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

Студент гр. 8383 _____ Федоров И.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

Выполнение работы.

Были импортированы необходимые для работы класы, модули и функции. *Pandas* необходим в работе для обработки данных, *Scikit-learn* - используется для подготовки данных для модели.

```
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
```

Набор данных загружается из файла "iris.csv", находящегося в той же директории, что и проект. Затем данные разделются на входные данные *X* (характеристики растения) и выходные *Y* (строка названия - метки).

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

Для подготовки (переход к категориальному вектору) текстовых меток (названий) был выбран метод прямого кодирования (one hot encoding). В данном случае каждому названию будет соответствовать вектор с нулевыми элементами и 1 по индексу равному индексу метки (например названию "Iris-

setosa" будет соответствовать вектор [1,0,0]). Метод является частью Keras (функция `to_categorial`).

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

Т.к. будет проверено несколько вариантов ИНС, для создания модели была реализована функция `create_model()`, принимающая на вход количество слоев и нейронов в них (кроме последнего, модель завершается слоем Dense с размером 3 = числу категорий). Полный текст программы приведен в приложении А.

Для построения графиков потерь и точности при обучении и тестировании была реализована функция `plot_model_loss_and_acc()`, которая принимает на вход объект `history`, который возвращает метод обучения модели `fit()`. При построении используется библиотека `matplotlib`.

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

При компиляции модели используется функция потерь `categorical_crossentropy`, которая определяет расстояние между распределениями вероятностей.

Для чистоты сравнения разных ИНС, данные были разделены на обучающие и тестовые в соотношении 70:30. В данном случае, исходный набор данных представляет собой 3 "поднабора" идущих подряд, каждый из который относится к одному цветку.

```
test_size = 15
part_x_train = np.concatenate((X[0:50-test_size], X[50:100-
test_size], X[100:150-test_size]))
part_y_train = np.concatenate((dummy_y[0:50-test_size],
dummy_y[50:100-test_size], dummy_y[100:150-test_size]))
x_test = np.concatenate((X[50-test_size:50], X[100-
test_size:100], X[150-test_size:150]))
y_test = np.concatenate((dummy_y[50-test_size:50], dummy_y[100-
test_size:100], dummy_y[150-test_size:150]))
```

В работе также использовалась встроенная в Keras визуализация моделей:

```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```

С помощью метода `fit()` осуществляется обучение модели:

```
history = model.fit(part_x_train, part_y_train,
                                epochs=75,
                                batch_size=5,
                                validation_data=(x_test, y_test),
                                verbose=0)
```

На вход подаются обучающие данные, проверочные данные, а также число эпох. Вызов метода возвращает объект `History`, который имеет поле `history`, вляющийся словарем с данными о процессе обучения. Он и используется при выводе графиков.

Первая модель имеет вид, представленный на рис. 1.

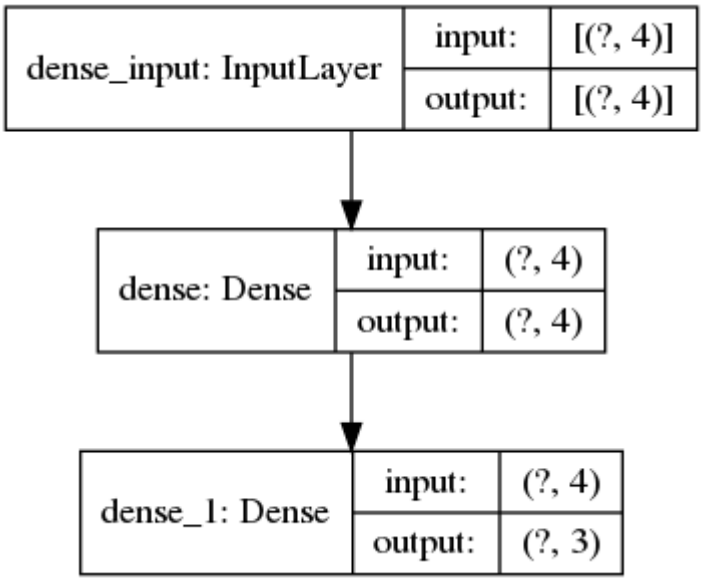

Рисунок 1 - Схема модели

В данном случае "?" в схеме означает размер одного пакета данныех - batch (т.к. стоит символ "?", то он может быть любым). Графики потерь и точности при обучении и тестировании показаны на рис. 2.

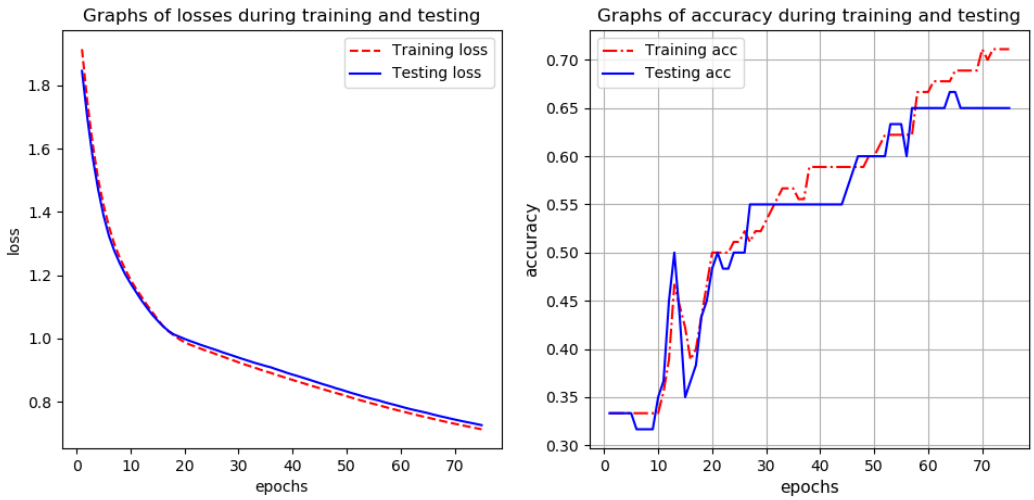

Рисунок 2 - Графики потерь и точности

Данная модель является не самой удачной, приемлимая точность достигается за достаточно большое количество эпох, также модель достаточно "нестабильная", т.к. при перезапуске результаты могут отличаться достаточно существенно (вероятно из-за разных начальных весов). Попробуем увеличить количество нейронов на слое до 8, оставив такое же количество слоев. Графики потерь и точности показаны на рис. 3.

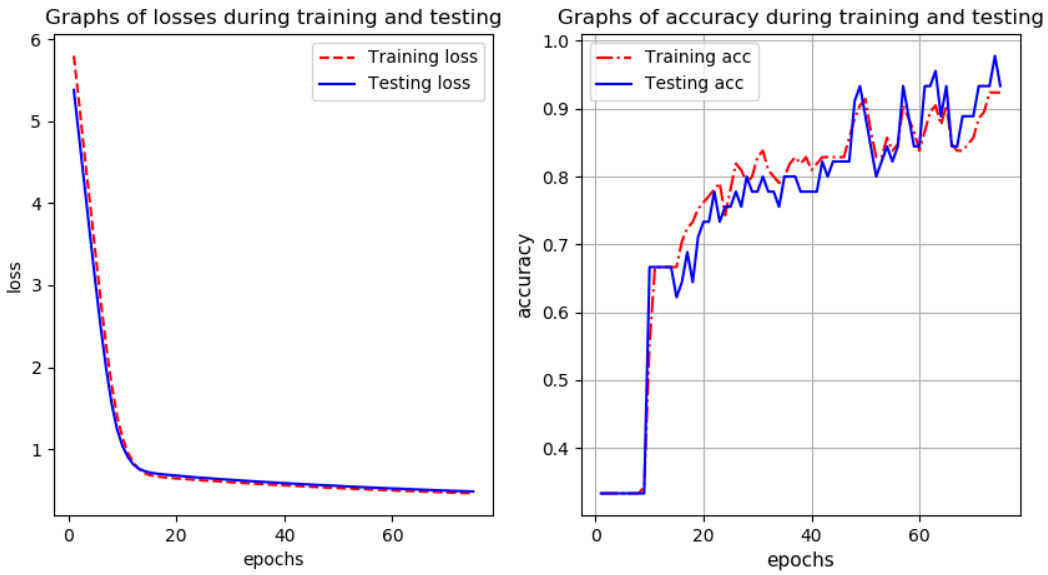

Рисунок 3

Можно заметить, что точность модели несколько повысилась, хотя для достижения высокого значения требуется прохождение большого количества эпох.

Улучшим данную модель, увеличив количество слоев и нейронов на них. Схема следующей модели показана на рис. 3.

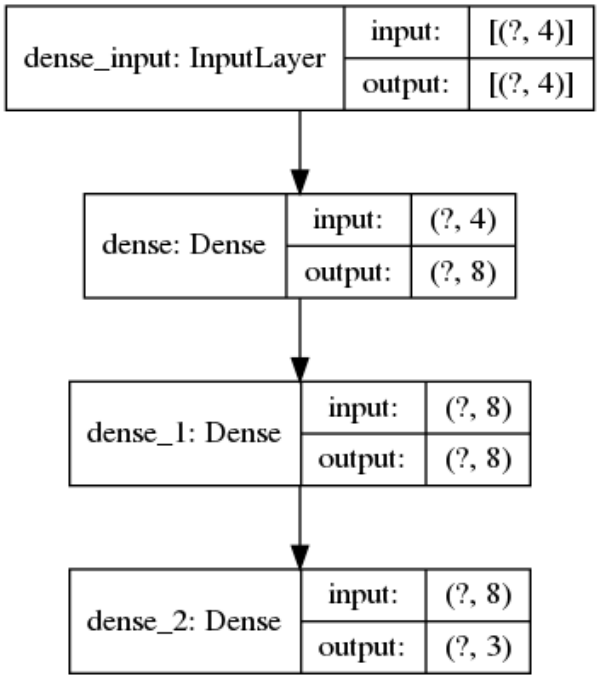

Рисунок 4 - Схема модели

Графики потерь и точности при обучении и тестировании показаны на рис. 5.

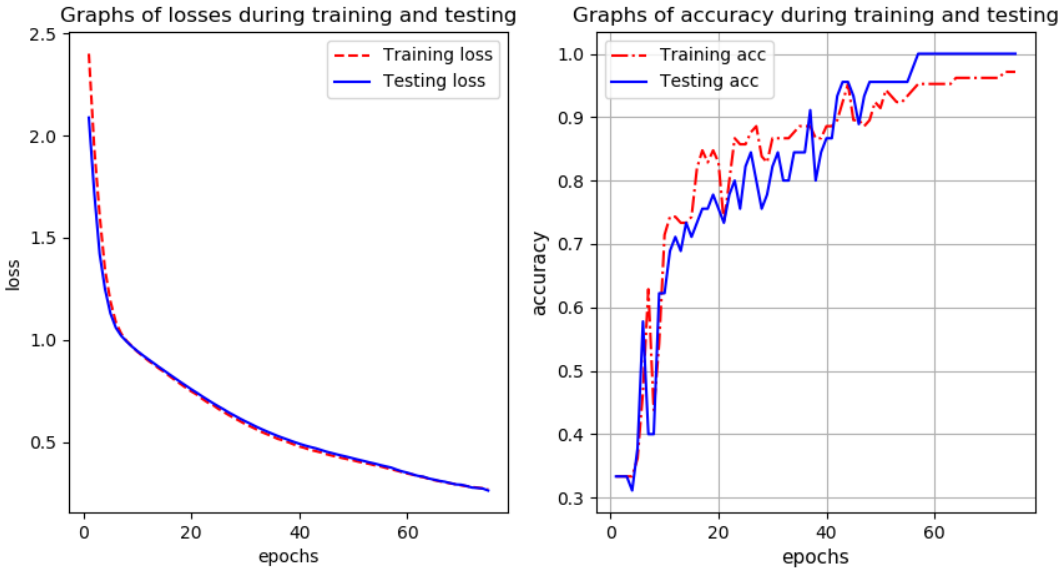

Рисунок 4 - Графики потерь и точности

Можно заметить, что модель стала лучше. Точность данной модели повысилась, а также уменьшилось число эпох, при которых достигается хорошая точность.

В качестве эксперимента увеличим число нейронов в слоях еще раз. Схема новой модели показана на рис. 6.

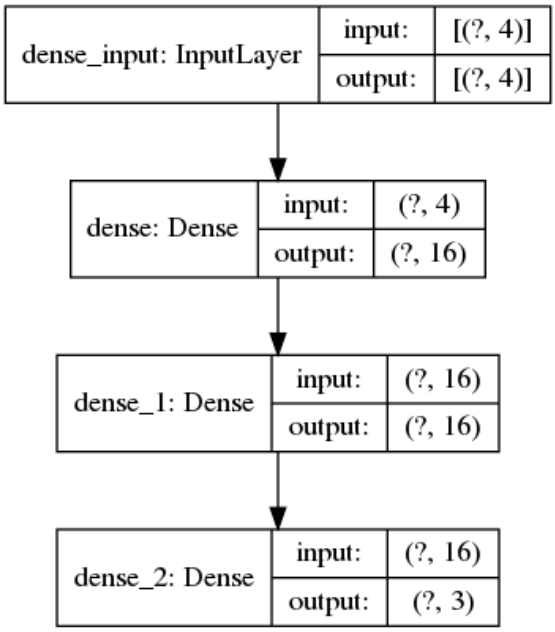

Рисунок 6 - Схема модели

Графики потерь и точности при обучении и тестировании показаны на рис. 7.

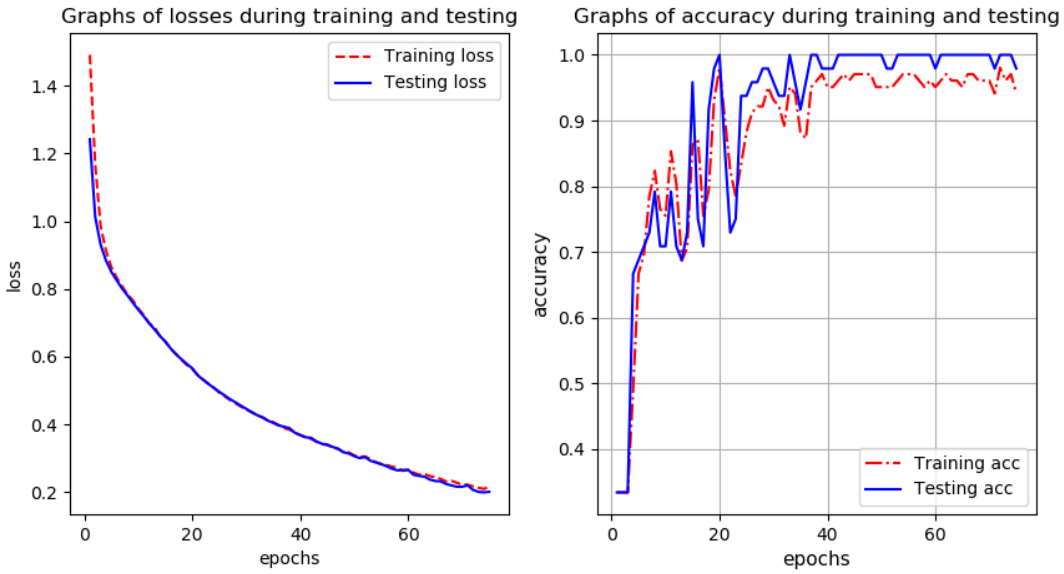

Рисунок 7 - Графики потерь и точности

Увеличение количества нейронов в слоях привело к увеличению скорости обучения, по сравнению с предыдущей моделью. Попробуем увеличить количество нейронов еще больше. Схема новой модели приведена на рис. 8.

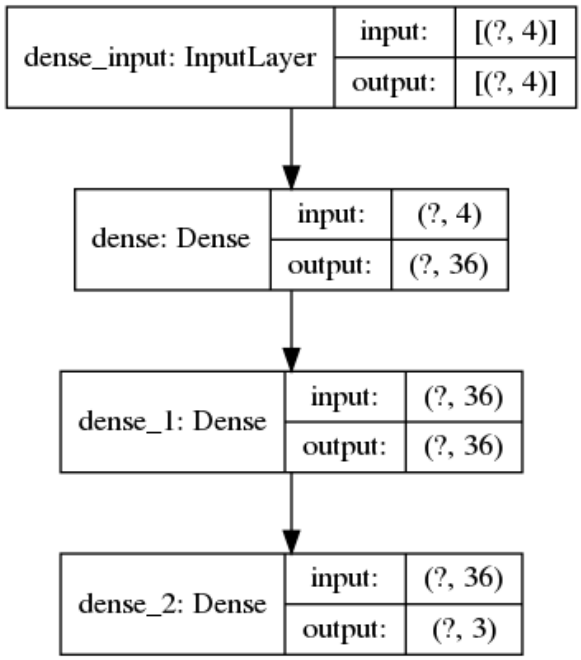

Рисунок 8 - Схема модели

Графики точности и потерь показаны на рис. 9.

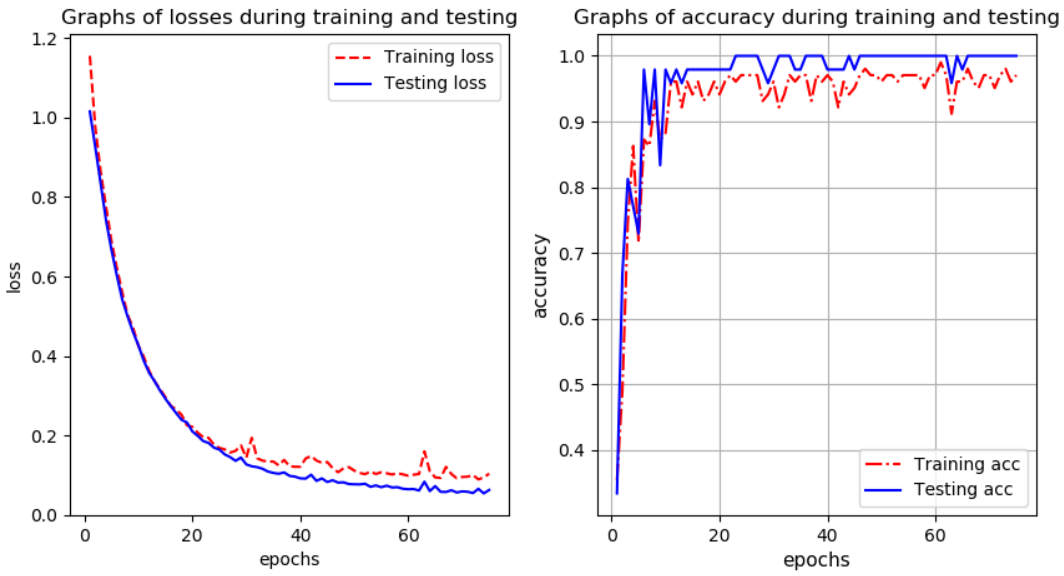

Рисунок 9 - Графики точности и потерь

Видно, что скорость обучения вновь возрасла, уже на 8 эпохе модель достигает высокой точности. Дальнейшее увеличение числа нейронов не привело к значительным изменениям, значит стоит остановиться на данном варианте.

Модель, показанная на рис. 8 показала оптимальный результат.

Проведем небольшой эксперимент: изменим параметр `validation_data` метода обучения `fit()` так, что соотношение данных для обучения и тестирование будет находится в соотношении 20:80. При этом применим оптимальную модель, показанную на рис. 8. График потерь и точности показан на рис. 10.

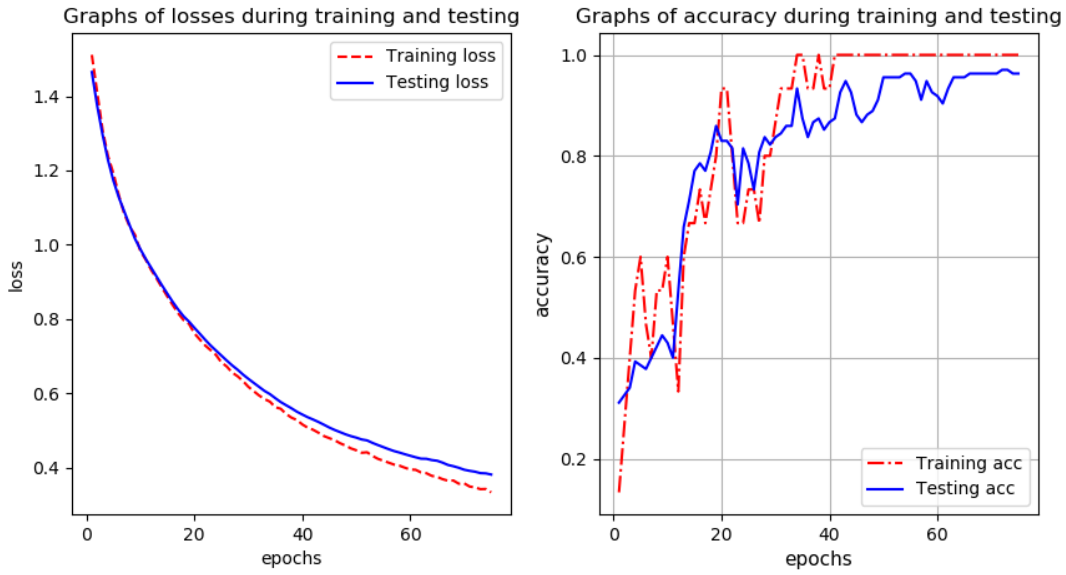

Рисунок 10 - График потерь и точности

Видно, что модель ухудшилась, т.к. точность понизилась. Также увеличилось число эпох, при котором модель достигает высокой точности.

Выводы.

В ходе выполнения работы была реализована искусственная нейронная сеть для классификации сортов растения ирис. Были получены базовые понятия о структуре ИНС, способы преобразования данных в удобный вид для обучения сети. Были построены необходимые графики и схемы.

Приложение А

```
# импорт модулей
import pandas
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import plot_model

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

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


# Создание модели сети
def create_model(num_layers=2, num_neurons=4):
    model = Sequential()
    model.add(Dense(num_neurons, activation='relu',  input_shape=(4,)))
    for i in range(num_layers-2):
        model.add(Dense(num_neurons, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    return model


# графики потерь и точности при обучении и тестирования
def plot_model_loss_and_accuracy(history, figsize_=(10,5)):
    plt.figure(figsize=figsize_)
    train_loss = history.history['loss']
    test_loss = history.history['val_loss']
    train_acc = history.history['acc']
    test_acc = history.history['val_acc']
    epochs = range(1, len(train_loss) + 1)

    plt.subplot(121)
    plt.plot(epochs, train_loss, 'r--', label='Training loss')
    plt.plot(epochs, test_loss, 'b-', label='Testing loss')
    plt.title('Graphs of losses during training and testing')
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.legend()
    plt.subplot(122)
    plt.plot(epochs, train_acc, 'r-.', label='Training acc')
    plt.plot(epochs, test_acc, 'b-', label='Testing acc')
    plt.title('Graphs of accuracy during training and testing')
    plt.xlabel('epochs', fontsize=11, color='black')
    plt.ylabel('accuracy', fontsize=11, color='black')
    plt.legend()
    plt.grid(True)

    plt.show()


model =  create_model(num_layers = 3, num_neurons = 36)
```

```
# инициализация параметров обучения (оптимизатор, функцию потерь,  метрика
мониторинга (точность))
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])


# разделение данных на обучающие и тестовые в соотношении 80 : 20
#part_x_train, x_test, part_y_train, y_test = train_test_split(X, dummy_y,
test_size=.20, random_state=42)
test_size = 16
part_x_train = np.concatenate((X[0:50-test_size], X[50:100-test_size],
X[100:150-test_size]))
part_y_train = np.concatenate((dummy_y[0:50-test_size], dummy_y[50:100-
test_size], dummy_y[100:150-test_size]))
x_test = np.concatenate((X[50-test_size:50], X[100-test_size:100], X[150-
test_size:150]))
y_test = np.concatenate((dummy_y[50-test_size:50], dummy_y[100-
test_size:100], dummy_y[150-test_size:150]))



# обучения сети
history = model.fit(part_x_train, part_y_train,
                                epochs=75,
                                batch_size=5,
                                validation_data=(x_test, y_test),
                                verbose=0)

plot_model_loss_and_accuracy(history)
plot_model(model, to_file='model.png', show_shapes=True)
```