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

ОТЧЕТ

По индивидуальному заданию
по дисциплине «Искусственные нейронные сети»
ТЕМА: Fashion-MNIST

Студент гр. 7382 _____ Ленковский В.В.

Студент гр. 7382 _____ Находько А.Ю.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель работы.

Задача заключается в классификации одежды по картинке.

Описание датасета.

Наш датасет состоит из изображений размером 28x28, каждый пиксель которого представляет собой оттенок серого.

Рис. 1

Набор данных содержит изображения футболок, топов, сандалей и даже ботинок. Вот полный список того, что содержит в себе наш набор данных MNIST(рис. 2):

Метка	Класс
0	Футболка / топ
1	Шорты
2	Свитер
3	Платье
4	Плащ
5	Сандали
6	Рубашка
7	Кроссовки
8	Сумка
9	Ботинки

Рис. 2 – Классы одежды

Каждому входному изображению соответствует одна из перечисленных выше меток. Набор данных Fashion MNIST содержит 70 000 изображений. Из этих 70 000 мы воспользуемся 60 000 для тренировки нейронной сети и 10 000 для тестирования.

Распределение ответственности.

Ленковский В.В. – разработка алгоритма, обучение и тестирование сети.
Находько А.Ю. – разработка алгоритма, построение нейронной сети.

Ход работы.

1. **Обработка данных**

В качестве входного значения нашей нейронной сети служит одномерный массив длиной 784, массив именно такой длины по той причине, что каждое изображение представляет собой 28x28 пикселей (=784 пикселей всего в изображении), которое мы преобразуем в одномерный массив.

Значение каждого пикселя в изображении находится в интервале [0,255]. Для того, чтобы модель работала корректно эти значения необходимо нормализовать - привести к значениям в интервале [0,1].

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Рис. 3

2. **Создание модели**

Сеть состоит из трёх слоёв:

- **входного** keras.layers.Flatten - этот слой преобразует изображения размером 28x28 пикселей в 1D-массив размером 784 (28 * 28). На этом слое у нас нет никаких параметров для обучения, так как этот слой занимается только преобразованием входных данных.
- **скрытый слой** keras.layers.Dense - плотносвязный слой из 128 нейронов. Каждый нейрон (узел) принимает на вход все 784 значения с предыдущего слоя, изменяет входные значения согласно внутренним

весам и смещениям во время тренировки и возвращает единственное значение на следующий слой.

- **выходной слой** keras.layers.Dense - softmax-слой состоит из 10 нейронов, каждый из которых представляет определённый класс элемента одежды. Как и в предыдущем слое, каждый нейрон принимает на вход значения всех 128 нейронов предыдущего слоя. Веса и смещения каждого нейрона на этом слое изменяются при обучении таким образом, чтобы результирующее значение было в интервале [0,1] и представляло собой вероятность того, что изображение относится к этому классу. Сумма всех выходных значений 10 нейронов равна 1.

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
```

Рис. 4

Перед тем как мы приступим к тренировке модели стоит ещё выполнить несколько настроек. Эти настройки производятся во время сборки модели при вызове метода compile:

- *функция потерь* - алгоритм измерения того, насколько далеко находится желаемое значение от спрогнозированного.
- *функция оптимизации* - агоритм "подгонки" внутренних параметров (весов и смещений) модели для минимизации функции потерь;
- *метрики* - используются для мониторинга процесса тренировки и тестирования. Пример ниже использует такую метрику как точность, процент изображений, которые были корректно классифицированы.

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

Рис. 5

3. Поиск оптимальной сети

1) Начальной архитектурой ИНС была выбрана модель с 1 промежуточным слоем и 10 нейронами на нем, 5 эпох(рис. 6):

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(10, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
```

Рис. 6

Проведем тест ИНС:

Accuracy: 84,42%

Loss: 41,42%

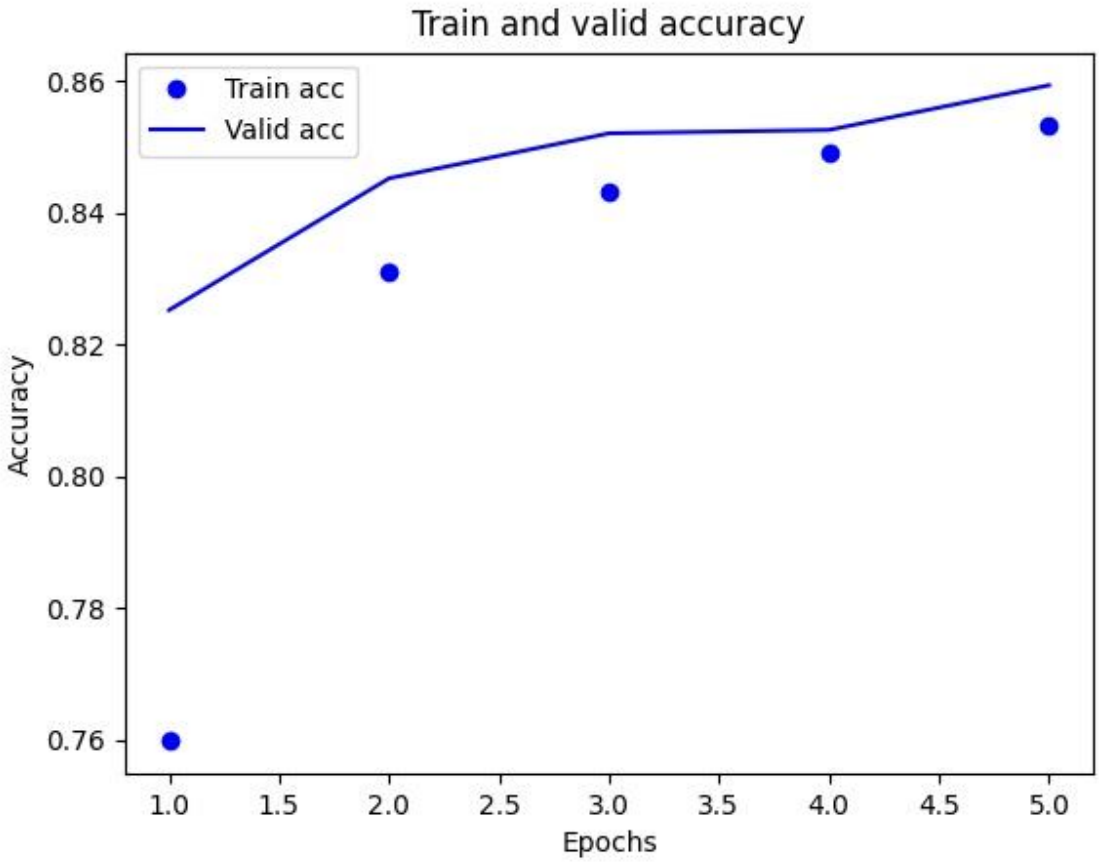

Рис. 7 – График точности

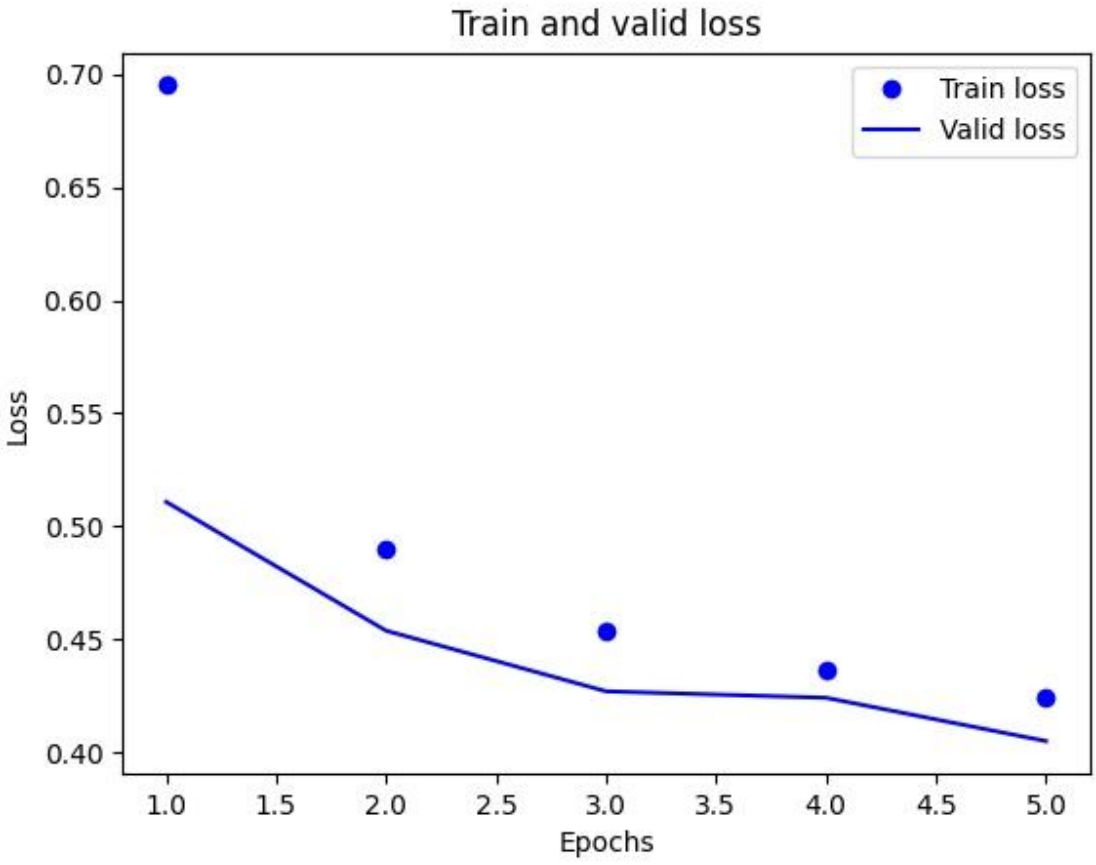

Рис. 8 – График потерь

2) Изменили количество нейронов на промежуточном слое с 10 до 128(рис. 9):

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
```

Рис. 9

Проведем тест ИНС:

Accuracy: 86,9%

Loss: 27,31%

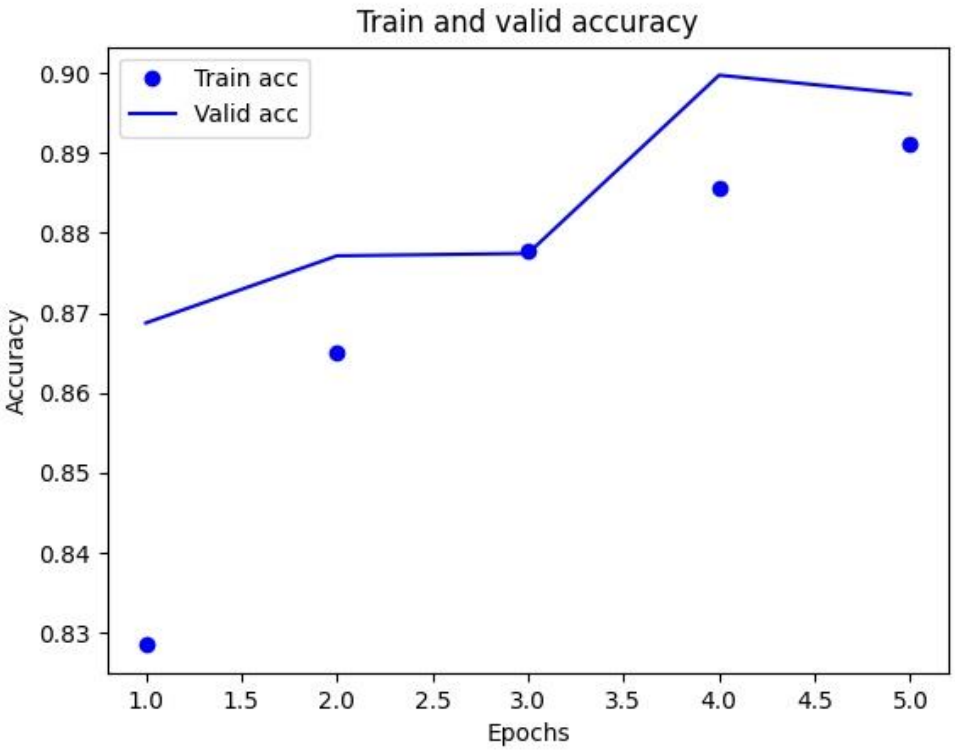

Рис. 10 – График точности

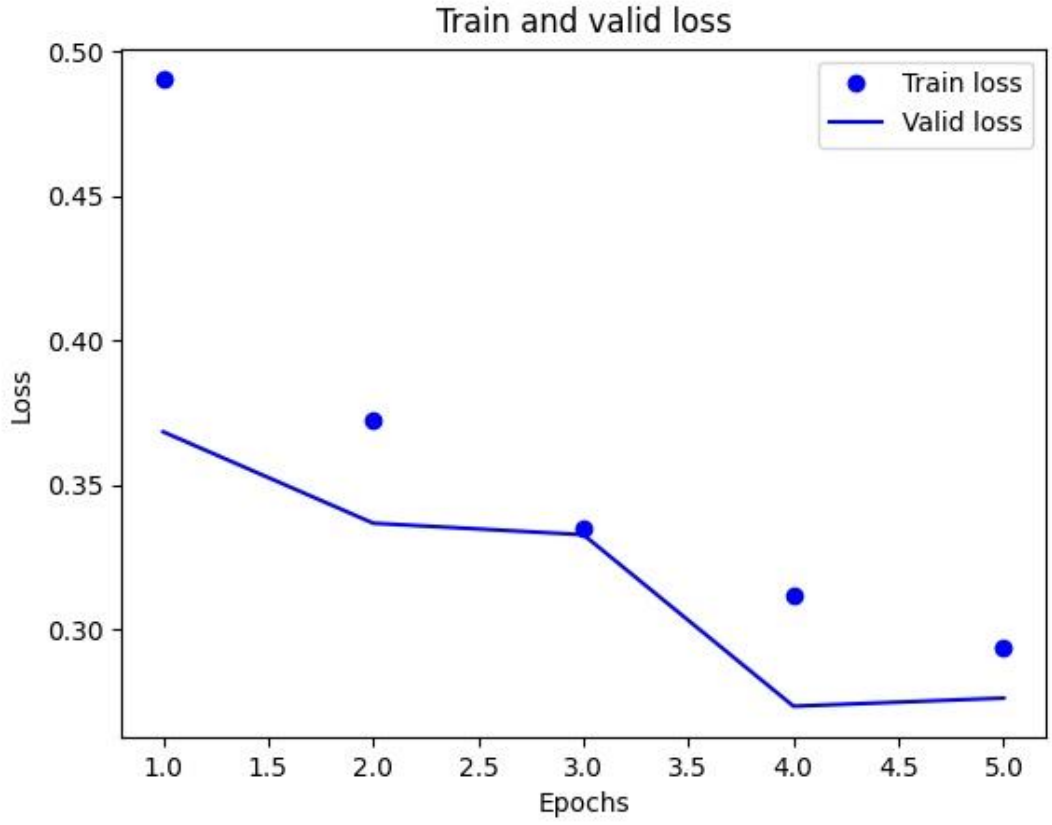

Рис. 11 – График потерь

3) Добавим еще один промежуточный слой с 10 нейронами(рис. 12):

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
```

Рис. 12

Проведем тест ИНС:

Accuracy: 86,9%

Loss: 29,67%

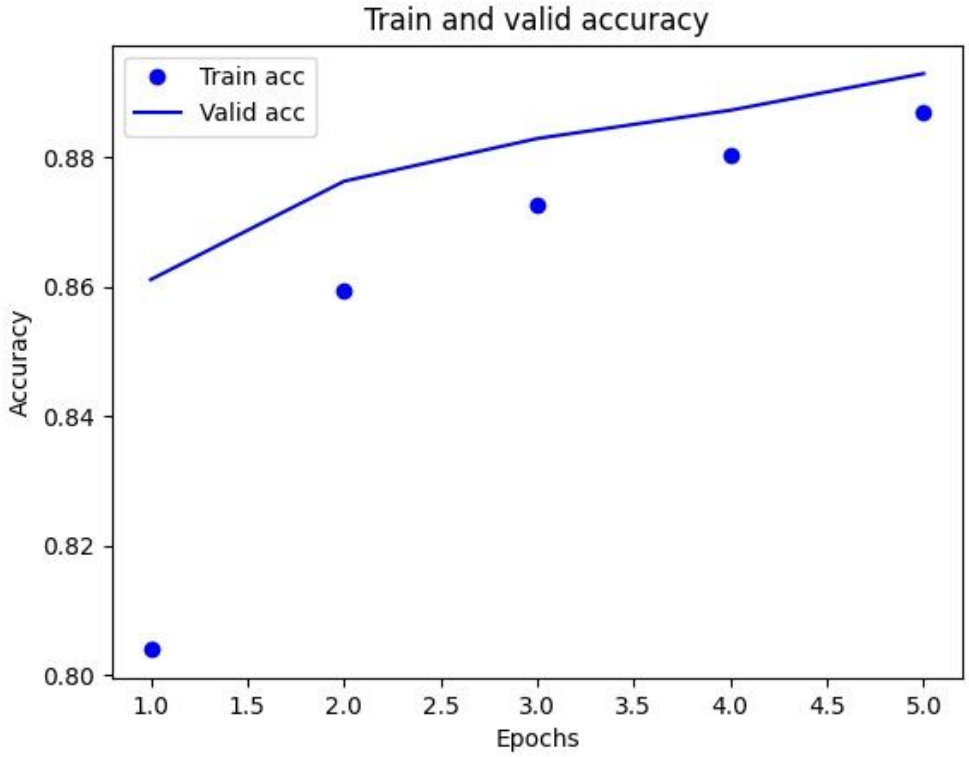

Рис. 13 – График точности

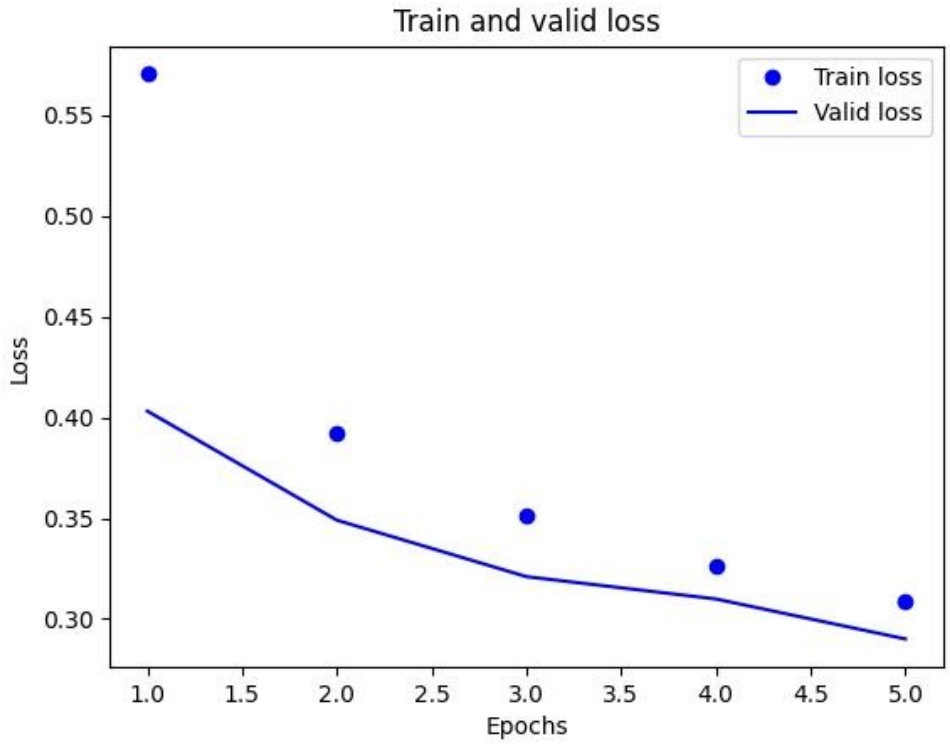

Рис. 14 – График потерь

При увеличении ко-во слоем изменении в точности не обнаружено, но потери увеличились.

4) Сделаем 2 слоя с 128 нейронами(рис. 15):

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
```

Рис. 15

Проведем тест ИНС:

Accuracy: 88,2%

Loss: 26,1%

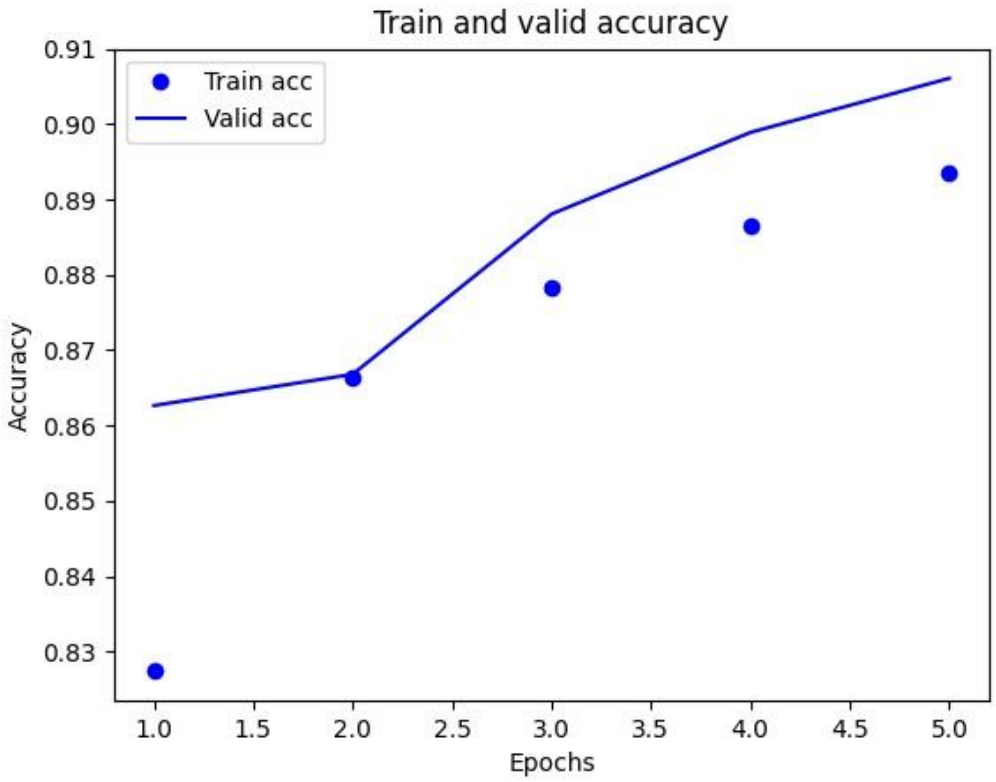

Рис. 16 – График точности

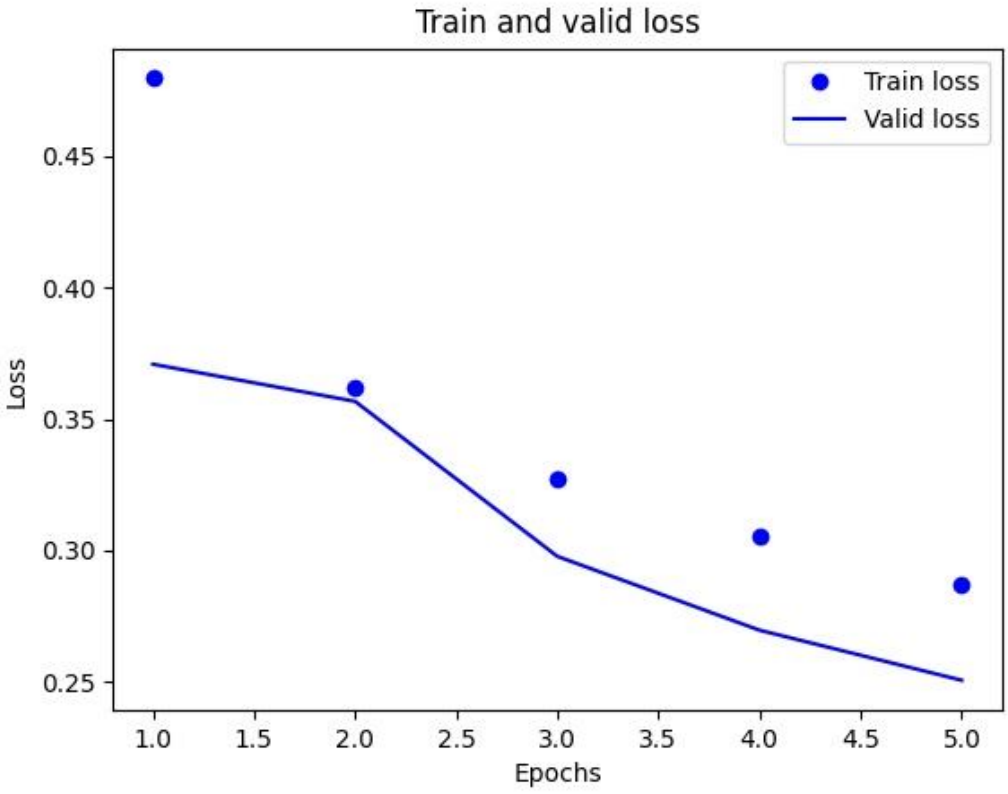

Рис. 17 – График потерь

Увеличение нейронов на добавленном промежуточном слое сказалось хорошо на сети, а именно: увеличилась точность и уменьшились потери.

5) Добавим еще 1 слой с 128 нейронами(рис. 18):

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
```

Рис. 18

Проведем тест ИНС:

Accuracy: 86,87%

Loss: 28,1%

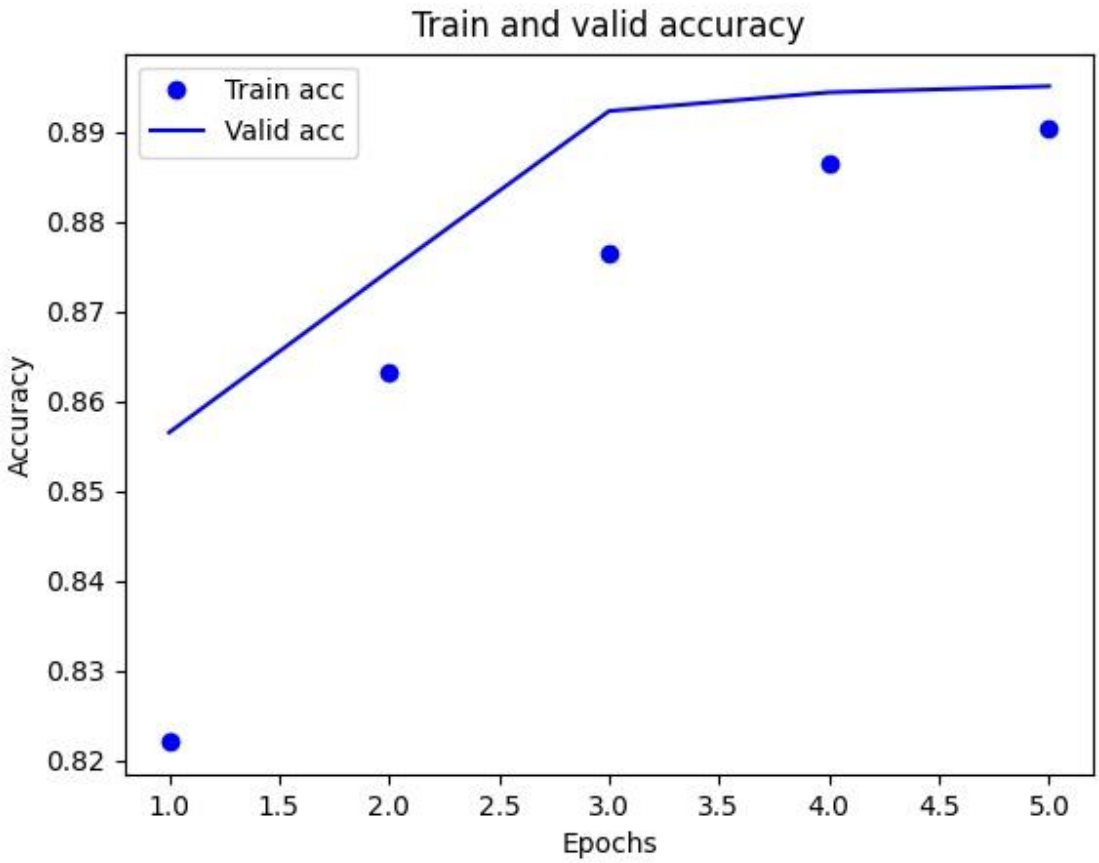

Рис. 19 – График точности

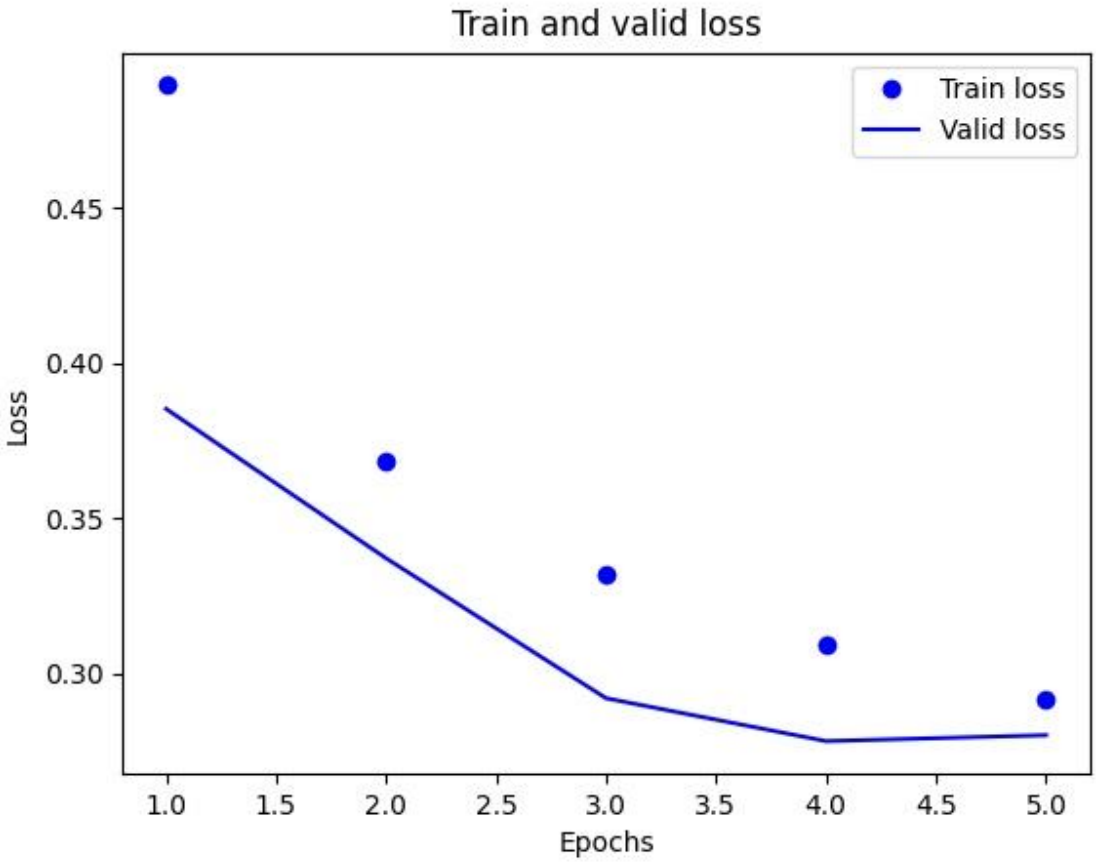

Рис. 20 – График потерь

Добавление ещё одного промежуточного слоя негативно сказывается на модели ИНС, т.к. происходит уменьшение точности

6) Добавим еще 1 Dropout слой(рис. 21):

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dropout(0.2),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
```

Рис. 21

Проведем тест ИНС:

Accuracy: 87,43%

Loss: 27,9%

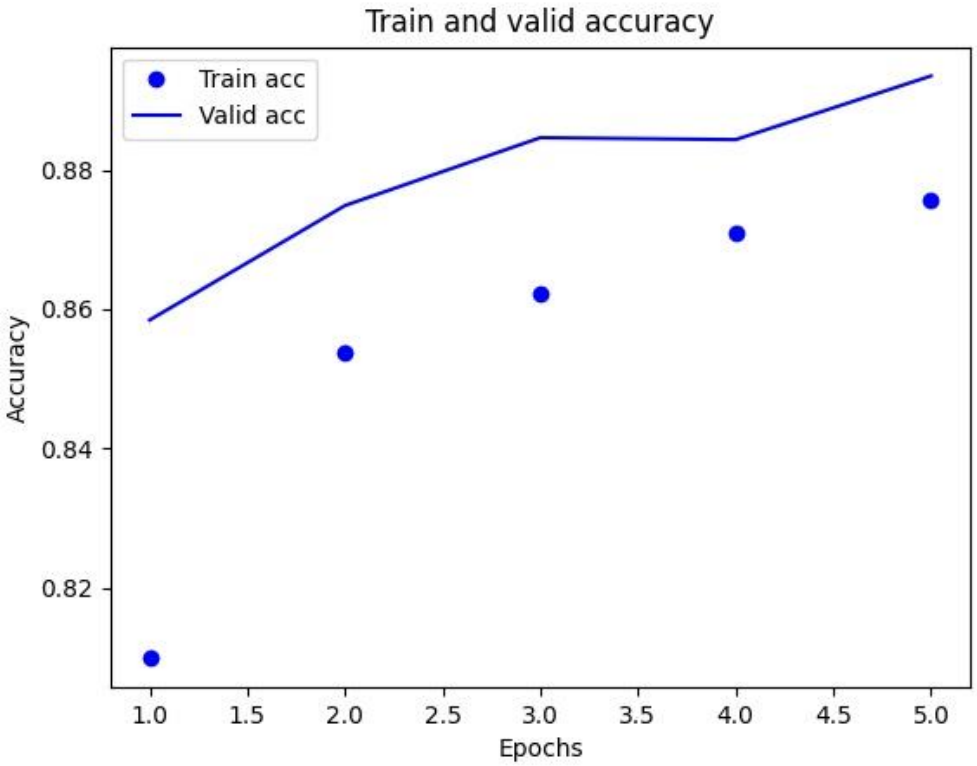

Рис. 22 – График точности

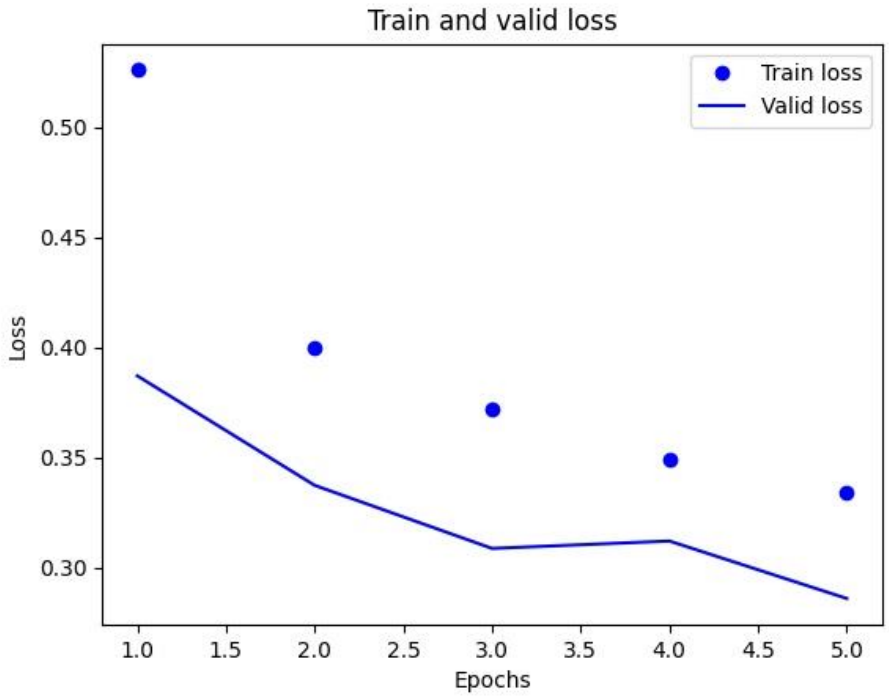

Рис. 23 – График потерь

Как можем заметить из предыдущих двух тестов, добавление ещё одного слоя негативно сказывается на ИНС, поэтому вернёмся к варианту с двумя промежуточными слоями и со 128 нейронами на каждом

7) Увеличим количество эпох до 10 для оптимальной архитектуры(рис.24):

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
```

Рис. 24

Проведем тест ИНС:

Accuracy: 88,72%

Loss: 23,7%

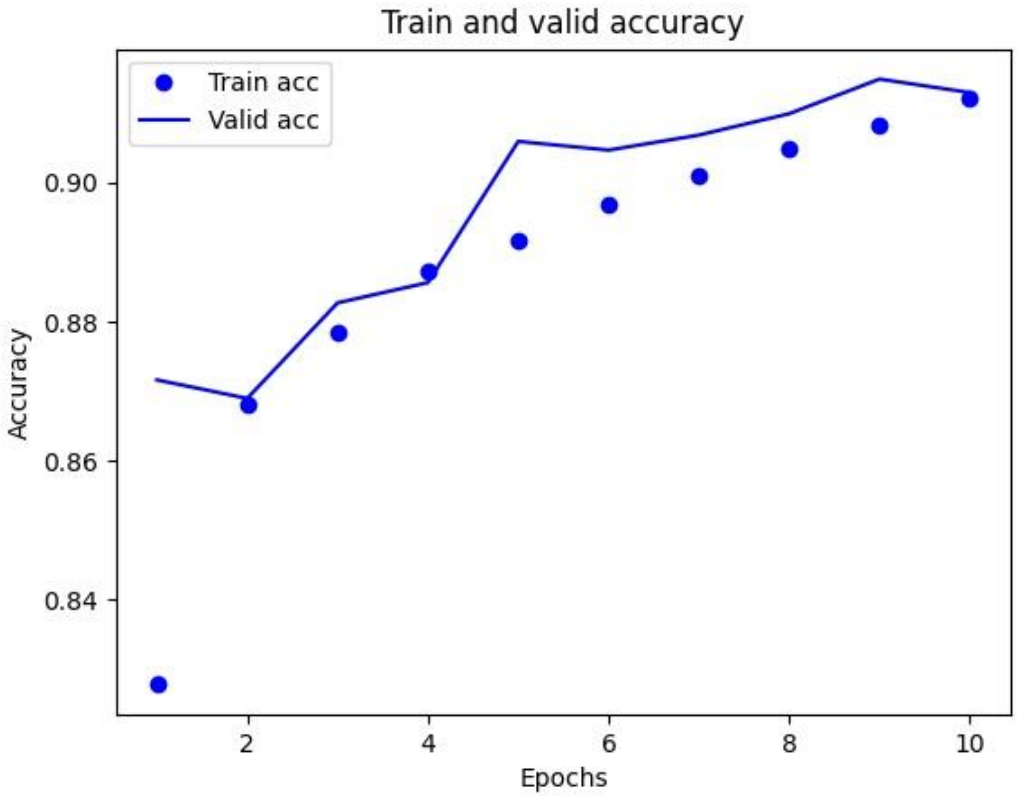

Рис. 25 – График точности

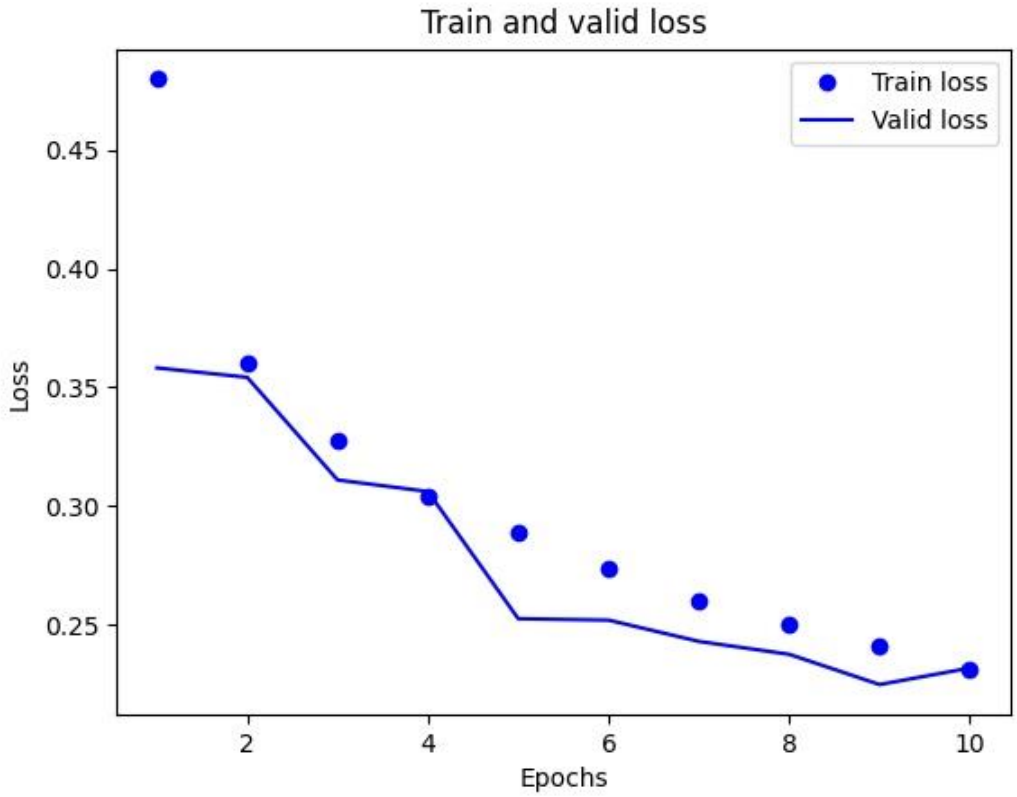

Рис. 26 – График потерь

8) Увеличим количество эпох до 15(рис. 27 и рис. 28):

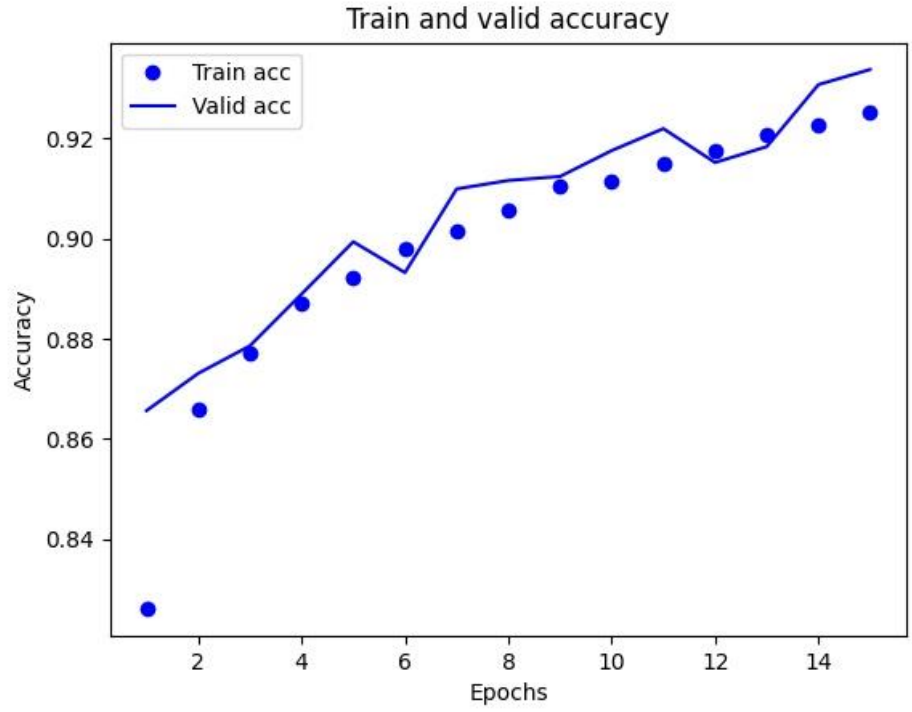

Рис. 27 – График точности

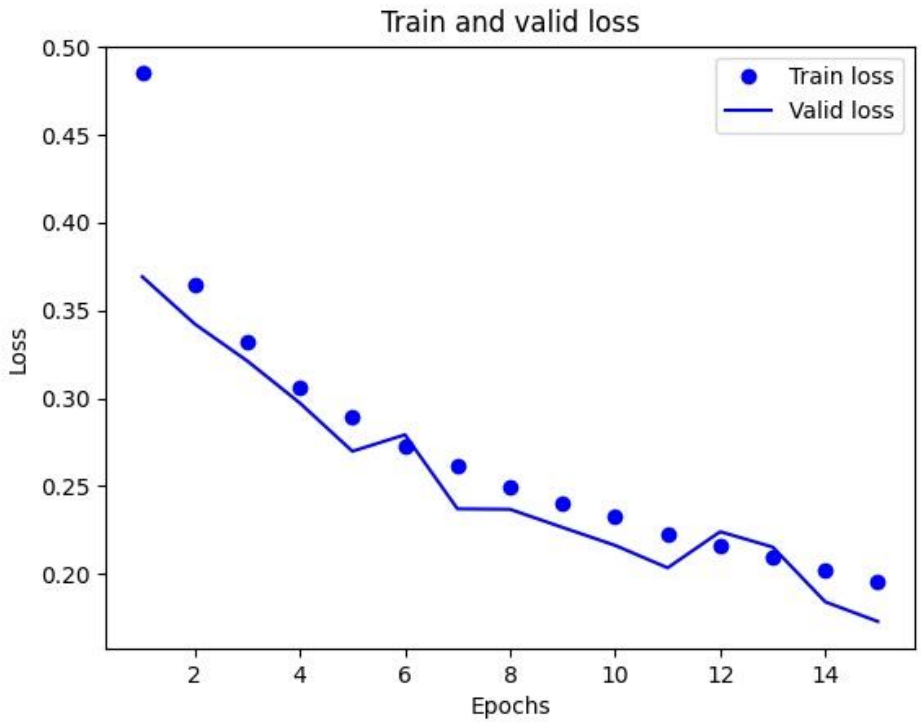

Рис.28 – График потерь

Проведем тест ИНС:

Accuracy: 89,63%

Loss: 18,3%

9) Увеличим количество эпох до 25(рис. 29 и рис. 30):

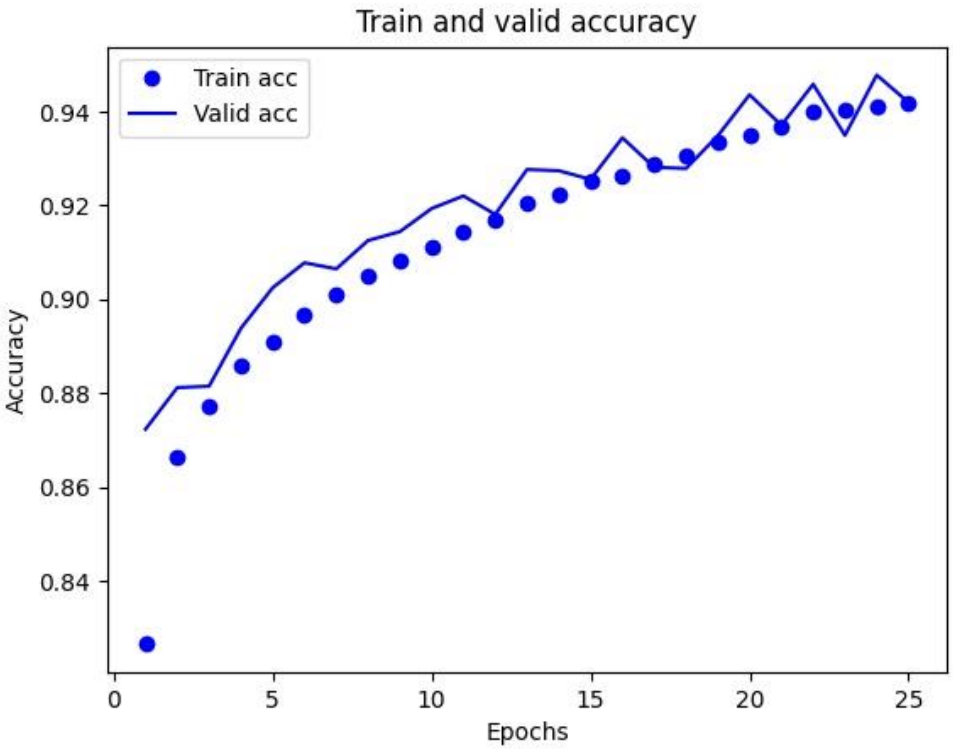

Рис. 29 – График точности

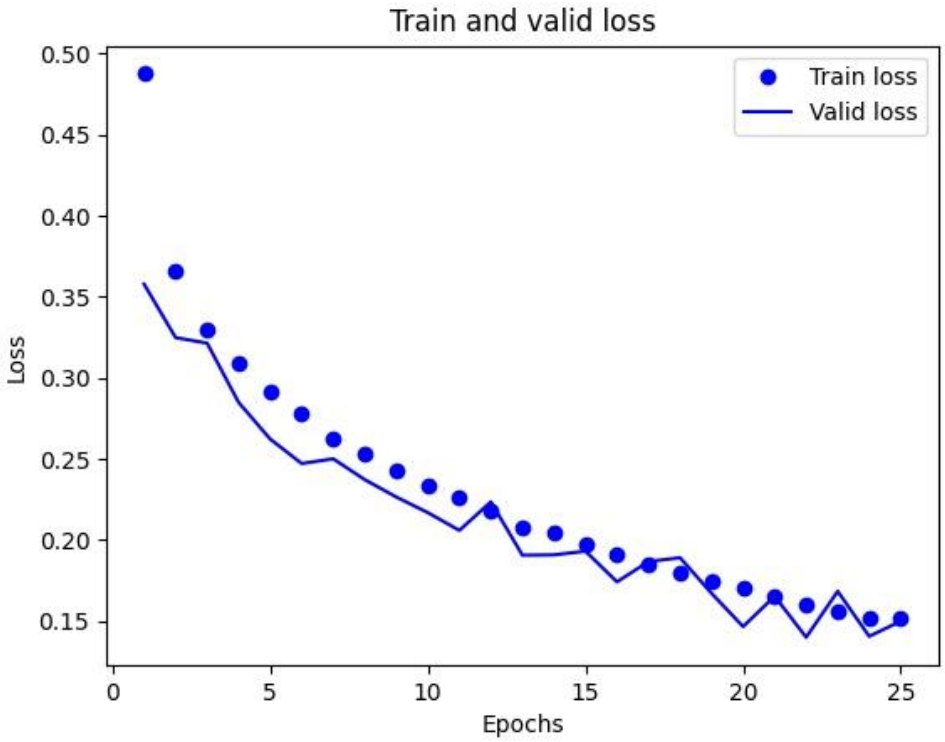

Рис. 30 – График потерь

Проведем тест ИНС:

Accuracy: 90,51%

Loss: 16,1%

10) Увеличим количество эпох до 35(рис. 31 и рис. 32):

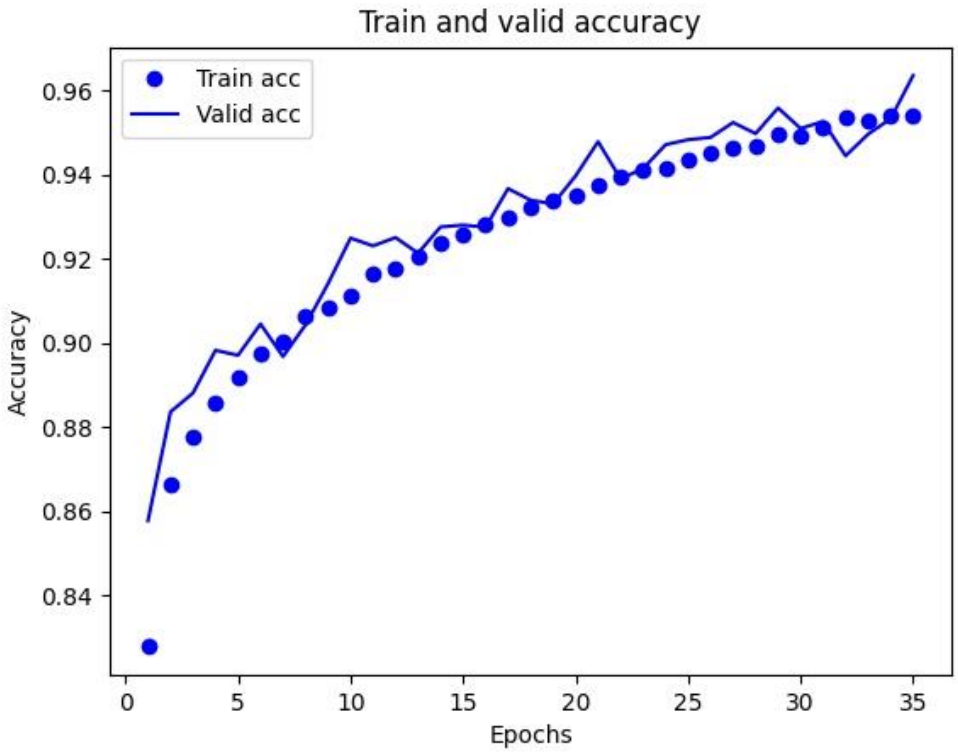

Рис. 31 – График точности

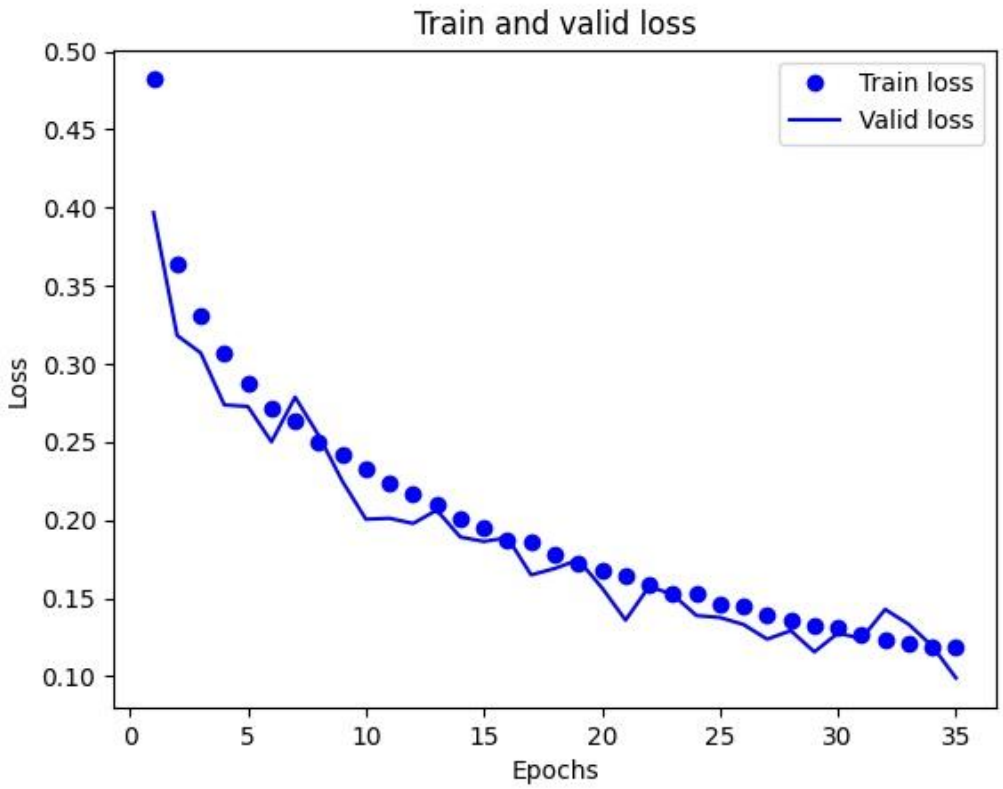

Рис. 32 – График потерь

Проведем тест ИНС:

Accuracy: 92,07%

Loss: 13,6%

Анализ результирующей модели.

Сделаем вывод по графикам, что увеличение эпох положительно сказывается на модели ИНС, т.к. возрастает точность и падают потери. Оптимальной будем считать модель, которая обучается за 30 эпох, т.к. дальнейшее увеличение эпох существенно не сказывается на модели.

Полученную оптимальную модель протестировали более 10 раз, результаты оказались устойчивыми, среднее значение точности получилось около 91,5%, а потерь – 14,1%.

Вывод.

В ходе выполнения ИДЗ была закреплены знания по обработке данных и работе с ними. Была проведена работа с датасетом Fashion-MNIST, который содержит 70 000 объектов данных(а именно картинок вещей), которые надо разделить на 10 классов. Была разработана модель на языке Python с использованием Keras API, которая выдает приемлемые результаты.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
# Подключение необходимых библиотек
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

# Получение данных
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Наименование искомых классов данных
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# преобразуем к значению от 0 до 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Настройка слоёв
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])

# Компиляция модели
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Тренировка модели
H = model.fit(train_images, train_labels, epochs=30, validation_data = (train_images, train_labels))

# Оценка точности
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)

print('\nТочность на проверочных данных:', test_acc)
# Предсказание модели
predictions = model.predict(test_images)

# Создание графиков
loss = H.history['loss']
val_loss = H.history['val_loss']
acc = H.history['accuracy']
val_acc = H.history['val_accuracy']
epochs = range(1, len(loss) + 1)

# График потерь
plt.plot(epochs, loss, 'bo', label='Train loss')
plt.plot(epochs, val_loss, 'b', label='Valid loss')
plt.title('Train and valid loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.clf()
```

```
# График точности
plt.plot(epochs, acc, 'bo', label='Train acc')
plt.plot(epochs, val_acc, 'b', label='Valid acc')
plt.title('Train and valid accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```