Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Институт информационных технологий и анализа данных

наименование института

ОТЧЕТ  
к лабораторной работе №3 по дисциплине

Интеллектуальные системы и технологии

«Анализ данных»

Наименование темы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы: | ИСМб-19-1 |  |  |  | Вовиков Д.Е. |
|  | Шифр группы |  | подпись |  | Фамилия ИО |
| Проверил |  |  |  |  | Массель А.Г. |
|  | должность |  | подпись |  | Фамилия ИО |
| Содержание отчета на: | |  | 9 | стр. | |

Иркутск 2022 г

Содержание

[1 Введение 3](#_Toc121427649)

[1.1 Цель лабораторной 3](#_Toc121427650)

[1.2 Задачи 3](#_Toc121427651)

[1.3 Описание Video Game Sales 3](#_Toc121427652)

[2 Таблица и гипотезы, которые можно вывести 4](#_Toc121427653)

[2.1 Использованные данные: 4](#_Toc121427654)

[2.2 Гипотезы: 4](#_Toc121427655)

[3 Программная реализация 5](#_Toc121427656)

[3.1 Чистка данных 5](#_Toc121427657)

[3.2 Обработка данных 5](#_Toc121427658)

[3.2.1 Генерация цветовой схемы 5](#_Toc121427659)

[3.2.2 Сортировка объектов 5](#_Toc121427660)

[3.3 Построение графиков 5](#_Toc121427661)

[3.3.1 График жанров 5](#_Toc121427662)

[3.3.2 График издательств 5](#_Toc121427663)

[3.3.3 График платформ 6](#_Toc121427664)

[3.3.4 График годов издания 6](#_Toc121427665)

[4 Графики и выводы по ним 7](#_Toc121427666)

[5 Машинное обучение 9](#_Toc121427667)

[5.1 Формирование обучающей и тестовой выборок 9](#_Toc121427668)

[5.2 Описание процессинга данных 10](#_Toc121427669)

[5.3 Архитектура нейронной сети 11](#_Toc121427670)

[5.4 Листинг кода 14](#_Toc121427671)

[5.5 Проверка работоспособности нейронной сети 15](#_Toc121427672)

1 Введение

1.1 Цель лабораторной

Разобрать пример исследования и классификации данных с использованием некоторых библиотек на Python

1.2 Задачи

* выбрать интересующий нас набор данных (DataSet). Разнообразные наборы Dataset'ы можно скачать с [1]. DataSet обычно представляет собой файл с таблицей в формате JSON или CSV.
* Провести анализ данных и сделать выводы.
* \*Разбить датасет на 2 части (20/80) применить методы машинного обучения для прогнозирования.

1.3 Описание Video Game Sales

Этот набор данных содержит список видеоигр, продажи которых превышают 100 000 копий. Он был сгенерирован на основе данных vgchartz.com.

Поля включают

Rank – Рейтинг общих продаж

Name – Название игры

Platform — платформа выпуска игр (например, ПК, PS4 и т. д.)

Year – Год выхода игры.

Genre – Жанр игры.

Publisher – Издатель игры

NA\_Sales — продажи в Северной Америке (в миллионах)

EU\_Sales — Продажи в Европе (в миллионах)

JP\_Sales — продажи в Японии (в миллионах)

Other\_Sales — Продажи в остальном мире (в миллионах)

Global\_Sales — общий объем продаж по всему миру.

2 Таблица и гипотезы, которые можно вывести

2.1 Использованные данные:

Platform — платформа выпуска игр (например, ПК, PS4 и т. д.)

Year – Год выхода игры.

Genre – Жанр игры.

Publisher – Издатель игры

2.2 Гипотезы:

1. Самая распространенная платформа
2. Самый распространенный год
3. Самый распространенный жанр
4. Самый распространенный издатель игр

3 Программная реализация

3.1 Чистка данных

df = df.replace({None: 'unknown'})

df = df.replace({np.NaN: 'unknown'})

3.2 Обработка данных

3.2.1 Генерация цветовой схемы

PLOT\_LABEL\_FONT\_SIZE = 25

def getColors(n):

COLORS = []

cm = plt.cm.get\_cmap('hsv', n)

for i in np.arange(n):

COLORS.append(cm(i))

return COLORS

3.2.2 Сортировка объектов

def dict\_sort(my\_dict):

keys = []

values = []

my\_dict = sorted(my\_dict.items(), key=lambda x:x[1], reverse=True)

for k, v in my\_dict:

keys.append(k)

values.append(v)

return (keys,values)

3.3 Построение графиков

3.3.1 График жанров

breed\_count = pd.value\_counts(df['Genre'].values, sort=True)

breed\_count\_keys, breed\_count\_values = dict\_sort(dict(breed\_count))

TOP\_breed = len(breed\_count\_keys)

plt.figure(figsize=(20,10))

plt.title('Жанры', fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.bar(np.arange(TOP\_breed), breed\_count\_values, color=getColors(TOP\_breed)) #метод построения гистаграмм

plt.xticks(np.arange(TOP\_breed), breed\_count\_keys, rotation=90, fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.yticks(fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.ylabel('Количество игр в жанре', fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.show()

3.3.2 График издательств

breed\_count = pd.value\_counts(df['Publisher'].values, sort=True)

breed\_count\_keys, breed\_count\_values = dict\_sort(dict(breed\_count))

print(range(len(breed\_count\_keys)))

del breed\_count\_values[50:579]

del breed\_count\_keys[50:579]

TOP\_breed = len(breed\_count\_keys)

plt.figure(figsize=(20,10))

plt.title('Издатель', fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.bar(np.arange(TOP\_breed), breed\_count\_values, color=getColors(TOP\_breed)) #метод построения гистаграмм

plt.xticks(np.arange(TOP\_breed), breed\_count\_keys, rotation=90, fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.yticks(fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.ylabel('Количество игр в издании', fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.show()

3.3.3 График платформ

breed\_count = pd.value\_counts(df['Platform'].values, sort=True)

breed\_count\_keys, breed\_count\_values = dict\_sort(dict(breed\_count))

TOP\_breed = len(breed\_count\_keys)

plt.figure(figsize=(20,10))

plt.title('Платформы', fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.bar(np.arange(TOP\_breed), breed\_count\_values, color=getColors(TOP\_breed)) #метод построения гистаграмм

plt.xticks(np.arange(TOP\_breed), breed\_count\_keys, rotation=90, fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.yticks(fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.ylabel('Количество игр на платформе', fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.show()

3.3.4 График годов издания

breed\_count = pd.value\_counts(df['Year'].values, sort=True)

breed\_count\_keys, breed\_count\_values = dict\_sort(dict(breed\_count))

del breed\_count\_values[20]

del breed\_count\_keys[20]

TOP\_breed = len(breed\_count\_keys)

plt.figure(figsize=(20,10))

plt.title('Год издания', fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.bar(np.arange(TOP\_breed), breed\_count\_values, color=getColors(TOP\_breed)) #метод построения гистаграмм

plt.xticks(np.arange(TOP\_breed), breed\_count\_keys, rotation=90, fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.yticks(fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.ylabel('Количество игр изданных в год', fontsize=PLOT\_LABEL\_FONT\_SIZE)

plt.show()

4 Графики и выводы по ним

Самые распространенный жанр это Action.

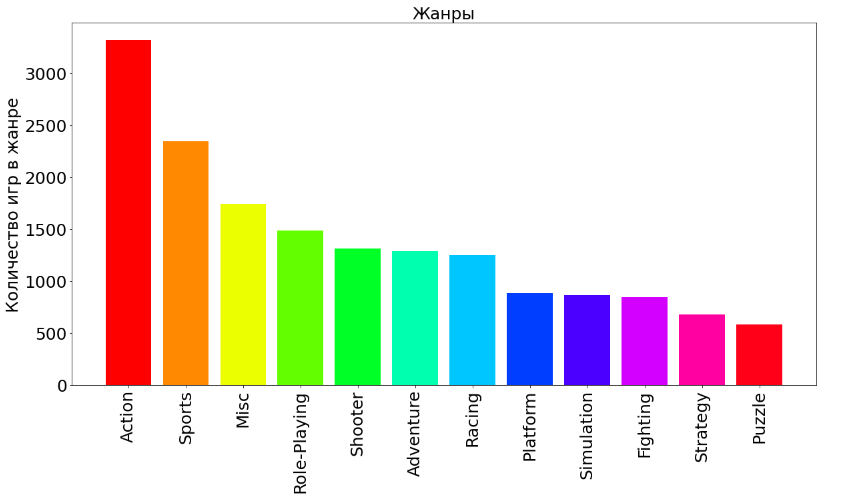


Рисунок 1 – Количество игр в жанрах

Самый распространённый издатель игр – это Electronic Arts.

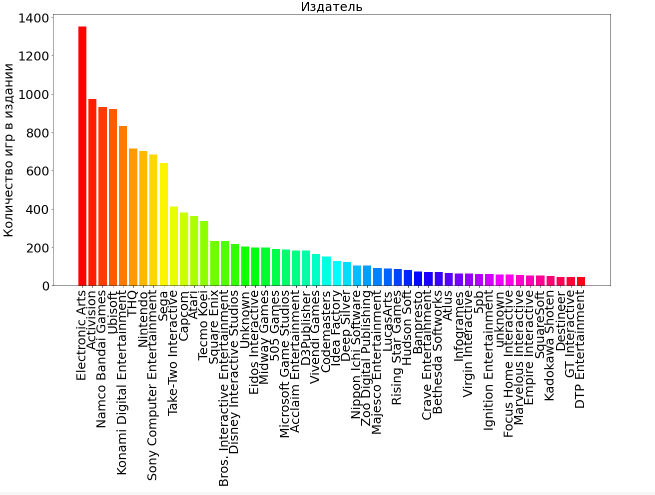


Рисунок 2 – Количество игр в издании

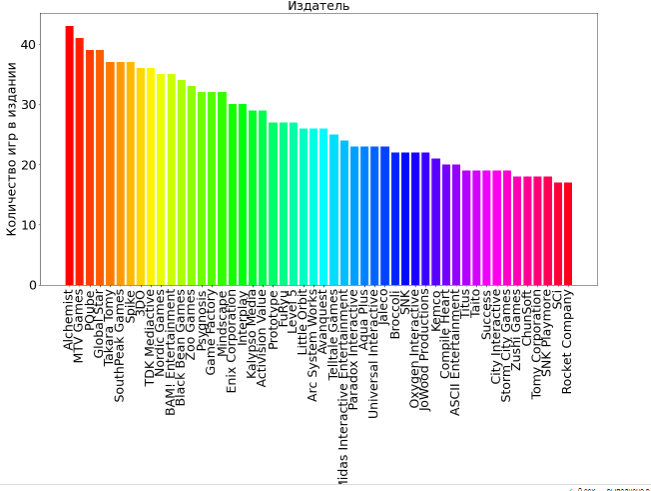


Рисунок 3 – Количество игр в издании

Самые распространенные платформы – это DS и PS2.

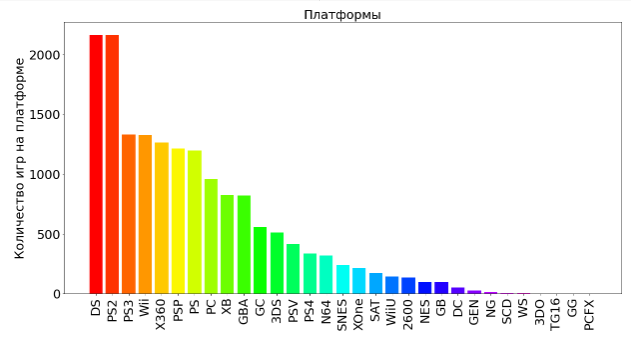


Рисунок 4 – Количество игр на платформе

Самый распространённый год для издания игра – это 2009 и 2008.

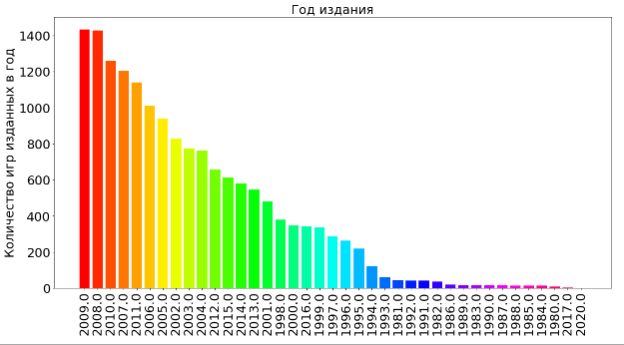


Рисунок 5 – Количество игр изданный в год

5 Машинное обучение

5.1 Формирование обучающей и тестовой выборок

В качестве выборки был использован dataset fashion\_mnist. Данный набор данных содержит 50000 цветных учебных изображений разрешением 28×28 пикселей, помеченных в 10 классах, и 10000 тестовых изображений. Метки данных классов представлены следующим образом:

Таблица 1 – Представление меток классов выборок

|  |  |
| --- | --- |
| Метка | Описание метки |
| 0 | T-shirt |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

Ниже представлен фрагмент кода формирования обучающей и тестовой выборок с использованием картежей. В переменную x\_train заносится набор изображений, в переменную y\_train набор меток. Аналогичные действия совершаются с переменными x\_test, y\_test.

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

5.2 Описание процессинга данных

Перед началом обучения нейронной сети был произведен процесс нормализации изображений.

x\_train = x\_train / 255

x\_test = x\_test / 255

Данный фрагмент кода нормализует изображения, имеющие первоначальные значения пикселя в пределах от 0 до 255 (где 0 – темный цвет пикселя, 255 – светлый цвет пикселя) в изображения с пикселя, находящихся в пределах от 0 до 1.

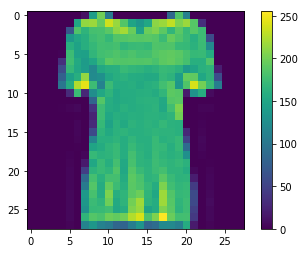


Рисунок 6 – Изображение до нормализации

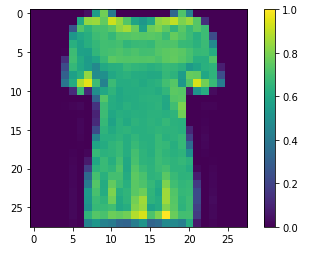


Рисунок 7 – Изображение после нормализации

5.3 Архитектура нейронной сети

За основу нейронной сети из библиотеки Keras был взят класс Sequential – класс архитектура нейронных сетей которого строится в виде последовательности слоев, начиная с входного и заканчивая выходным.

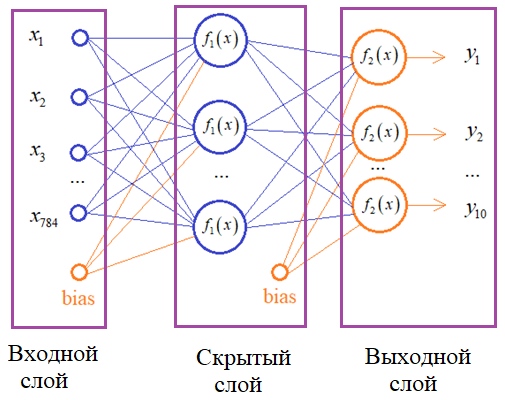


Рисунок 8 – Пример последовательной нейронной сети

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28,28)), #Преобрабование изорбажения в массив

keras.layers.Dense(128, activation="relu"), #Входной слой (полносвязный)

keras.layers.Dense(10, activation="softmax")

])

**В качестве входного слоя** используется слой Flatten – используется для конвертации входящих данных в меньшую размерность. В результате параметры размерности изображения будут преобразованы в 784 нейрона входного слоя.

**В качестве скрытого слоя** применяется слой Dense – плотно связанный слой. В качестве его входных параметров служат количество нейронов и функцией активации ReLu.

ReLu возвращает значение х, если х положительно, и 0 в противном случае.

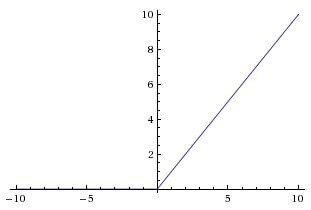


Рисунок 9 – График функции ReLu

На первый взгляд кажется, что ReLu имеет все те же проблемы, что и линейная функция, так как ReLu линейна в первом квадранте. Но на самом деле, ReLu нелинейна по своей природе, а комбинация ReLu также нелинейна. Такая функция является хорошим аппроксиматором, так как любая функция может быть аппроксимирована комбинацией ReLu. Это означает, что мы можем стэкать слои. Область допустимых значений   
ReLu — [0,inf), то есть активация может “взорваться”.

ReLu менее требовательно к вычислительным ресурсам, чем гиперболический тангенс или сигмоида, так как производит более простые математические операции.

**В качестве выходного слоя** аналогично применяется слой Dense. В качестве его параметров служат количество нейронов и функцией активации softmax.

Функция softmax, также известная как softargmax или нормализованная экспоненциальная функция – это обобщение логистической функции на несколько измерений. Он используется в полиномиальной логистической регрессии и часто используется в качестве последней функции активации нейронной сети для нормализации выходных данных сети до распределения вероятностей по прогнозируемым выходным классам.

Далее происходит компилирование оптимизатора, функции потерь и метрики.

model.compile(optimizer=tf.keras.optimizers.SGD(), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

**В качестве оптимизатора** выступает SGD. Стохастический градиентный спуск (англ. Stochastic gradient descent, SGD) — итерационный метод для оптимизации целевой функции с подходящими свойствами гладкости (например, дифференцируемость или субдифференцируемость). Его можно расценивать как стохастическую аппроксимацию оптимизации методом градиентного спуска, поскольку он заменяет реальный градиент, вычисленный из полного набора данных его оценкой, вычисленной из случайно выбранного подмножества данных.

**В качестве альтернативы** можно использовать оптимизатор Adam.

Adam (сокращение от «метод адаптивной оценки моментов», англ. Adaptive Moment Estimation) — это обновление оптимизатора RMSProp. В этом оптимизационном алгоритме используются скользящие средние как градиентов, так и вторых моментов градиентов. Если даны параметры , а функция потерь , где t отражает индекс текущей итерации (отчёт начинается с 0), пересчёт параметра алгоритмом Adam задаётся формулами:

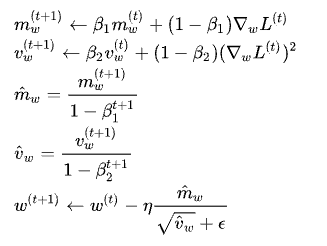


Рисунок 10 – формулы пересчета параметра алгоритмом Adam

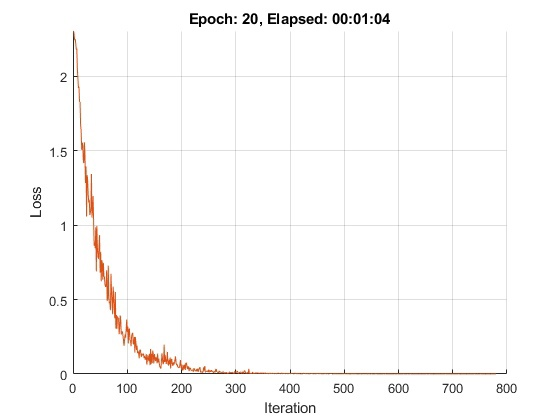


Рисунок 11 – график работы оптимизатора Adam

**В качестве функции потерь** используется sparse\_categorical\_crossentropy – данная функция вычисляет потерю кроссэнтропии между метками и прогнозами.

**В качестве метрики** используется метод accuracy. Accuracy — это показатель, который описывает общую точность предсказания модели по всем классам. Это особенно полезно, когда каждый класс одинаково важен. Он рассчитывается как отношение количества правильных прогнозов к их общему количеству.

Результат зависимости функции потерь и метрики от эпохи представлены на рисунках 7 и 8.

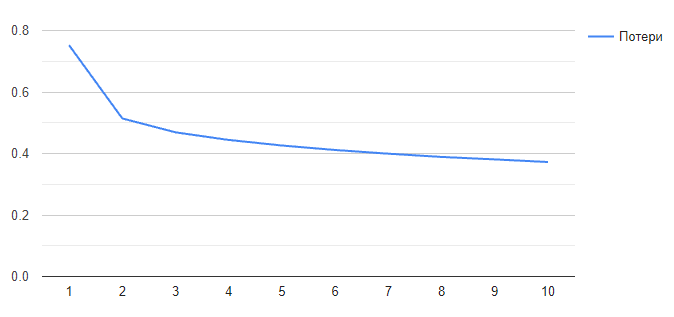


Рисунок 12 – график зависимости функции потерь от эпохи

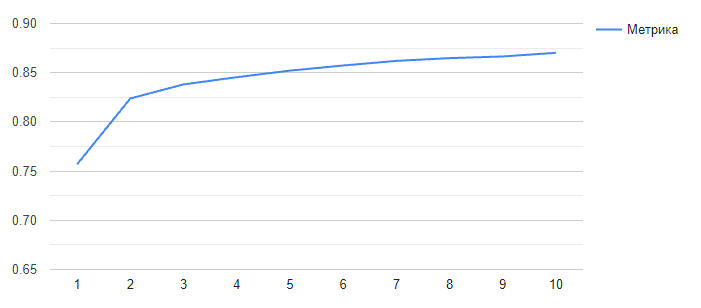


Рисунок 13 – график зависимости функции метрики от эпохи

Исходя из рисунков можно определить, что с каждой эпохой значение функции потери падает, а значение метрики растет.

5.4 Листинг кода

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow import keras

%matplotlib inline

from tensorflow.keras.datasets import fashion\_mnist #В обучающей выборке 60 тыс. рисунков и 10 тыс в тестовой

from tensorflow.keras.models import Sequential #Последовательная модель слоев

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras import utils

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

class\_names = ['T-shirt', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'] #Даем название классам

plt.figure()

plt.imshow(x\_train[10])

plt.colorbar()

plt.grid(False)

x\_train = x\_train / 255 #Делим на 255, чтобы диапазон входных данных был в пределах от 0 до 1

x\_test = x\_test / 255

plt.figure()

plt.imshow(x\_train[10])

plt.colorbar()

plt.grid(False)

plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.imshow(x\_train[i])

plt.xlabel(class\_names[y\_train[i]])

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28,28)), #Преобрабование изорбажения в массив

keras.layers.Dense(128, activation="relu"), #Входной слой (полносвязный)

keras.layers.Dense(10, activation="softmax")

])

model.compile(optimizer=tf.keras.optimizers.SGD(), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.summary()

model.fit(x\_train, y\_train, epochs=10) #Обучение с учителем (что видно, но если забуду)

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print('Точность предсказания:', test\_acc)

predict = model.predict(x\_train)

np.argmax(predict[k])

y\_train[k]

plt.figure()

plt.imshow(x\_train[k])

plt.colorbar()

plt.grid(False)

class\_names[np.argmax(predict[k])]

5.5 Проверка работоспособности нейронной сети

Проверка работоспособности нейронной сети осуществляется следующим образом:

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print('Точность предсказания:', test\_acc)

Evaluate() вычисляет значение потерь и значения всех метрик, которые мы выбрали при составлении модели. В качестве параметров указывается тестовая выборка и набор меток для тестов.

Результат оценки работоспособности нейронной сети представлен на рисунке 9.

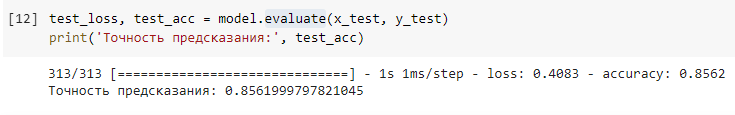


Рисунок 14 – Результат оценки нейронной сети

Исходя из рисунка 9 можно сделать вывод, что точность обучения нейронной сети составляет 0.8562, а уровень ошибки 0.4083.

Также сеть была вручную протестирована на 5 объектах тестовой выборки. Результаты представлены на рисунках 10-14.

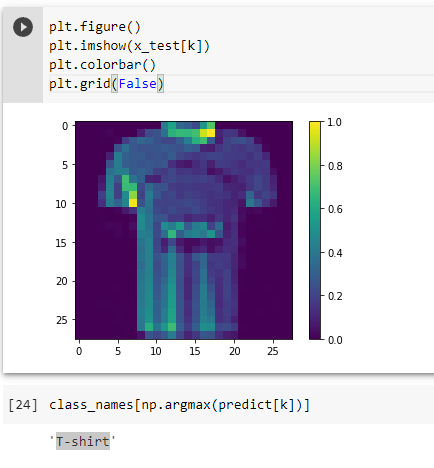


Рисунок 15 – Результат теста №1

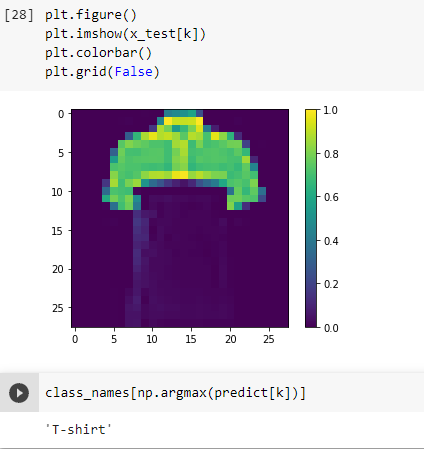


Рисунок 16 – Результат теста №2

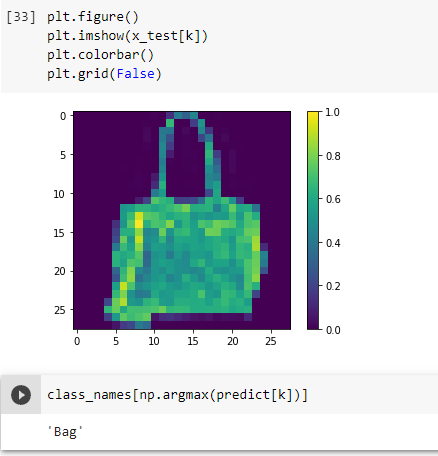


Рисунок 17 – Результат теста №3

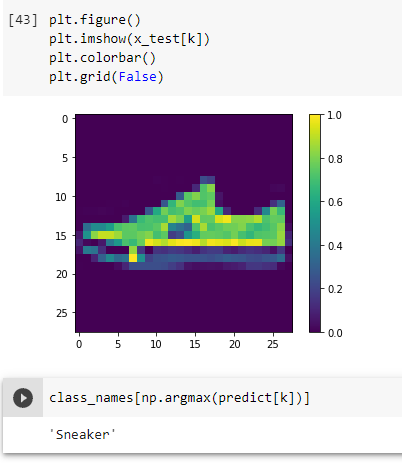


Рисунок 18 – Результат теста №4

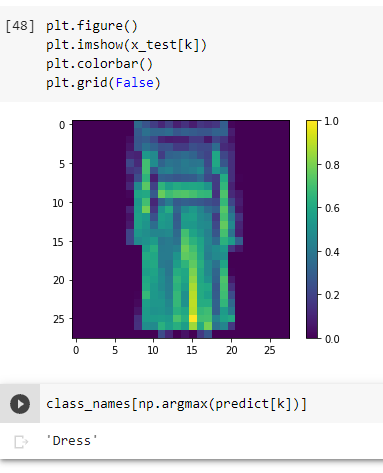


Рисунок 19 – Результат теста №5