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

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине «Искусственные нейронные сети»

Тема: «Бинарная классификация отраженных сигналов радара»

Студент гр. 7381 _____ Дорох С.В.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург

2020

Цели.

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.

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

Выполнение работы.

1) Была создана и обучена модель искусственной нейронной сети в соответствии с условиями(весь код представлен в приложении А).

2) При исследовании разных архитектур и обучение при различных параметрах обучения ИНС необходимо было:

- уменьшить размер входного слоя в два раза
- добавить скрытый слой в архитектуру сети с 15 нейронами

В итоге получаем 4 модели ИНС, которые нужно сравнить. Ниже на рис. 1-8 представлены графики точности и ошибок моделей в ходе обучения.

Графики аналогичны графикам из предыдущей лабораторной работы.

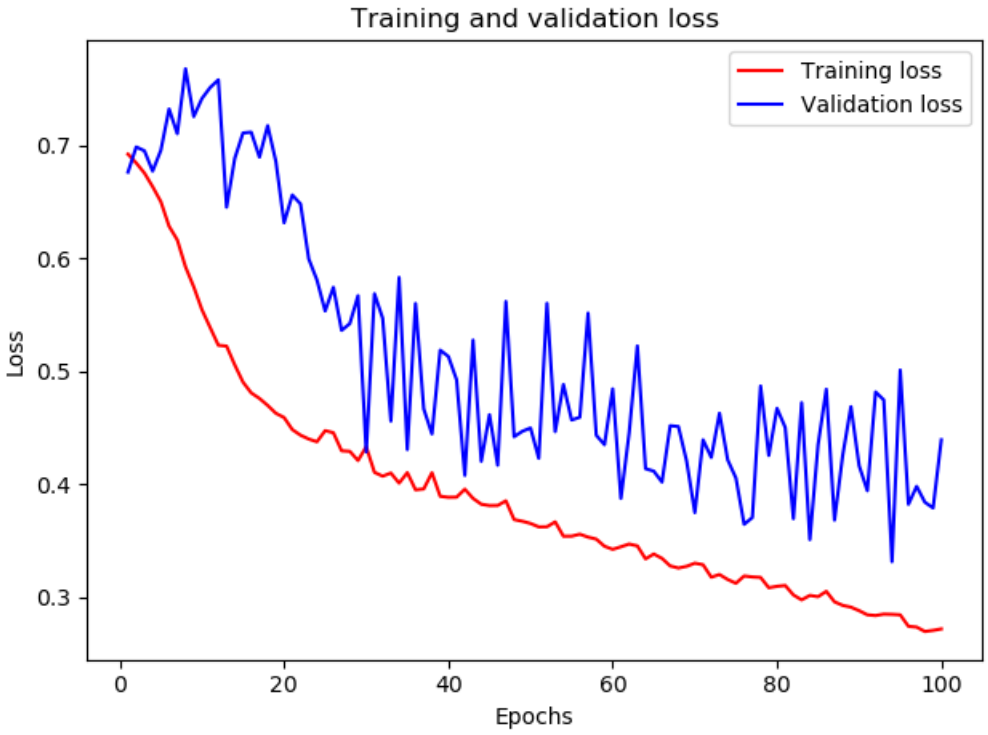

рисунок 1 – график ошибок изначальной модели ИНС

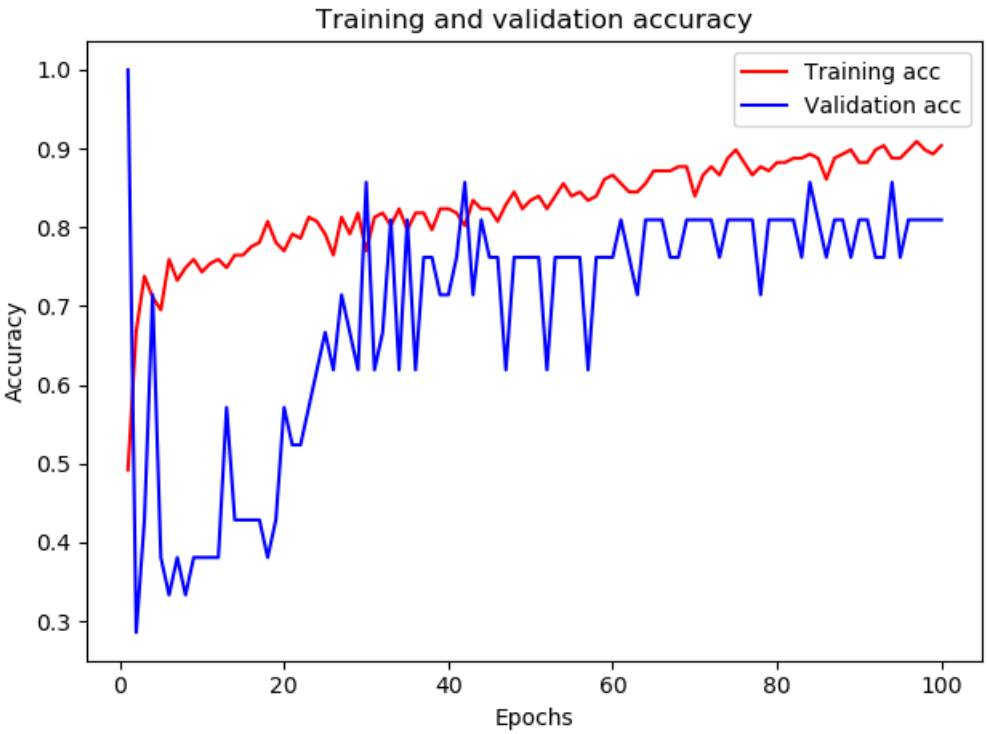

рисунок 2 – график точности изначальной модели ИНС

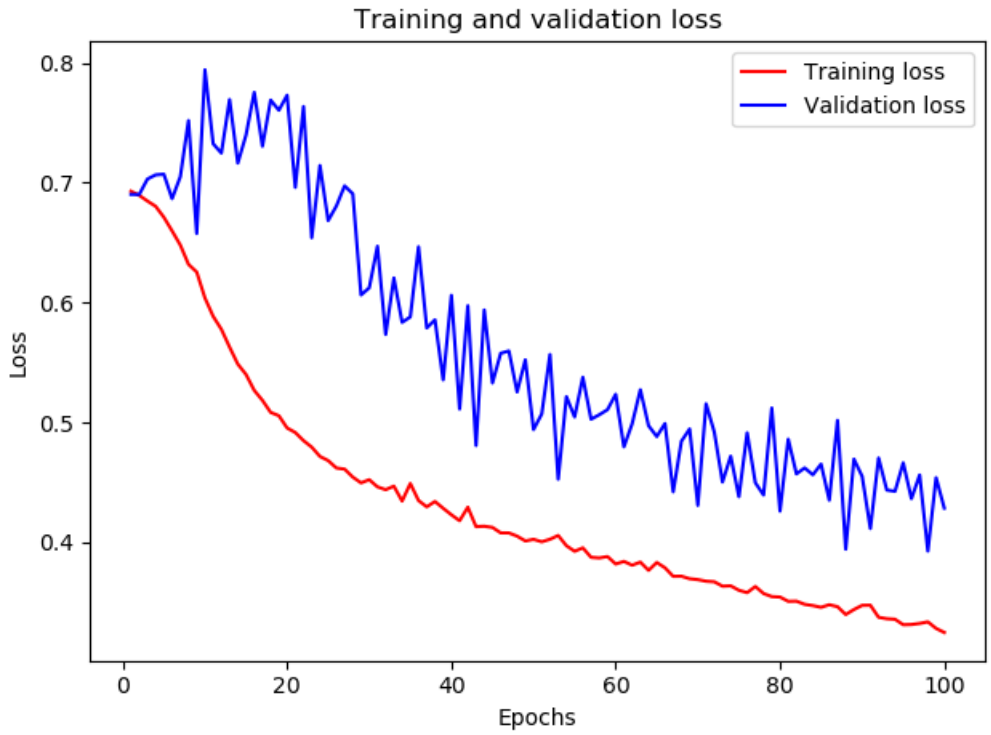

рисунок 3 – график ошибок модели ИНС с уменьшенным количеством нейронов в входном слое

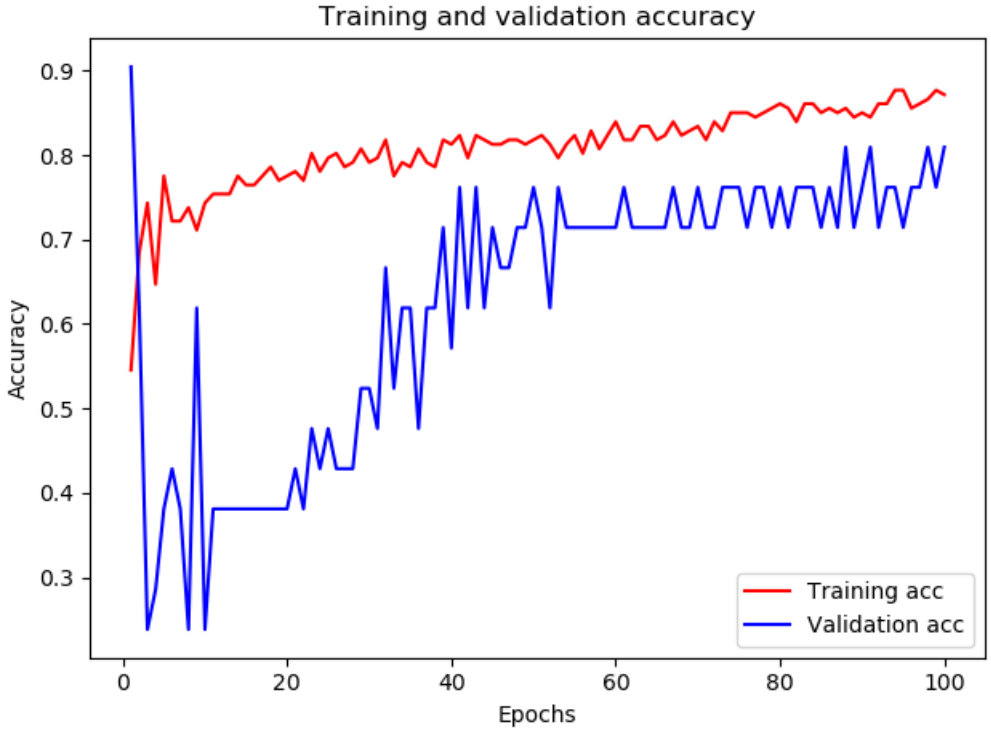

рисунок 4 – график точности модели ИНС с уменьшенным количеством нейронов в входном слое

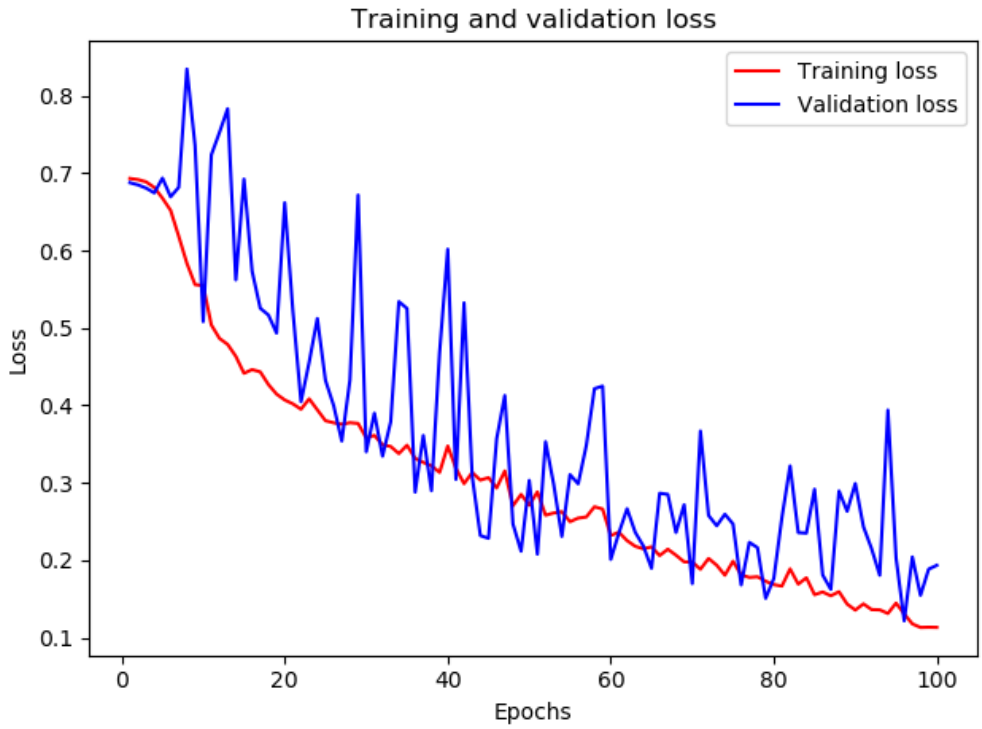

рисунок 5 – график ошибок модели ИНС со скрытым слоем

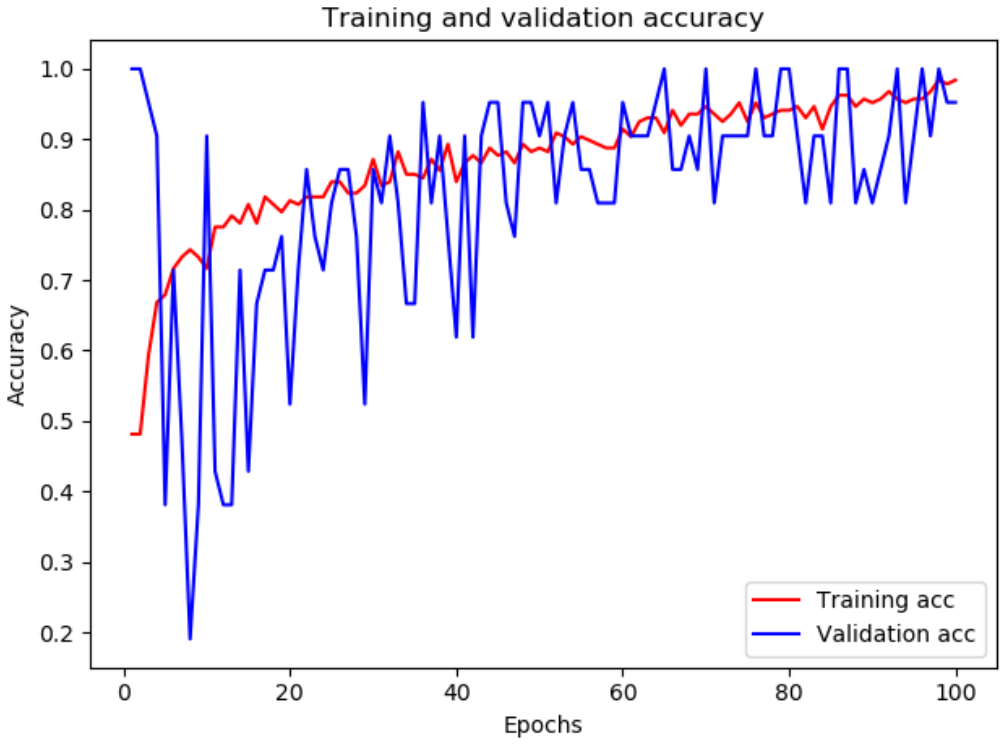

рисунок 6 – график точности модели ИНС со скрытым слоем

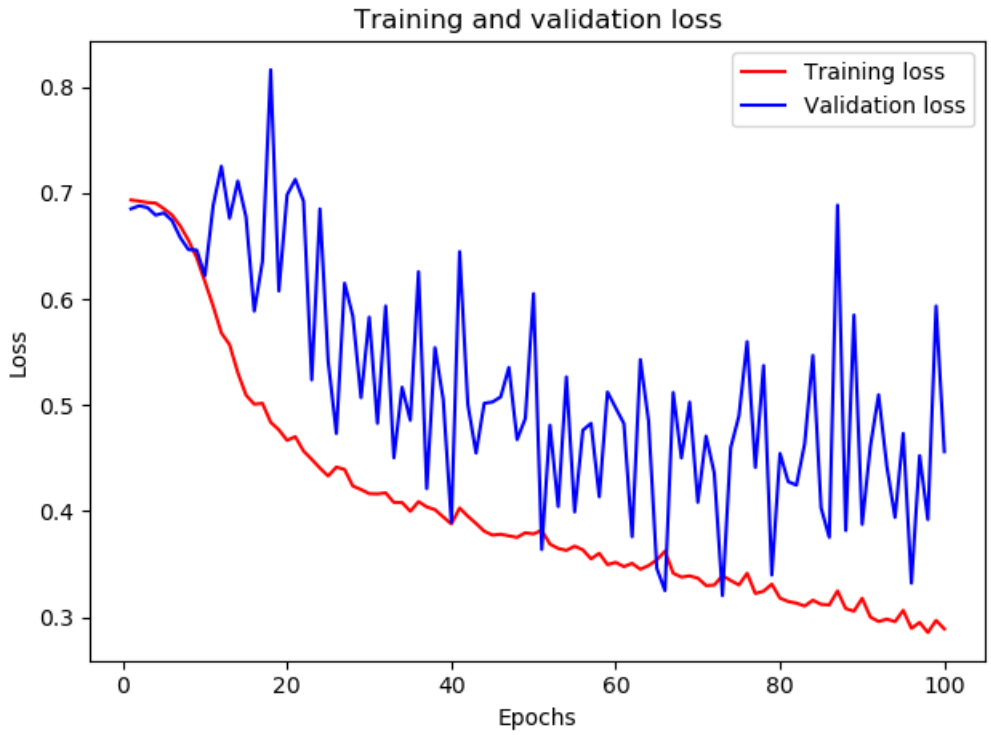

рисунок 7 – график ошибок модели ИНС с уменьшенным количеством нейронов во входном слое и скрытым слоем

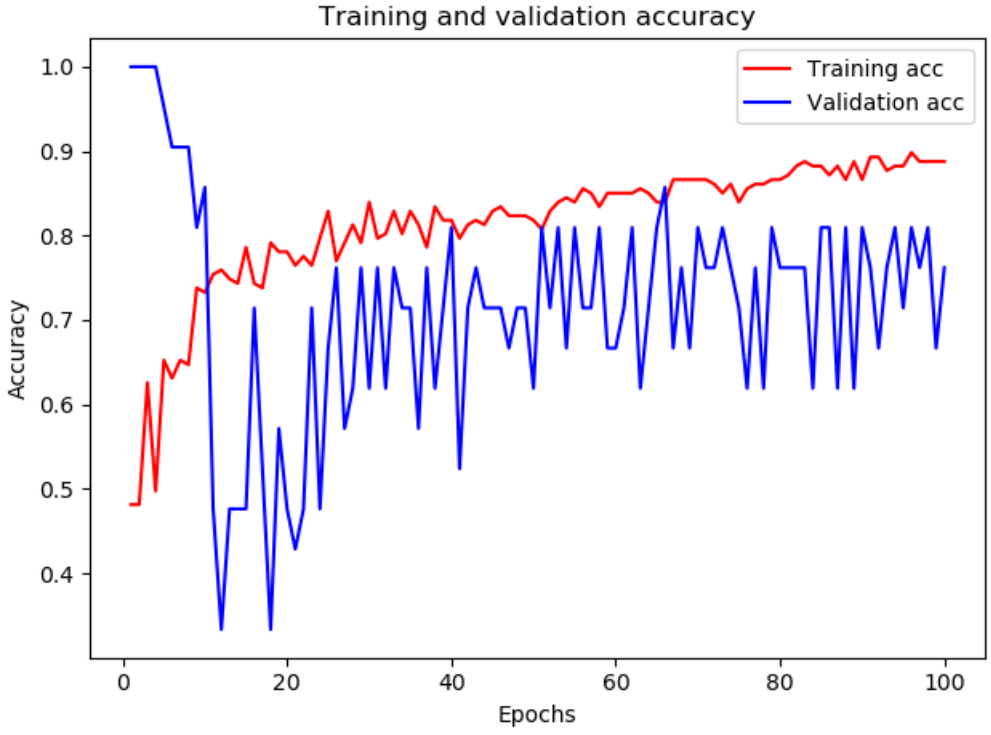

рисунок 7 – график точности модели ИНС с уменьшенным количеством нейронов во входном слое и скрытым слоем

3) При попытке найти модель с наименьшей потерей на тестировочных данных наилучший результат показала модель с 1 промежуточным слоем с 60 нейронами. Результат составил val_loss: 0.0239 – val_acc: 1.0000.

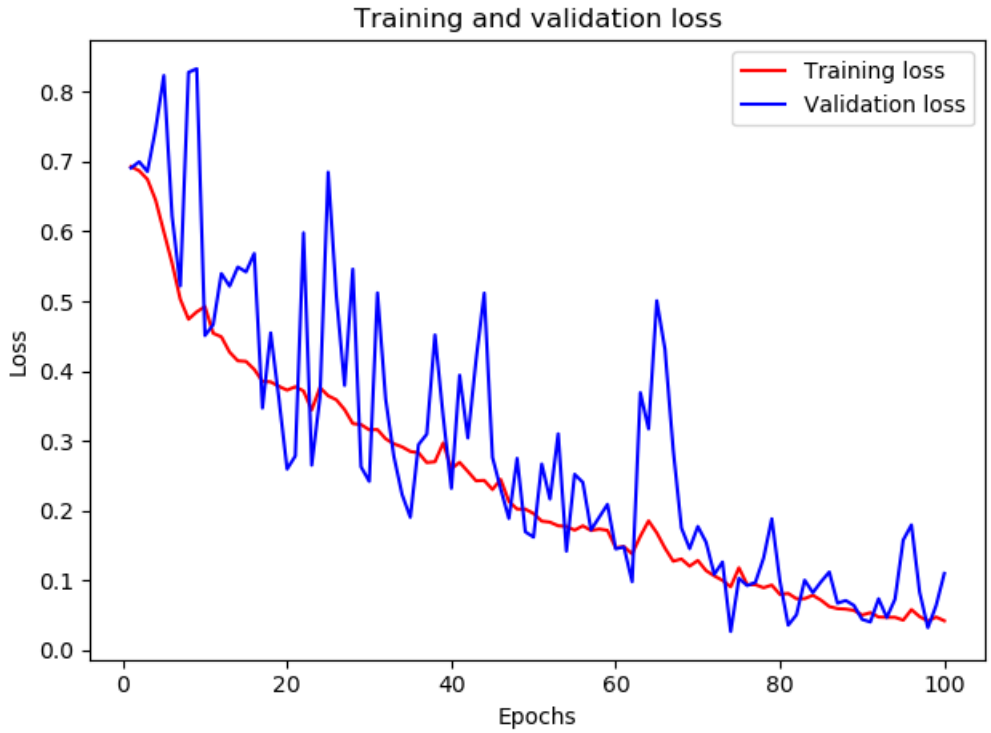

Рисунок 8 – график потерь модели ИНС с наилучшими результатами

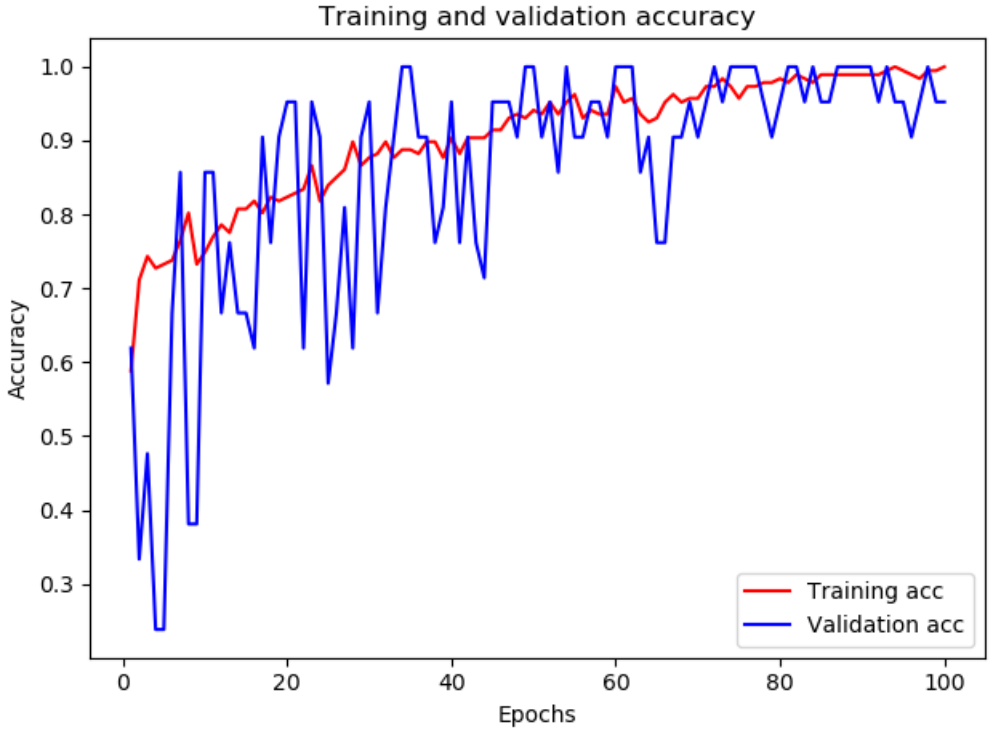

Рисунок 9 – график точности модели ИНС с наилучшими результатами

Анализируя полученные графики, можно увидеть, что с увеличением скрытых слоёв (до 2-х) увеличивается точность и уменьшаются потери. С уменьшением количества входных нейронов или нейронов на скрытых слоях точность уменьшается, а потери увеличиваются, это связано с недостаточной интерпретацией свойств входных данных для обучения модели.

Вывод.

В ходе выполнения данной работы были получены навыки в бинарной классификации в библиотеке keras, была изучена зависимость количества входных слоёв и нейронов от результирующей точности и потери обучаемой модели ИНС.

Приложения

Приложение А

```
import sys
sys.environ = ['TF_CPP_MIN_LOG_LEVEL']
import matplotlib.pyplot as plt
import pandas
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model

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

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

best_model = 0
best_val_loss = 1
best_neurons_num = 0
best_dense_num = 0


model = Sequential()
model.add(Dense(60, input_dim=60, kernel_initializer='normal',
                activation='relu'))
model.add(Dense(15, 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'])
mc = ModelCheckpoint(filepath=f'best_model.hdf5',
monitor='val_loss',
                     save_best_only=True)
history = model.fit(X, encoded_Y, epochs=100, batch_size=10,
                    validation_split=0.1, callbacks=[mc])
current_model = load_model(f'best_model.hdf5')
current_values = current_model.evaluate(X[187:], encoded_Y[187:])

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'r', label='Training loss')
```

```
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.clf()
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
plt.plot(epochs, acc_values, 'r', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# for best model finding
for denses in range(3, 4):
    for neurons_num in range(5, 60, 5):
        model = Sequential()
        for dense in range(1, denses):
            model.add(Dense(neurons_num, 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'])
        mc = ModelCheckpoint(filepath=f'best_model.hdf5',
monitor='val_loss',
                             save_best_only=True)
        history = model.fit(X, encoded_Y, epochs=100, batch_size=10,
                            validation_split=0.1, callbacks=[mc],
verbose=0)
        current_model = load_model(f'best_model.hdf5')
        current_values = current_model.evaluate(X[187:],
encoded_Y[187:])
        if current_values[0] < best_val_loss:
            best_val_loss = current_values[0]
            best_model = current_model
            best_dense_num = denses
            best_neurons_num = neurons_num
            print(denses, neurons_num, current_values)
```