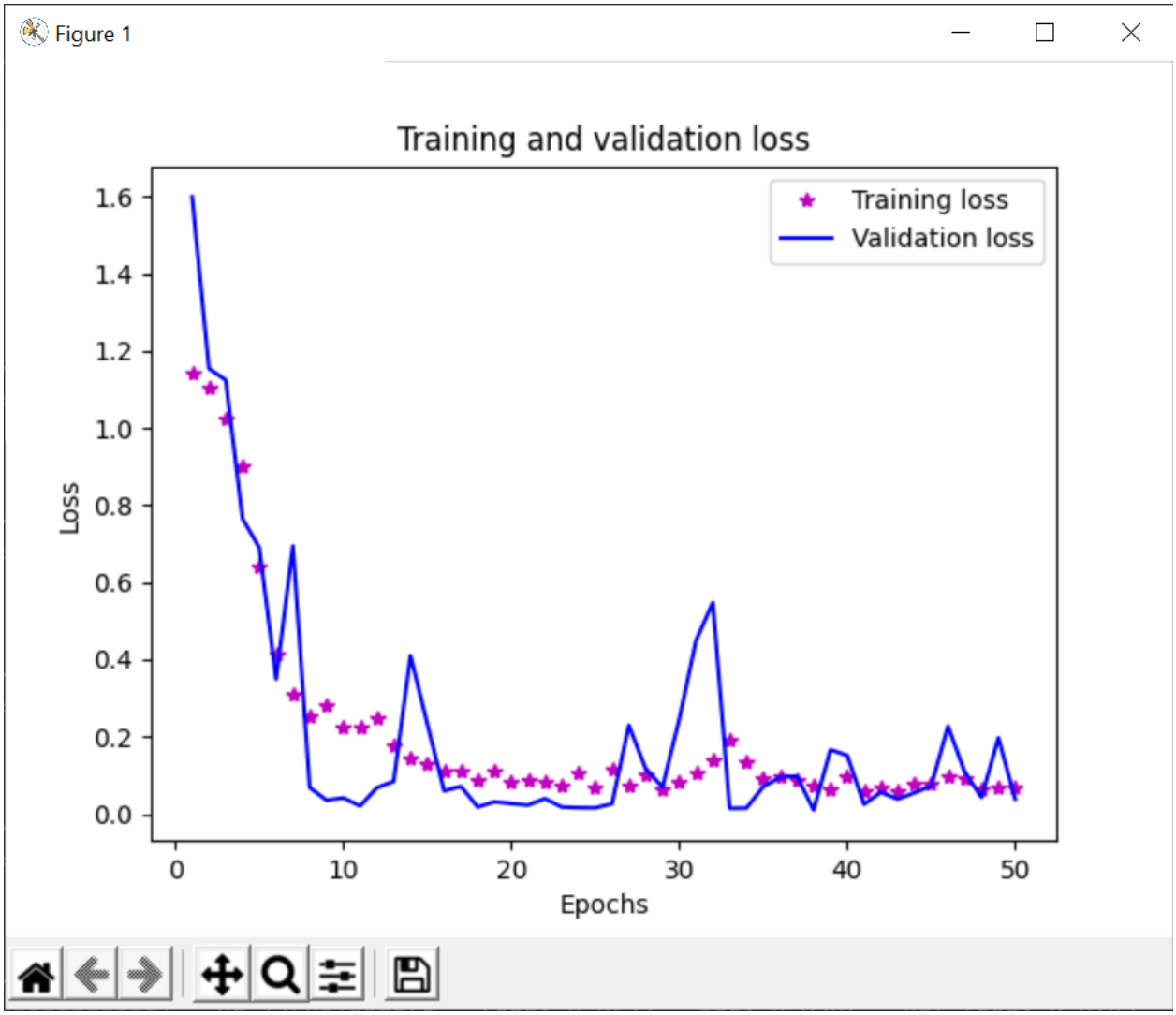

Рисунок 23 – графики потери сети на обучающих данных и данных, не участвовавших в обучении.

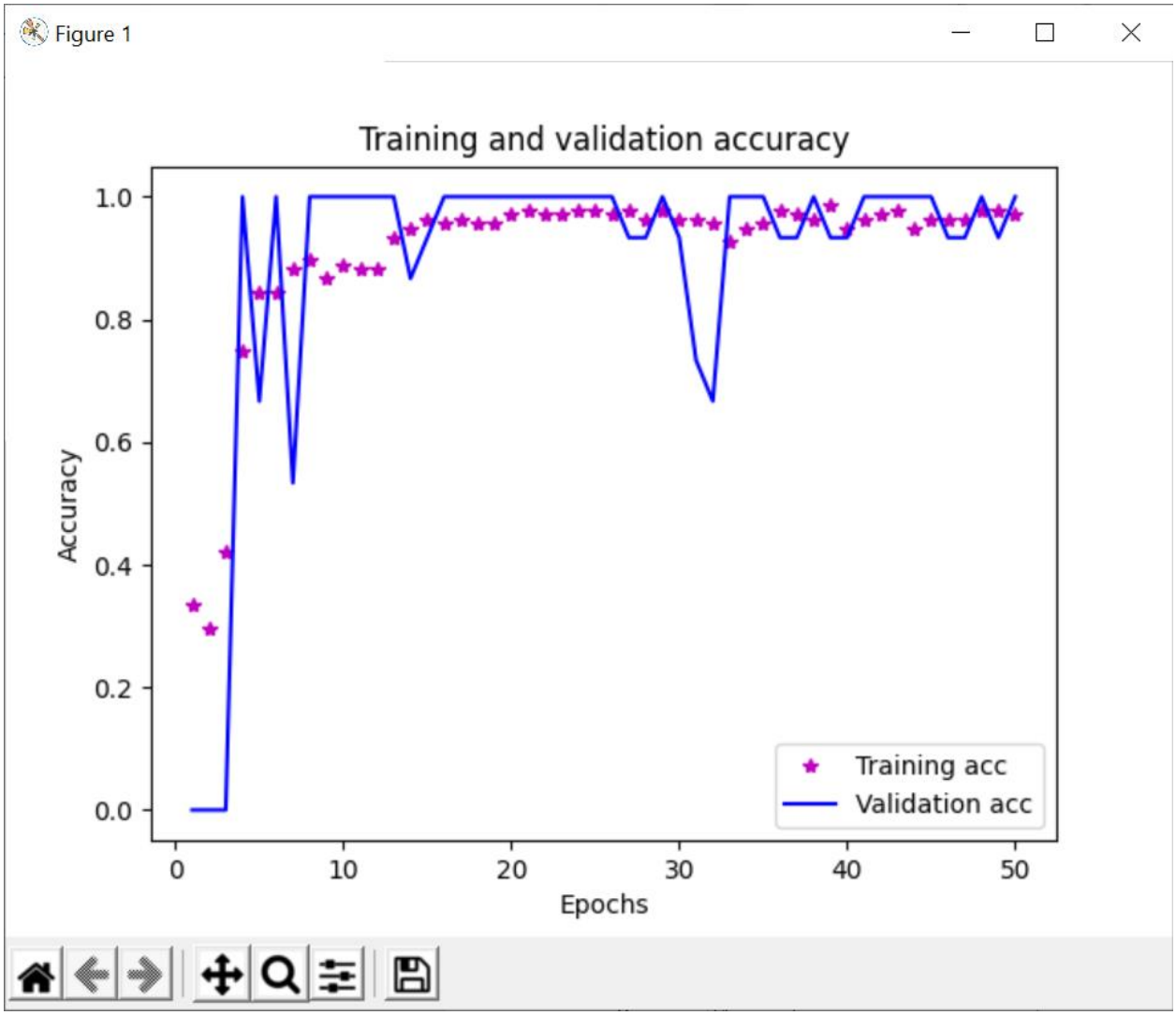

Рисунок 24 – графики точности сети на обучающих данных и данных, не участвовавших в обучении.

Для данной задачи можно выбрать параметр epochs равный 50, так как обучаемость сети достаточно быстрая, а значит epochs = 75 вполне избыточно.

Таким образом, была выбрана лучшая сеть, со следующими слоями и параметрами обучения:

```
model.add(Dense(4, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

epochs = 50

batch_size = 10

validation_split = 0.1

Выводы.

В ходе выполнения лабораторной работы были изучены различные архитектуры ИНС, выполнено обучение при различных параметрах ф-ции fit, а также построены графики ошибок и точности в ходе обучения и выбрана наилучшая модель.