

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»  
МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ

ГЕРАСИМОВ ЛЕОНИД АЛЕКСЕЕВИЧ

**МОЯ ПЕРВАЯ НЕЙРОННАЯ СЕТЬ. РАСПОЗНАВАНИЕ ПРЕДМЕТА ОДЕЖДЫ  
ПО ФОТОГРАФИИ**

ПРОЕКТ СТУДЕНТА ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ БАКАЛАВРИАТА  
«ПРИКЛАДНАЯ МАТЕМАТИКА»  
ПО НАПРАВЛЕНИЮ ПОДГОТОВКИ 01.03.04 ПРИКЛАДНАЯ МАТЕМАТИКА

Студент  
Л. А. Герасимов

Руководитель проекта  
Доктор физико-математических наук, профессор  
В.Ю. Попов

Москва 2024г.

# Содержание

<b>1</b>	<b>Аннотация</b>	<b>2</b>
<b>2</b>	<b>Введение</b>	<b>2</b>
<b>3</b>	<b>Этапы выполнения проекта</b>	<b>3</b>
3.1	Подключение необходимых библиотек и модулей . . . . .	3
3.2	Загрузка обучающей и тестовой выборки . . . . .	3
3.3	Создание списка с названиями предметов одежды . . . . .	4
3.4	Отображение первых 25 картинок . . . . .	4
3.5	Изменение размерности данных в наборе . . . . .	5
3.6	Стандартизация данных . . . . .	6
3.7	Преобразование выходных значений в векторы по категориям . . . . .	6
3.8	Формирование нейронной сети и вывод её структуры . . . . .	6
3.9	Компиляция модели нейронной сети . . . . .	7
3.10	Обучение нейронной сети . . . . .	8
3.11	Визуализация точности и функции ошибки на графиках . . . . .	8
3.12	Запуск распознавания набора данных . . . . .	9
3.13	Проверка результата для произвольного значения . . . . .	9
3.14	Некоторые случаи неправильной работы нейронной сети . . . . .	10
3.15	Распознавание изображения по загружаемой фотографии . . . . .	11
3.15.1	Описание программного кода . . . . .	11
3.15.2	Примеры работы программы с загружаемыми изображениями . . . . .	12
<b>4</b>	<b>Заключение</b>	<b>14</b>
	<b>Список литературы</b>	<b>15</b>

# 1 Аннотация

В рамках данного проекта решается задача обучения модели классификации, которая сможет определить, к какому классу одежды относится предмет на изображении. Для этого используется язык программирования Python. Работа реализуется с помощью библиотеки TensorFlow и её высокоуровневого интерфейса Keras. Для написания программного кода применяется среда Google Colab.

В результате выполнения работы будет создана модель нейронной сети, которая сможет с высокой точностью распознать предметы гардероба из тестовой выборки используемого набора данных, а также определить класс одежды на отдельно загруженной фотографии.

## 2 Введение

Целью проекта является создание простейшей нейронной сети на языке Python с использованием библиотеки Keras. Ожидаемые результаты для меня были следующими: я пойму структуру нейронной сети, научусь самостоятельно писать сеть.

В качестве набора данных было решено задействовать датасет Fashion MNIST[1], содержащий изображения одежды 10 различных категорий (футболка, брюки, свитер, платье, пальто, туфли, рубашка, кроссовки, сумка, ботинки). Всего в наборе содержится 70000 фотографий в оттенках серого, из которых 60000 приходится на обучение, а ещё 10000 – на тестирование. Размер каждой картинке из набора –  $28 \times 28$  пикселей. Все изображения записаны в бинарном виде. Датасет состоит из двух файлов: с картинками и с метками классов.

В наборе данных наблюдается соответствие между предметами одежды и метками (смотреть таблицу 1). То есть конкретному предмету сопоставляется определённое число.

0	1	2	3	4	5	6	7	8	9
футболка	брюки	свитер	платье	пальто	туфли	рубашка	кроссовки	сумка	ботинки

Таблица 1: Метки предметов одежды

## 3 Этапы выполнения проекта

На каждом этапе записан фрагмент кода программы и его описание с объяснениями.

### 3.1 Подключение необходимых библиотек и модулей

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import utils
```

Библиотека `numpy` пригодится нам для удобной работы с массивами, а модуль `pyplot` из `matplotlib` необходим для отображения картинок и построения графиков.

Далее подключим нужные модули из Keras. Импортируем модуль для работы с набором данных Fashion MNIST. После чего подключаем модель `Sequential`, которая используется для представления нейронной сети, где слои идут друг за другом, и `Dense` – основной слой для нашей модели, необходимый для полносвязного соединения всех входных данных с выходным слоем. Также нам пригодятся утилиты Keras (`utils`) для приведения данных в нужный формат.

### 3.2 Загрузка обучающей и тестовой выборки

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

С помощью метода `fashion_mnist.load_data` Keras автоматически загружает набор данных и записывает результат в указанные переменные. Так, в `x_train` записываются изображения,

которые будут в дальнейшем использоваться для обучения, а в `y_train` – метки классов, указывающие на то, что конкретно изображено на каждой картинке. То же самое можно сказать про переменные `x_test` и `y_test`, использующиеся для работы с тестовой выборкой.

### 3.3 Создание списка с названиями предметов одежды

```
names = ['футболка', 'брюки', 'свитер', 'платье', 'пальто', 'туфли', 'рубашка', 'кроссовки', 'сумка', 'ботинки']
```

Мы создали список, где в качестве метки выступает индекс, а в качестве предмета одежды – элемент массива. Это было сделано исходя из существующего соответствия в нашем наборе данных (футболке соответствует метка 0, брюкам – метка 1 и так далее). Данное представление очень удобно для последующих этапов работы.

### 3.4 Отображение первых 25 картинок

```
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], cmap=plt.cm.binary)
    plt.xlabel(names[y_train[i]])
```

Сделаем это для понимания, с чем нам в принципе придётся иметь дело. Используя модуль `pyplot` библиотеки `matplotlib`, отобразим первые 25 изображений, и под каждым из них запишем, какой предмет одежды на нём находится. В результате выполнения кода мы получим следующее отображение (смотреть рис. 1):



Рис. 1: Примеры изображений

### 3.5 Изменение размерности данных в наборе

```
x_train = x_train.reshape(60000, 28*28)
x_test = x_test.reshape(10000, 28*28)
```

Преобразовываем картинки в плоские вектора. Меняем размерность изображений с помощью метода `reshape` (первый аргумент – количество изображений, второй аргумент – количество пикселей).

### 3.6 Стандартизация данных

```
x_train = x_train / 255
x_test = x_test / 255
```

Делим интенсивность каждого пикселя в изображении на 255. Так мы добьёмся того, что входные значения нейронной сети будут в диапазоне от 0 до 1. Это удобно для работы алгоритма оптимизации, необходимого для обучения нейронной сети.

### 3.7 Преобразование выходных значений в векторы по категориям

```
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)
```

Приводим правильные ответы к нужному виду. Так как нейронная сеть будет выдавать не один номер класса, а целых 10 значений на выходе, являющихся вероятностью того, что изображение принадлежит данному классу, необходимо привести наши метки к виду one hot encoding. То есть мы представим их в виде списка, где на месте элемента с индексом, равным значению метки, будет стоять 1, а на других местах – 0.

### 3.8 Формирование нейронной сети и вывод её структуры

```
model = Sequential()
model.add(Dense(256, input_dim=784, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(10, activation="softmax"))

print(model.summary())
```

С помощью Sequential создаём последовательную модель. Добавляем в неё полносвязные слои, пользуясь методом Dense. Во второй строке мы вписали следующее: 256 – количество нейронов, input\_dim=784 – количество входов в каждый нейрон ( $28 \times 28$ ), activation='relu' –

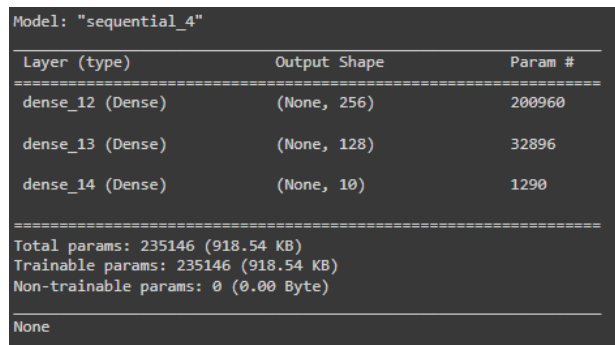
активационная функция, которая определяется по формуле:

$$f(x) = \max(0, x)$$

Аналогично добавляем следующий слой, состоящий из 128 нейронов, но здесь нам уже не потребуется вписывать значение `input_dim`. На выходном слое задействуем 10 нейронов и активационную функцию `softmax`. Для  $K$  объектов она выглядит следующим образом:

$$f(y_i) = \frac{e^{y_i}}{\sum_{j=1}^K e^{y_j}}$$

В четвертой строке фрагмента кода выводим структуру нейронной сети. В результате увидим следующее (смотреть рис. 2):



```
Model: "sequential_4"
Layer (type)                Output Shape              Param #
=====
dense_12 (Dense)             (None, 256)               200960
dense_13 (Dense)             (None, 128)               32896
dense_14 (Dense)             (None, 10)                1290
=====
Total params: 235146 (918.54 KB)
Trainable params: 235146 (918.54 KB)
Non-trainable params: 0 (0.00 Byte)
None
```

Рис. 2: Структура модели нейронной сети

Структура определена верно. Стоит уточнить, что в третьем столбце число параметров определяется как число связей, которые соединены с нейронами конкретного слоя, причём при подсчёте учитывается нейрон смещения `bias`. К примеру, для слоя `dense_12` имеем  $(784 + 1) \times 256 = 200960$  настраиваемых параметров. Здесь 784 – число входных нейронов, 1 добавляем, так как учитываем `bias`, а 256 – число нейронов слоя `Dense`. Как видим, всё сформировалось правильно.

### 3.9 Компиляция модели нейронной сети

```
model.compile(loss="categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
```



Запустим компиляцию модели. Здесь "categorical\_crossentropy" (категориальная кросс-энтропия) – функция ошибки, "SGD" (стохастический градиентный спуск) – оптимизатор, а последний аргумент "accuracy" (точность) – метрика качества.

### 3.10 Обучение нейронной сети

```
history = model.fit(x_train, y_train, batch_size=200, epochs=100, validation_split=0.2)
model.evaluate(x_test, y_test)
```

Обучим нашу модель с помощью метода `model.fit`. Укажем все необходимые параметры. Здесь `x_train`, `y_train` – обучающая выборка, `batch_size = 200` – размер мини-выборки, `softmax = 100` – количество эпох обучения, `validation_split=0.2` – размер проверочной выборки (20%).

Далее узнаем параметры `loss` и `accuracy`, получаемые на тестовой выборке (с помощью метода `model.evaluate`). Для нас они вполне приемлемы (см. рис. 3):

```
313/313 [=====] - 1s 3ms/step - loss: 0.3602 - accuracy: 0.8731
[0.3602445423603058, 0.8730999827384949]
```

Рис. 3: Параметры `loss` и `accuracy`

### 3.11 Визуализация точности и функции ошибки на графиках

```
plt.plot([i+1 for i in range(100)], history.history['accuracy'])
plt.plot([i+1 for i in range(100)], history.history['val_accuracy'])
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.title('Точность')
plt.legend(['accuracy', 'validation accuracy'])

plt.plot([i+1 for i in range(100)], history.history['loss'])
plt.plot([i+1 for i in range(100)], history.history['val_loss'])
plt.xlabel('epochs')
plt.ylabel('loss')
plt.title('Функция ошибки')
plt.legend(['loss', 'validation loss'])
```

На предыдущем шаге мы сохранили результаты обучения в `history`. Теперь, исходя из них,

строим графики для ошибки и точности на обучающей и проверочных выборках с помощью библиотеки matplotlib.

Получим следующие результаты (см. рис. 4 и 5):

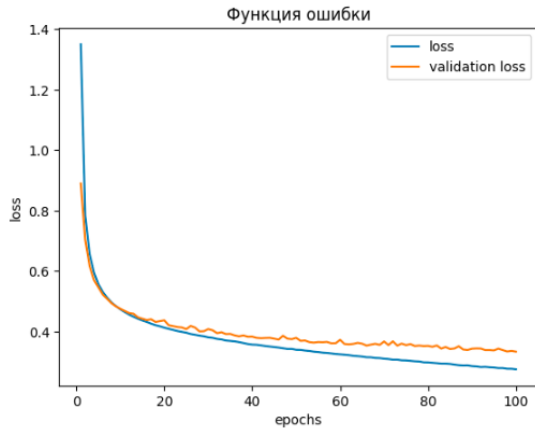


Рис. 4: График функции ошибки

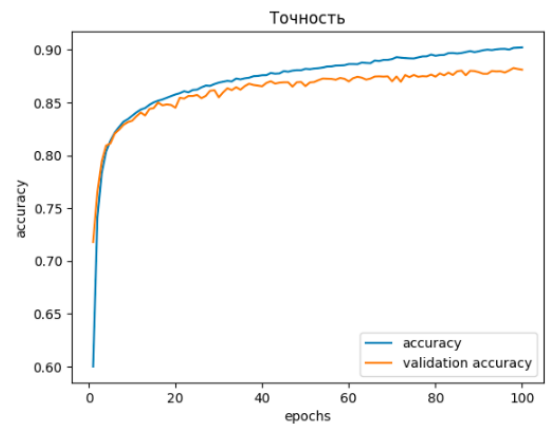


Рис. 5: График точности

Как мы видим, графики на каждой из выборок весьма схожи и причин для беспокойства нет.

### 3.12 Запуск распознавания набора данных

```
predictions = model.predict(x_test)
```

В данном случае в список predictions будут записаны предсказания для каждого из изображений, находящихся в x\_test.

### 3.13 Проверка результата для произвольного значения

```
n = np.random.randint(0, 10000)
plt.imshow(x_test[n].reshape(28, 28), cmap=plt.cm.binary)
plt.show()
print(predictions[n])
print('Предмет:', names[np.argmax(predictions[n])])
print('Правильный ответ:', names[np.argmax(y_test[n])])
```

Будем распознавать случайное изображение и посмотрим на результат. При одном из запусков получим следующее (см. рис. 6):

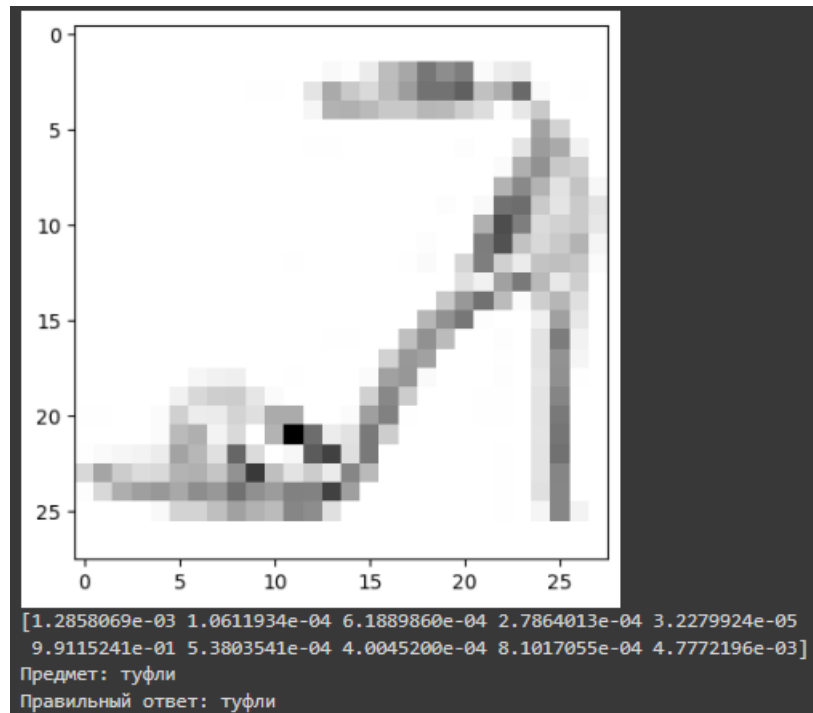


Рис. 6: Распознавание туфли

Как мы видим, модель правильно определила предмет одежды. Стоит обратить внимание на то, что в списке выходных значений нейронной сети максимальное из десяти чисел соответствует индексу 5, что и говорит нам о туфлях на изображении.

### 3.14 Некоторые случаи неправильной работы нейронной сети

```
n = np.random.randint(0, 10000)
plt.imshow(x_test[n].reshape(28, 28), cmap=plt.cm.binary)
plt.show()
print(predictions[n])
print('Предмет:', names[np.argmax(predictions[n])])
print('Правильный ответ:', names[np.argmax(y_test[n])])
```

Рассмотрим случаи, когда нейронная сеть выдаёт неправильные ответы среди первых 20 изображений в тестовой выборке. Таких случаев 2 (см. рис. 7 и 8):

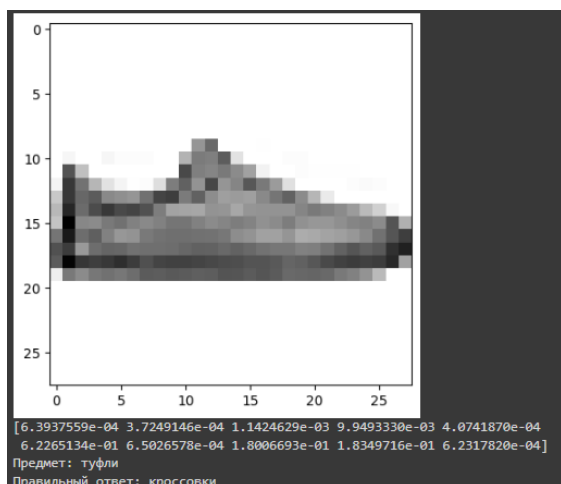


Рис. 7: Неправильное распознавание кроссовок

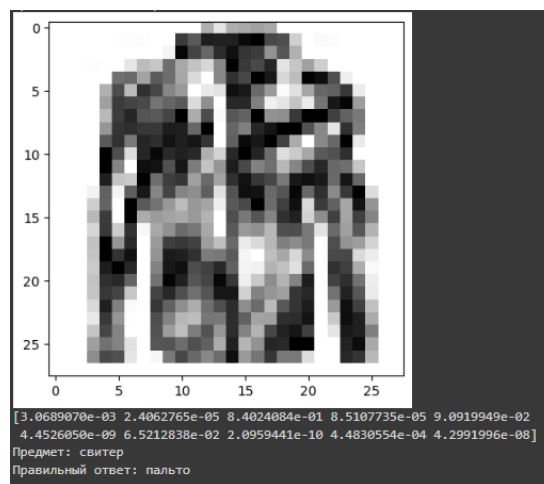


Рис. 8: Неправильное распознавание пальто

Можно заметить, что кроссовки действительно можно спутать с туфлями, как и пальто со свитером.

### 3.15 Распознавание изображения по загружаемой фотографии

#### 3.15.1 Описание программного кода

В начале импортируем необходимые модули:

```
from google.colab import files
from IPython.display import Image
from tensorflow.keras.preprocessing import image
```

Далее будем загружать изображение для распознавания. Загрузим картинку с названием “image.jpg”:

```
d = files.upload()
imag = "image.jpg"
```

Отобразим фотографию, убедившись, что картинка успешно загружена:

```
Image(imag, width=150, height=150)
```

Загрузим картинку в память с помощью метода `image.load_img`. В параметрах указываем

наше изображение (img), размер, к которому его приводим, а также то, что картинка должна быть в градациях серого:

```
our_image = image.load_img(img, target_size=(28, 28), color_mode = "grayscale")
```

Проведём предварительную обработку изображения. Сначала мы преобразуем его в массив. Далее делаем из картинки плоский вектор из 784 значений (так как размер изображения  $28 \times 28 = 784$  пикселя). Далее инвертируем изображение и нормализуем его:

```
x = image.img_to_array(our_image)
x = x.reshape(1, 784)
x = 255 - x
x /= 255
```

Запустим распознавание модели и узнаем предсказания:

```
prediction = model.predict(x)
prediction
```

Выведем результат в наглядном виде:

```
u = np.argmax(prediction)
print("Номер класса:", u)
print("Название класса:", names[u])
```

### 3.15.2 Примеры работы программы с загружаемыми изображениями

Попробуем распознать туфли. Загрузим изображение “image.jpg”, которое выглядит следующим образом (см. рис. 9):



Рис. 9: Изображение с туфлями

Оно успешно загрузится (см. рис. 10), и на мы получим верный результат (см. рис. 11).



Рис. 10: Отображение загруженной фотографии с туфлями

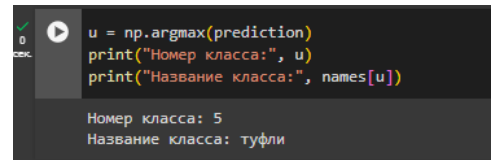


Рис. 11: Ответ в наглядном виде

Аналогичным образом попробуем распознать брюки (см. рис. 12):



Рис. 12: Изображение с брюками

Как и в предыдущий раз, нейронная сеть успешно распознает предмет одежды (см. рис. 13, 14).



Рис. 13: Отображение загруженной фотографии с брюками

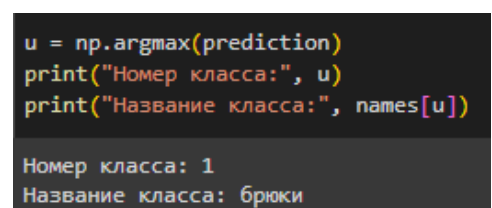


Рис. 14: Ответ в наглядном виде

## 4 Заключение

В ходе работы я смог достигнуть поставленной цели - создать простейшую нейронную сеть с использованием библиотеки Keras. Мне удалось обучить модель классификации, которая с высокой точностью может распознать предмет одежды на картинке. Кроме того, я подробно узнал принципы работы нейронной сети, научился визуализировать данные, необходимые для работы с моделью. В дальнейшем это позволит мне реализовывать более сложные проекты.

## Список литературы

1. Fashion MNIST - Википедия. — URL: [https://en.wikipedia.org/w/index.php?title=Fashion\\_MNIST&oldid=1166741889](https://en.wikipedia.org/w/index.php?title=Fashion_MNIST&oldid=1166741889).
2. *Балакирев С.* Создание первого проекта на Python. — 2020. — URL: <https://www.youtube.com/playlist?list=PLA0M1Bcd0w8yv0XGiF1wjerjSZVSrYbjh>.
3. Документация по Overleaf с официального сайта. — URL: <https://ru.overleaf.com/learn>.
4. Курс: Введение в LaTeX (Introduction to LaTeX). — URL: <https://edu.hse.ru/course/view.php?id=136849>.
5. Платформа: NeuroHive - Нейронная сеть. — URL: <https://neurohive.io/ru>.
6. *Созыкин А.* Изучение Python для начинающих. — 2016-2019. — URL: [https://www.youtube.com/playlist?list=PLtPJ9lKvJ4oiz9aaL\\_xcZd-x0qd8G0VN\\_](https://www.youtube.com/playlist?list=PLtPJ9lKvJ4oiz9aaL_xcZd-x0qd8G0VN_).