

МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ
КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
Ордена Трудового Красного Знамени
Федеральное государственное бюджетное образовательное учреждение высшего
образования
«Московский технический университет связи и информатики»

КУРСОВАЯ РАБОТА
по дисциплине «Системы искусственного интеллекта»
на тему:
Решение задачи классификации изображений с применением методов искусственного
интеллекта

Выполнил:
студент Туркова Елена Дмитриевна
группа БВТ2103
Проверил:

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1 Методы и алгоритмы распознавания объектов	4
1.1 Классификация методов распознавания	4
1.2 Обзор существующих алгоритмов	5
1.3 Сравнительный анализ методов	6
2 Сбор и подготовка набора данных	8
2.1 Источники открытых наборов данных	8
2.2 Процесс сбора изображений	9
3 Разработка и обучение модели	11
3.1 Загрузка и подготовка данных	11
3.2 Создание и обучение модели	14
3.3 Итоги обучения модели	16
3.4 Альтернативная архитектура нейронной сети	18
ЗАКЛЮЧЕНИЕ	21
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	22
ПРИЛОЖЕНИЕ А. Исходный код модели	23
ПРИЛОЖЕНИЕ Б. Дополнительные материалы	27

ВВЕДЕНИЕ

В наше время искусственный интеллект и методы машинного обучения стали важнейшим элементом в различных отраслях науки и техники. Одной из наиболее актуальных задач является распознавание объектов на изображениях, которое широко используется в таких сферах, как компьютерное зрение, образование, развлечение и медиа. В данной курсовой работе рассматривается задача распознавания различных категорий одежды на открытых наборах данных.

Актуальность выбора темы обусловлена растущим интересом к моде и стилю, а также необходимостью автоматизации процессов, связанных с классификацией и рекомендацией одежды. В современном мире, где онлайн-торговля и цифровые технологии становятся все более популярными, эффективное распознавание категорий одежды имеет важное значение для улучшения пользовательского опыта и повышения эффективности бизнес-процессов.

В данной работе будут рассмотрены различные алгоритмы и методы, используемые для распознавания изображений, такие как сверточные нейронные сети¹, алгоритмы машинного обучения и методы предобработки данных. *Целью работы* является разработка модели, способной эффективно распознавать и классифицировать различные категории одежды на основе открытых наборов данных. Для достижения этой цели будут поставлены следующие *задачи*:

- изучить существующие методы и алгоритмы распознавания объектов на изображениях;
- подготовить набор данных с изображениями одежды;
- разработать и обучить модель для распознавания изображений;
- оценить эффективность разработанной модели и провести анализ полученных результатов по необходимым метрикам;
- сформулировать выводы и рекомендации по выбранной теме.

Таким образом, данная курсовая работа направлена на исследование и практическое применение методов распознавания изображений в контексте видов одежды, что может способствовать дальнейшему развитию технологий в этой области.

¹ Сверточные нейронные сети (CNN) — это класс глубоких нейронных сетей, специально разработанных для обработки данных с сетчатой структурой, таких как изображения.

1 Методы и алгоритмы распознавания объектов

Теория распознавания образа представляет собой раздел машинного обучения, который позволяет осуществлять классификацию и идентификацию предметов, явлений, процессов, сигналов и ситуаций. Термин «машинное зрение» можно связывать с задачей интерпретации сцены наблюдения по ее двумерным проекциям, а также практическое использование этих результатов [1].

Мы часто не осознаем, как в повседневной жизни взаимодействуем с технологиями, основанными на «компьютерном зрении». Примеры включают считыватели штрих-кодов, которые автономно распознают закодированную информацию, и камеры, анализирующие автомобильные номера для контроля скорости на дорогах, и многие другие приложения.

Для решения задач распознавания объектов в реальных условиях необходимо учитывать биологические и психологические аспекты человеческого зрения, а также определить этапы формирования опыта запоминания и классификации объектов. Рассмотрим ситуацию, когда человек впервые видит объект, например, самолёт. В его сознании формируется описательный образ этого предмета, а также его характеристики и свойства. При повторной встрече мозг, опираясь на накопленный опыт и воспоминания, восстанавливает образ и классифицирует объект.

Но как происходит распознавание у компьютера? Поскольку компьютер не обладает опытом, в его программу закладываются определенные алгоритмы и правила, которые позволяют ему выдавать ответ. В первую очередь определяются массивы данных, описывающие различные факторы, такие как условия окружающей среды, освещение и так далее. Эти данные служат примерами, которые затем анализируются и преобразуются в формулы или математические модели.

1.1 Классификация методов распознавания

На сегодняшний день изображение рассматривается в согласии с модульной парадигмой, предложенной Д. Марром². В связи с этим обработка изображения осуществляется на основе нескольких уровней восходящей информационной линии, включая от «иконического» до символического представления объектов [2].

Стоит обратить внимание на само изображение, которое в видимом поле представлено как функция распределения яркости или цвета на двумерной плоскости, обозначаемой как $f(x, y)$, где x и y — декартовы координаты, описывающие плоскость изображения. Это изображение состоит из дискретных цветных прямоугольников, организованных в регулярную прямоугольную матрицу. С математической точки зрения цифровое изображение можно представить как двумерную матрицу $Im[x, y]$ размером

² Марр, Д. (1982). Vision. San Francisco: W.H. Freeman and Company.

$\text{DimX} \times \text{DimY}$, где x — целое число от 0 до $\text{DimX}-1$, указывающее номер элемента в строке матрицы, а y — целое число от 0 до $\text{DimY}-1$, обозначающее номер строки, в которой находится данный элемент. Каждый элемент цифрового изображения, имеющий скалярное целочисленное значение, соответствующее значению функции распределения яркости, называется пикселем.

Пиксель изображения кодируется тремя величинами $\langle x, y, I \rangle$, где первые две определяют геометрическое положение на плоскости, а третья величина I указывает на яркость и интенсивность пикселя. Такое представление упрощает задачу для разработчика, позволяя понижать порядок задачи и, соответственно, значительно экономить время. Следует также отметить, что яркостная составляющая I описывает одномерный массив гистограммы, отражающий частоту встречаемости пикселей с одинаковой яркостью, в то время как другие параметры представлены двумерным массивом информации. Это позволяет сводить двумерные задачи к одномерным.

Кроме того, изображение можно рассматривать с точки зрения его цветовых характеристик. Чаще всего используются изображения, закодированные в RGB-пространстве, где по осям располагаются цвета: красный, зеленый и синий (x, y, z соответственно), а интенсивность варьируется от 0 до 255. Каждый цветовой сигнал кодируется 8 битами информации. Существует также модель HSV (Hue, Saturation, Value). Hue — это характеристика, показывающая цветовой тон, Saturation — насыщенность цвета, отражающая «чистоту» цвета, а Value — ось, определяющая количественную характеристику светового потока. Эта модель является нелинейной по отношению к RGB и более близка к восприятию цвета человеком. Этапы обработки изображений включают преобразование, сегментацию, выделение геометрической структуры и определение структуры и семантики.

1.2 Обзор существующих алгоритмов

В настоящее время существует множество алгоритмов, методов и математических формализмов, применяемых при анализе изображений.

Метод *гистограммных преобразований* используется для улучшения контраста изображения путем изменения его гистограммы. Он позволяет перераспределить яркость пикселей, чтобы сделать изображение более четким. Гистограммные преобразования отличаются от других методов тем, что они работают с распределением яркости, а не с самими пикселями.

Анализ проекций включает в себя проекцию изображения на оси, что позволяет выявить основные характеристики, такие как форма и расположение объектов. Он

отличается от других методов тем, что фокусируется на глобальных свойствах изображения, а не на локальных деталях.

Линейная фильтрация использует линейные операции для сглаживания или выделения деталей изображения, тогда как *нелинейная фильтрация* применяет более сложные функции, чтобы лучше справляться с шумами и артефактами. *Нелинейные* методы, такие как *медианная фильтрация*, часто более эффективны в условиях сильного шума.

Яркостная сегментация основывается на различиях в яркости пикселей для разделения объектов, тогда как *текстурная сегментация* использует текстурные характеристики, такие как однородность или контраст. Эти методы отличаются по подходу к анализу изображения: первый фокусируется на цвете, а второй — на текстуре.

Методы *выделения контуров*, такие как оператор Собеля и оператор Кэнни, используются для обнаружения границ объектов в изображении. Они работают, анализируя изменения яркости между соседними пикселями, и отличаются от других методов тем, что акцентируют внимание на резких переходах.

Алгоритмы кластеризации, такие как K-средних, группируют пиксели изображения на основе их характеристик, таких как цвет или текстура. Кластеризация позволяет выделить однородные области, в то время как другие методы могут не учитывать подобные группы.

Методы *машинного обучения и глубокого обучения* используют алгоритмы, которые обучаются на больших объемах данных для распознавания паттернов и объектов в изображениях. Они отличаются от традиционных методов тем, что могут адаптироваться и улучшаться с течением времени, основываясь на новых данных.

Алгоритмы детекции и распознавания объектов, такие как YOLO и SSD, используются для идентификации и классификации объектов в изображениях. Они отличаются от других методов тем, что могут одновременно обнаруживать несколько объектов и определять их классы.

1.3 Сравнительный анализ методов

Существует множество других методов, которые могут использоваться в различных задачах анализа изображений. Важно провести сравнительный анализ методов, чтобы определить, какой из них наиболее подходит для конкретной задачи.

В контексте разработки модели, способной эффективно распознавать и классифицировать различные категории одежды на основе открытых наборов данных, можно рассмотреть использование алгоритмов глубокого обучения, таких как сверточные нейронные сети, которые хорошо зарекомендовали себя в задачах классификации

изображений. Эти алгоритмы могут быть обучены на специализированных наборах данных, что позволит достичь высокой точности в распознавании и классификации.

Свёрточная нейронная сеть – одна из моделей ИНС (искусственных нейронных сетей), была предложена в 1988 году французским учёным в области информатики Яном Лекуном. Целью разработки данного послужило увеличение эффективности распознавания образов, которое находится в составе технологий глубокого обучения. Идея, заключённая в свёрточных нейронных сетях, состоит в попеременном использовании субдискретизирующих и свёрточных слоёв [3]. Сеть состоит из множества слоёв, начиная с входного слоя, который принимает изображение. Далее сигнал проходит через последовательность свёрточных слоёв, что позволяет эффективно извлекать и группировать карты признаков. На каждом последующем слое размер карты признаков уменьшается, в то время как количество каналов увеличивается. Это свойство способствует улучшению способности сети к распознаванию сложных признаков. В дополнение к свёрточным слоям, на выходе сети располагаются слои полносвязной нейронной сети, которые принимают на вход результаты работы свёрточных слоёв (конечные карты признаков), в соответствии с рисунком 1.1.

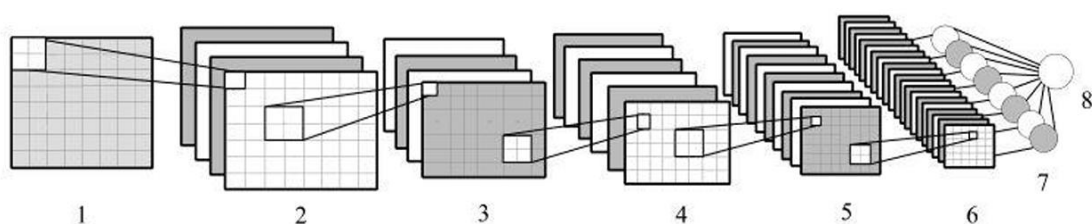


Рисунок 1.1 - Архитектура свёрточной нейронной сети: 1 – вход, 2,4,6 – свёрточные слои, 3,5 – подвыборочные слои, 7 – слои из обычных нейронов, 8 – выход

После проведения сравнительного анализа методов распознавания изображений становится очевидным, что выбор подходящего алгоритма, такого как свёрточные нейронные сети, является важным этапом в разработке эффективной модели. Для достижения высоких результатов в распознавании различных категорий одежды необходимо также обратить внимание на качество и разнообразие используемых данных.

2 Сбор и подготовка набора данных

Сбор и подготовка набора данных играют ключевую роль в обучении модели, так как именно от них зависит способность алгоритма обобщать информацию и правильно классифицировать объекты. В этом контексте важно рассмотреть доступные источники открытых наборов данных, которые могут быть использованы для обучения модели.

Набор данных (НД) – это совокупность данных, прошедших предварительную подготовку (обработку) в соответствии с требованиями законодательства Российской Федерации об информации, информационных технологиях и о защите информации и необходимых для разработки программного обеспечения на основе искусственного интеллекта [4].

Разметка данных – этап обработки структурированных и неструктурированных данных, в процессе которого данным (в том числе текстовым документам, фото- и видеоизображениям) присваиваются идентификаторы, отражающие тип данных (классификация данных), и (или) осуществляется интерпретация данных для решения конкретной задачи, в том числе с использованием методов машинного обучения [4].

Размер набора данных (математически – размер выборки) и баланс классов определяются исходя из целей и задач проводимого исследования и требований технического задания на проведение исследований, а также с учетом требований заказчика.

2.1 Источники открытых наборов данных

Сбор и использование открытых наборов данных являются важными этапами в задачах распознавания изображений. Они предоставляют исследователям и разработчикам возможность обучать свои алгоритмы на разнообразных и репрезентативных данных, что способствует повышению точности и надежности моделей. Одним из наиболее известных и широко используемых наборов данных для задач классификации изображений является Fashion-MNIST.

Fashion-MNIST представляет собой набор данных, содержащий 70,000 изображений одежды, разделенных на 10 категорий, таких как футболки, брюки, свитеры, платья, пальто, сандалии, рубашки, кроссовки, сумки и ботинки. Каждая категория включает в себя 7,000 изображений, что обеспечивает достаточное количество данных для обучения и тестирования моделей. Набор данных был создан как замена классическому набору MNIST, который использовался для распознавания рукописных цифр. Fashion-MNIST предлагает более сложные и разнообразные задачи, что делает его идеальным для обучения моделей, работающих с изображениями одежды.

Использование Fashion-MNIST в данной работе обусловлено его доступностью и простотой в использовании, а также тем, что он предоставляет возможность исследовать различные архитектуры нейронных сетей и методы предобработки данных. Набор данных уже широко используется в научных исследованиях и образовательных проектах, что позволяет легко находить примеры и сравнивать результаты. Существуют и другие источники открытых наборов данных, которые могут быть полезны.

ImageNet является одним из крупнейших наборов данных для задач классификации изображений, содержащий миллионы аннотированных изображений, включая изображения различных объектов и одежды.

Open Images Dataset от Google содержит более 9 миллионов изображений с аннотациями, включая разметку объектов и их классы. Он подходит для задач распознавания и локализации объектов.

COCO (Common Objects in Context) - набор данных, содержащий изображения с аннотациями для объектов, сцен и их контекста. COCO подходит для задач сегментации и распознавания объектов.

Google Dataset Search – это поисковая система, которая помогает находить открытые наборы данных по различным темам и категориям. Она может быть полезной для поиска специфических наборов данных.

2.2 Процесс сбора изображений

В данном исследовании для сбора данных был выбран готовый набор данных Fashion-MNIST, что значительно упростило и ускорило процесс. В данном наборе данных представлены 10 типов вещей. Примеры изображений одежды из датасета можно увидеть на рисунке 2.1 [5].



Рисунок 2.1 – Пример изображений из датасета Fashion-MNIST

Изображения, которые содержатся в датасете, будут распределены на две группы. В первую группу будет включено 60 000 изображений, которые будут использованы для обучения нейросети. Оставшиеся изображения будут использованы для проверки точности распознавания.

Каждый из массивов, представляет собой набор изображений, представленных в виде значений пикселей, которые находятся в диапазоне от 0 до 255. Каждая категория одежды имеет соответствующую метку, которая состоит из массива целых чисел в диапазоне от 0 до 9. Так как в наборе данных не включены имена меток их необходимо прописать вручную. Имена меток отображены в таблице 1.

Таблица 1 – Наименование меток датасета.

Метка	Класс
0	T-shirt (Футболка)
1	Trouser (Брюки)
2	Pullover (Свитер)
3	Dress (Платье)
4	Coat (Пальто)
5	Sandal (Сандали)
6	Shirt (Рубашка)
7	Sneaker (Кроссовки)
8	Bag (Сумка)
9	Ankle boot (Ботильоны)

3 Разработка и обучение модели

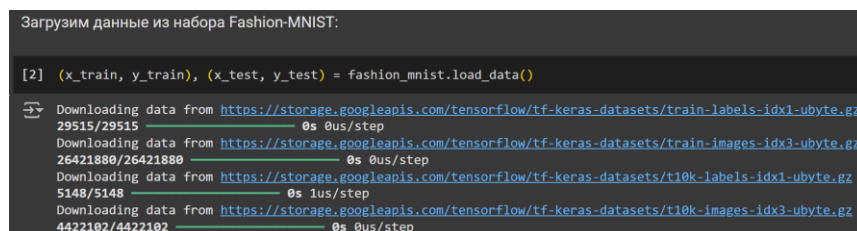
В данной курсовой работе рассматривается процесс разработки нейронной сети для распознавания изображений с использованием библиотеки TensorFlow. Мы будем использовать набор данных Fashion-MNIST, который содержит изображения одежды и аксессуаров, и обучим модель для классификации этих изображений. В процессе работы мы подробно объясним каждый фрагмент кода и ключевые понятия, такие как сбалансированность классов, нормализация данных и аугментация изображений.

Произведем установку необходимых библиотек. TensorFlow является основным инструментом для разработки и обучения нейронных сетей. TensorFlow предоставляет мощные инструменты для работы с данными и построения моделей машинного обучения.

- numpy: библиотека для работы с многомерными массивами и матрицами, а также для выполнения математических операций;
- matplotlib.pyplot: библиотека для визуализации данных, которая позволяет строить графики и диаграммы;
- tensorflow: основная библиотека для работы с нейронными сетями;
- layers и models: модули из Keras (входит в TensorFlow), которые позволяют создавать и настраивать слои нейронной сети;
- fashion_mnist: набор данных, содержащий изображения одежды, который мы будем использовать для обучения;
- ImageDataGenerator: класс для аугментации изображений, который позволяет увеличивать объем данных за счет применения различных трансформаций;
- classification_report: функция для оценки качества модели, которая выводит метрики, такие как точность, полнота и F1-мера.

3.1 Загрузка и подготовка данных

Далее загружаем набор данных Fashion-MNIST. Данные разделяются на обучающую выборку (x_{train} , y_{train}) и тестовую выборку (x_{test} , y_{test}). Код представлен на рисунке 3.1.1.



```
Загрузим данные из набора Fashion-MNIST:

[2] (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 — 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 — 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 — 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422182/4422182 — 0s 0us/step
```

Рисунок 3.1.1 - x_{train} и x_{test} содержат изображения, а y_{train} и y_{test} — соответствующие метки классов.

Визуализируем распределение классов. Создадим функцию `plot_class_distribution`, представленную на рисунке 3.1.2.

```
def plot_class_distribution(labels):
    # Подсчет количества изображений в каждом классе
    class_counts = np.bincount(labels)

    # Нормализация для получения процентного соотношения
    class_percentages = class_counts / len(labels) * 100

    plt.figure(figsize=(10, 5))
    plt.bar(range(10), class_percentages, color='skyblue', edgecolor='black')
    plt.xticks(range(10), ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                           'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'])
    plt.title('Class Distribution in Fashion-MNIST')
    plt.xlabel('Classes')
    plt.ylabel('Percentage of Images (%)')
    plt.ylim(0, 100) # Установка пределов по оси Y от 0 до 100%
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Вызов функции для отображения распределения классов
plot_class_distribution(y_train)
```

Рисунок 3.1.2 – Код для визуализации сбалансированности³ классов в обучающей выборке.

На графике, изображенном на рисунке 3.1.3 можно заметить, что данные распределены по высоте колонок и одинаковы. Это значит, что данные уже нормализованы.

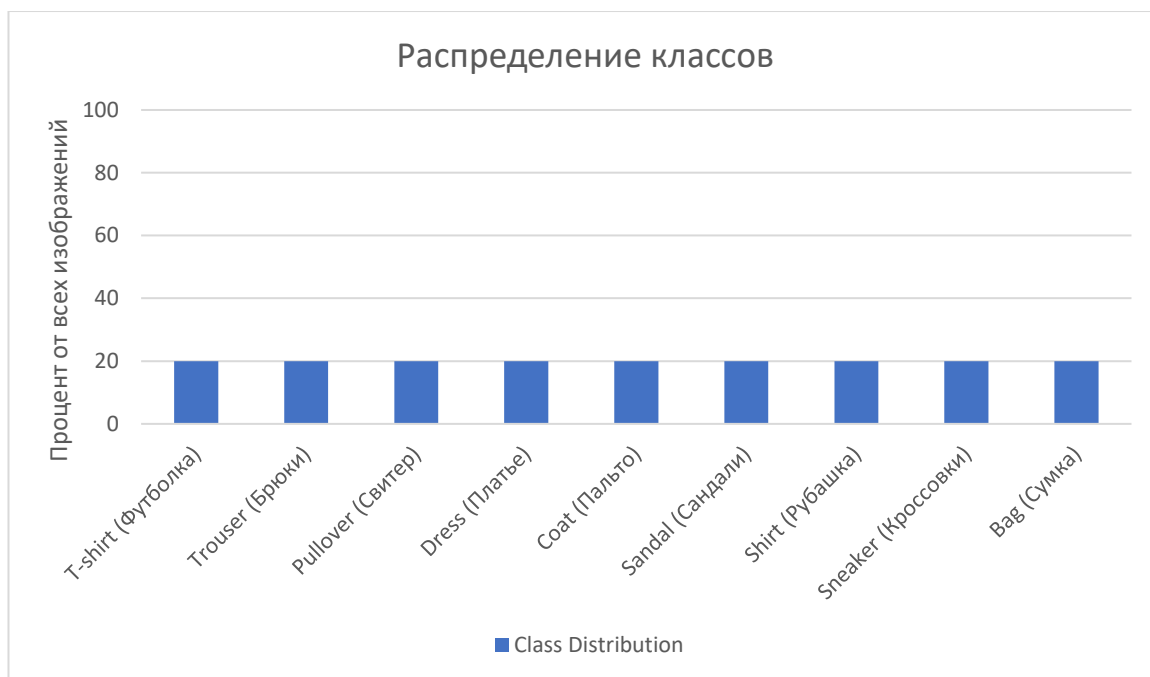


Рисунок 3.1.3 – График отображения распределения классов

Следующим этапом следовала нормализация данных. Это процесс приведения значений пикселей изображений к диапазону от 0 до 1. В данном случае мы делим значения пикселей (которые изначально находятся в диапазоне от 0 до 255) на 255.0. Это важно для последующего обучения, поскольку нормализованные данные помогают ускорить процесс обучения и улучшить сходимость модели. Кроме того, снижается

³ Сбалансированность классов — это характеристика распределения классов в наборе данных, при которой количество примеров для каждой категории примерно одинаково. Сбалансированные данные позволяют модели обучаться эффективно, избегая предвзятости в сторону более представленных классов.

вероятность возникновения проблем с градиентами, таких как исчезновение или взрыв градиентов. Код выполнен в соответствии с рисунком 3.1.4.

```
Нормализация данных для улучшения сходимости модели:  
  
[6] x_train = x_train.astype('float32') / 255.0  
    x_test = x_test.astype('float32') / 255.0
```

Рисунок 3.1.4 – Нормализация данных

Функция `np.expand_dims` добавляет новую ось к массиву, что позволяет изменить размерность данных. В данном случае мы добавляем ось для каналов (цветовых каналов), так как Fashion-MNIST содержит черно-белые изображения, в соответствии с рисунком 3.1.5.

```
x_train = np.expand_dims(x_train, axis=-1)  
x_test = np.expand_dims(x_test, axis=-1)
```

Рисунок 3.1.5 - Изменение формы данных для соответствия входным требованиям создаваемой модели

После этой операции `x_train` и `x_test` будут иметь форму (количество изображений, 28, 28, 1), где 1 — это количество цветовых каналов (в данном случае 1, так как изображения черно-белые).

Аугментация изображений — это метод увеличения объема обучающей выборки путем применения различных трансформаций к исходным изображениям. Это помогает улучшить обобщающую способность модели. В данном случае мы используем следующие параметры:

- `rotation_range=20`: случайное вращение изображений на угол до 20 градусов;
- `width_shift_range=0.2`: случайное смещение изображения по ширине до 20% от ширины;
- `height_shift_range=0.2`: случайное смещение изображения по высоте до 20% от высоты;
- `shear_range=0.2`: применение сдвига (`shear`) до 20%;
- `zoom_range=0.2`: случайное увеличение изображения до 20%;
- `horizontal_flip=True`: случайное горизонтальное отражение изображений;
- `fill_mode='nearest'`: метод заполнения пикселей, которые могут появиться после трансформации.

Метод `fit` подготавливает данные для аугментации. Он вычисляет необходимые параметры (например, средние значения и стандартные отклонения) на основе обучающей выборки. Это необходимо для корректного применения аугментации к изображениям. Далее переходим к созданию модели.

3.2 Создание и обучение модели

Мы создаем последовательную модель нейронной сети с использованием класса Sequential⁴. Модель состоит из следующих слоев:

- а. Conv2D: Сверточный слой, который применяет 32 фильтра размером 3x3 к входным данным. Активация relu (Rectified Linear Unit) помогает в обучении, так как она позволяет избежать проблемы исчезающих градиентов.
- б. MaxPooling2D: Слой подвыборки, который уменьшает размерность выходных данных, сохраняя наиболее важные признаки. В данном случае используется размер подвыборки 2x2.
- в. Второй и третий сверточные слои аналогичны первому, но с 64 фильтрами.
- г. Flatten: Преобразует многомерный массив в одномерный, чтобы подготовить данные для полносвязного слоя. Это необходимо, так как полносвязные слои ожидают входные данные в виде одномерного вектора.
- д. Dense(64, activation='relu'): Полносвязный слой с 64 нейронами и активацией ReLU. Этот слой позволяет модели учиться на более сложных признаках, извлеченных из предыдущих слоев.
- е. Dense(10, activation='softmax'): Последний полносвязный слой с 10 нейронами (по количеству классов в наборе данных Fashion-MNIST) и активацией softmax. Этот слой преобразует выходные значения в вероятности для каждого класса, что позволяет интерпретировать результаты как вероятности принадлежности к каждому из классов.

Создание модели представлено на рисунке 3.2.1.

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax') # 10 классов для классификации
])
```

Рисунок 3.2.1 – Создание модели сверточной нейронной сети

Метод compile настраивает модель для обучения. Мы указываем:

⁴ Класс Sequential — это класс в Keras для создания нейронных сетей, позволяющий добавлять слои последовательно, что упрощает построение простых моделей.

- `optimizer='adam'`: оптимизатор Adam, который адаптивно изменяет скорость обучения и хорошо работает в большинстве случаев;
- `loss='sparse_categorical_crossentropy'`: функция потерь для многоклассовой классификации⁵. Мы используем `sparse_categorical_crossentropy`, так как метки классов представлены в виде целых чисел, а не в виде one-hot кодирования;
- `metrics=['accuracy']`: мы хотим отслеживать точность модели во время обучения и тестирования.

Код представлен на рисунке 3.2.2.

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

Рисунок 3.2.2 – Компиляция модели

Метод `fit` запускает процесс обучения модели. Для успешного обучения передаем следующие параметры:

- `datagen.flow(x_train, y_train, batch_size=32)`: Генератор данных, который будет подавать аугментированные изображения в модель. Параметр `batch_size=32` указывает, что модель будет обрабатывать 32 изображения за один шаг.
- `epochs=20`: Количество эпох (полных проходов по обучающей выборке). В данном случае мы обучаем модель в течение 20 эпох.
- `validation_data=(x_test, y_test)`: Тестовая выборка, на которой будет оцениваться модель после каждой эпохи. Это позволяет отслеживать, как модель обобщает на новых данных.

Код выполнен в соответствии с рисунком 3.2.3.

```
history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
                   epochs=20,
                   validation_data=(x_test, y_test))
```

Рисунок 3.2.3 - Обучение модели с использованием аугментации данных

⁵ Многоклассовая классификация — это задача машинного обучения, в которой модель обучается распознавать и классифицировать объекты на более чем два класса. В отличие от бинарной классификации, где есть только два возможных класса, многоклассовая классификация требует, чтобы модель могла различать и предсказывать несколько категорий на основе входных данных.

3.3 Итоги обучения модели

Проанализируем, как происходило наше обучение. На первой эпохе точность на обучающем наборе составила 56.95%, что указывает на начальную стадию обучения. К концу 20-й эпохи точность увеличилась до 85.50%. Это свидетельствует о том, что модель успешно обучается и улучшает свои предсказания. Начальные потери на первой эпохе составили 1.1637, что является довольно высоким значением. К 20-й эпохе потери снизились до 0.3937, что указывает на улучшение качества модели и ее способности правильно классифицировать изображения.

Валидационная точность на первой эпохе составила 75.33%, что уже выше, чем обучающая точность, что может указывать на хорошую обобщающую способность модели. Валидационная точность достигла 87.20% к концу обучения, что подтверждает, что модель не только обучается на тренировочных данных, но и хорошо обобщает на новых данных.

В процессе обучения наблюдается положительная корреляция между уменьшением потерь и увеличением точности как на обучающем, так и на валидационном наборе. Это говорит о том, что модель обучается эффективно. В последние эпохи наблюдается небольшое колебание в валидационной точности и потерях, что может указывать на приближение к плато. Это может быть признаком того, что модель достигла своего предела в обучении, и дальнейшее обучение может не привести к значительным улучшениям.

После завершения обучения модели, мы можем оценить ее производительность на тестовой выборке и визуализировать результаты в соответствии с рисунком 3.3.1. Для оценки модели можно использовать метод `evaluate`, который возвращает значение функции потерь и метрики, указанные при компиляции модели.

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_acc:.4f}')

313/313 - 3s - 10ms/step - accuracy: 0.8720 - loss: 0.3570
Test accuracy: 0.8720
```

Рисунок 3.3.1 – Производительность модели на тестовой выборке

Результаты тестирования показывают, что модель достигла хороших результатов в задаче многоклассовой классификации на наборе данных Fashion-MNIST. Тестовая точность в 87.20% и низкие потери свидетельствуют о том, что модель успешно обучена и

способна обобщать на новых данных. Это делает ее подходящей для применения в реальных задачах, связанных с распознаванием и классификацией изображений одежды.

Рассмотрим результаты нашего обучения на графиках. Результаты представлены на рисунке 3.3.2.

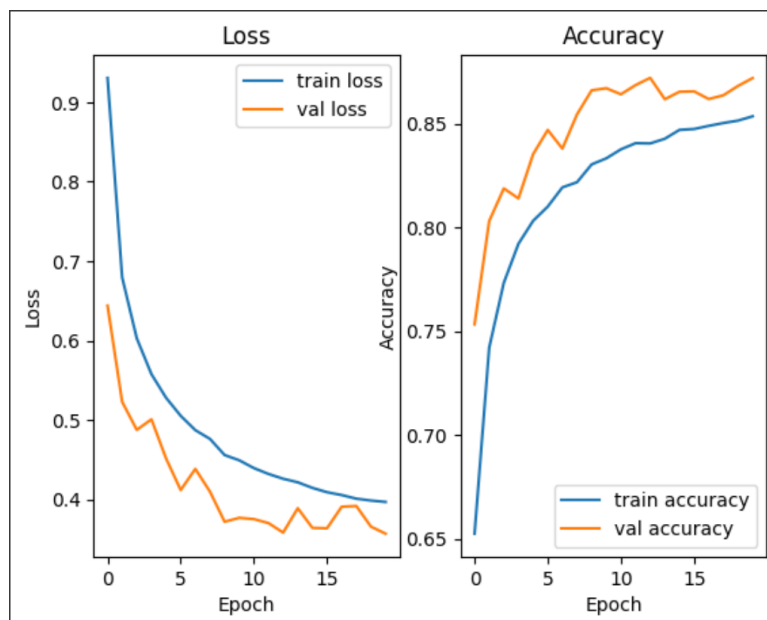


Рисунок 3.3.2 – Метрики обучения модели

На графике отображаются значения потерь для обучающей выборки (train loss) и валидационной выборки (val loss) в зависимости от эпохи. Ожидается, что значения потерь будут уменьшаться по мере обучения модели. Если график потерь для валидационной выборки начинает расти, это может указывать на переобучение модели. В нашем случае график потерь в целом уменьшается, что значит, что модель не столкнулась с переобучением.

На графике показаны значения точности для обучающей (train accuracy) и валидационной выборки (val accuracy) в зависимости от эпохи. Увеличение точности на обоих графиках указывает на успешное обучение модели. Если точность на валидационной выборке перестает расти или начинает снижаться, это также может быть признаком переобучения. В нашем случае график точности показывает положительные результаты, показывая улучшения.

Далее опишем функцию для визуализации предсказаний. Функция `plot_predictions` отображает изображения из тестового набора, а также истинные метки и предсказания модели. Для каждого изображения показывается: Истинная метка (True): Класс, к которому на самом деле принадлежит изображение. Предсказанная метка (Pred): Класс,

который модель предсказала для данного изображения. Визуализация позволяет наглядно оценить, насколько хорошо модель справляется с задачей классификации, она представлена на рисунке 3.3.3.

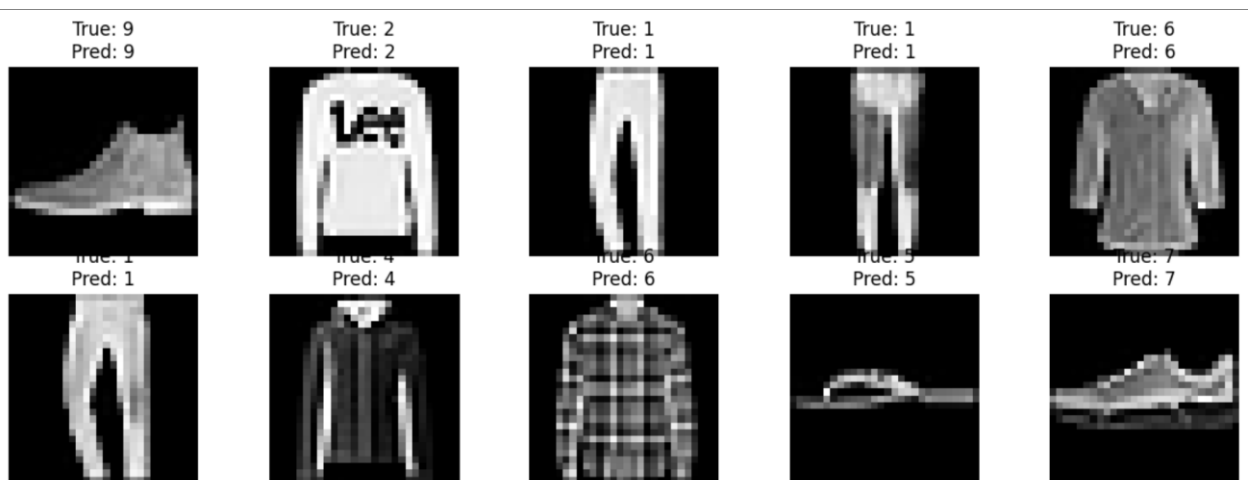


Рисунок 3.3.3 – Визуализация предсказаний

Как видим, модель правильно классифицировала все изображения, распределив их по классам. Это значит, что модель обучена успешно и цель выполнения работы была достигнута.

3.4 Альтернативная архитектура нейронной сети

Рассмотрим альтернативную архитектуру нейронной сети, которая была разработана для улучшения результатов распознавания изображений по сравнению с предыдущей моделью. Мы обсудим изменения, внесенные в архитектуру, и сравним характеристики обеих моделей. Код представлен на рисунке 3.4.1.

```
model2 = models.Sequential([
    layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5), # Дропаут для борьбы с переобучением
    layers.Dense(10, activation='softmax')
])

# Компиляция и обучение второй модели
model2.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9),
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

history2 = model2.fit(datagen.flow(x_train, y_train, batch_size=32),
                     epochs=20,
                     validation_data=(x_test, y_test))

# Оценка второй модели
test_loss2, test_acc2 = model2.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy of model 2: {test_acc2:.4f}')
```

Рисунок 3.4.1 – 2 модель

Изменения по сравнению с предыдущей моделью.

Увеличение количества фильтров. В первой сверточной сети использовалось 32 фильтра, в то время как во второй модели их количество увеличено до 64 и 128. Это позволяет модели извлекать более сложные и детализированные признаки из изображений.

Добавление слоя Dropout. В новой архитектуре добавлен слой Dropout с вероятностью 0.5. Этот слой помогает предотвратить переобучение, случайным образом отключая 50% нейронов во время обучения. Это способствует улучшению обобщающей способности модели.

Изменение оптимизатора. Для второй модели был выбран оптимизатор SGD (Stochastic Gradient Descent) с параметрами `learning_rate=0.01` и `momentum=0.9`. Это может помочь в более стабильном и быстром обучении по сравнению с оптимизатором Adam, который использовался в первой модели.

Графики потерь и точности представлены на рисунке 3.4.2.

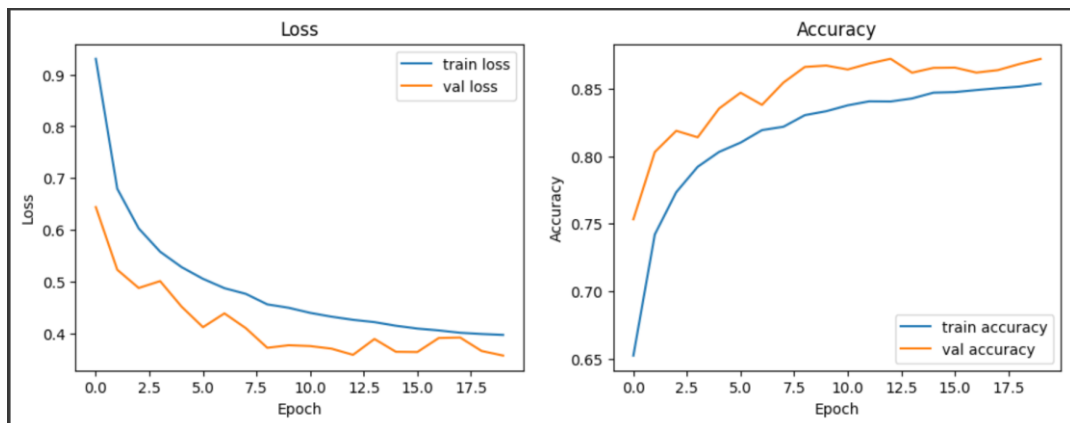


Рисунок 3.4.2 – Визуализация графиков потерь и точности

Точность на тренировочных данных постепенно увеличивается, что говорит о том, что модель учится на данных. Потеря на тренировочных и валидационных данных также снижается, что является положительным знаком. Валидационная точность в целом растет, что указывает на то, что модель не переобучается, по крайней мере, на протяжении первых 20 эпох. Модель демонстрирует хорошую производительность с точки зрения как тренировочных, так и валидационных метрик. Тестовая точность 86.78% является обнадеживающим результатом, и дальнейшие эксперименты могут помочь улучшить модель еще больше.

Для более детального анализа производительности модели можно использовать `classification_report` из библиотеки `sklearn`. Код с результатами представлен на рисунке 3.4.3.

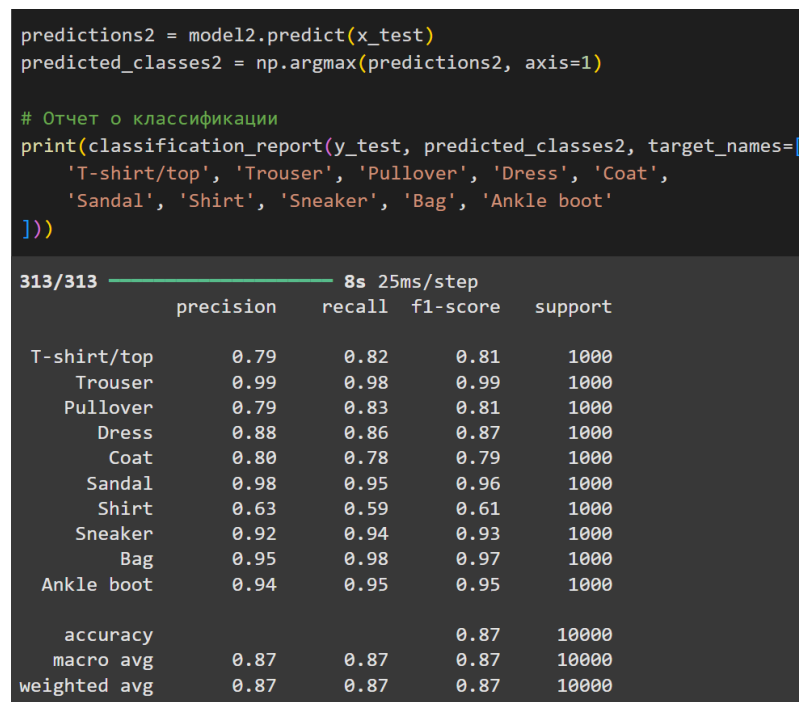


Рисунок 3.4.3 - Прогнозирование на тестовом наборе для второй модели

Метрики по классам:

- Precision (точность):** Это доля истинных положительных результатов среди всех положительных предсказаний.
- Recall (полнота):** Это доля истинных положительных результатов среди всех фактических положительных примеров.
- F1-score:** Это гармоническое среднее между точностью и полнотой, что позволяет оценить баланс между этими двумя метриками.
- Support:** Это количество фактических примеров для каждого класса в тестовом наборе.

Модель показывает высокую точность (precision) и полноту (recall) для большинства классов, особенно для класса "Trouser", который имеет почти идеальные значения. Класс "Shirt" имеет наименьшие значения точности и полноты, что указывает на то, что модель испытывает трудности с правильной классификацией этого класса. Это может быть связано с недостатком данных для обучения или с тем, что класс "Shirt" имеет схожесть с другими классами.

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены две модели сверточных нейронных сетей (CNN) для классификации изображений из набора данных Fashion MNIST. Обе модели были обучены на одних и тех же данных, но имели различные архитектуры и гиперпараметры, что позволило провести их сравнительный анализ.

Модель 1 [Приложение А] состоит из трех сверточных слоев с увеличением числа фильтров, за которыми следуют слои подвыборки (MaxPooling) и полносвязные слои. Она использует оптимизатор Adam и достигает точности на тестовом наборе данных в 87.20% после 20 эпох обучения. Модель демонстрирует стабильный рост точности на обучающем и валидационном наборах, что указывает на хорошую способность к обобщению.

Модель 2 [Приложение А] имеет более сложную архитектуру с увеличенным числом фильтров и добавлением слоя дропаут (Dropout) для борьбы с переобучением. Она использует оптимизатор SGD с моментумом и достигает точности на тестовом наборе данных в 86.78%. Хотя *Модель 2* также показывает рост точности, она не достигает такой же высокой производительности, как *Модель 1*.

Сравнение результатов показывает, что *Модель 1* превосходит *Модель 2* по точности на тестовом наборе данных. Это может быть связано с тем, что более простая архитектура модели 1 лучше подходит для данного набора данных, в то время как более сложная модель 2 может страдать от переобучения, несмотря на использование дропаутов.

Таким образом, можно сделать вывод, что для задачи классификации изображений из набора данных Fashion MNIST *Модель 1* является более эффективной и предпочтительной, так как она демонстрирует более высокую точность и стабильность в обучении. Однако стоит отметить, что выбор модели всегда зависит от конкретной задачи и требований к производительности, и в других сценариях более сложные архитектуры могут показать лучшие результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Иванько А.Ф., Иванько М.А., Горчакова Я.В. МЕТОДЫ РАСПОЗНАВАНИЯ ОБРАЗОВ И ЗАДАЧИ ЛОГИЧЕСКОГО ВЫДЕЛЕНИЯ ОБЪЕКТОВ // Научное обозрение. Технические науки. – 2019. – № 3. – С. 36-40; (1 вступление в тему)
2. Визильтер Ю.В., Желтов С.Ю., Бондаренко А.В., Ососков М.В., Моржин А.В. Обработка и анализ изображений в задачах машинного зрения: Курс лекций и практических занятий. М.: Физматкнига, 2010. 672 с. (вступление)
3. Маршалко Д.А., Кубанских О.В. Архитектура свёрточных нейронных сетей // Ученые записки Брянского государственного университета. 2019. № 4. С. 10. УДК 004.89.
4. Васильев Ю. А., Арзамасов К. М., Владзимирский А. В., Омелянская О. В., Бобровская Т. М., Шарова Д. Е., Никитин Н. Ю., Коденко М. Р. Подготовка набора данных для обучения и тестирования программного обеспечения на основе технологии искусственного интеллекта: учебное пособие. — Издательские решения, 2024. — 131 с. — 53 ил. — ISBN 978-5-006-21244-2.
5. Стеценко А. И. Разработка нейронной сети для распознавания изображений на базе TensorFlow / А. И. Стеценко, Асемгуль С. Смагулова. — Карагандинский Технический Университет имени Абылкаса Сагинова, 2023. — 50 с.
6. Потапов А. Автоматический анализ изображений и распознавание образов – М.: LAP Lambert Academic Publishing, 2017. – 292 с.
7. Паттанаяк С. Глубокое обучение и TensorFlow для профессионалов – М.: Диалектика, 2019. – 480 с.
8. Гриценко А.В., Дорошенко Н.С. Исследование и классификация методов распознавания изображений в системах компьютерного зрения. // Вестник Ставропольского государственного университета. 2011. № 4. С. 84-89.
9. Магамедова Д.М. OpenCV - инструмент компьютерного зрения. // Тенденции развития науки и образования. 2020. № 63-3. С. 42-48.
10. Омеляненко Я. Эволюционные нейросети на языке Python – М.: ДМК Пресс, 2020. – 310 с.

ПРИЛОЖЕНИЕ А

Исходный код модели

```
!pip install tensorflow
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
def plot_class_distribution(labels):
    # Подсчет количества изображений в каждом классе
    class_counts = np.bincount(labels)

    # Нормализация для получения процентного соотношения
    class_percentages = class_counts / len(labels) * 100

    plt.figure(figsize=(10, 5))
    plt.bar(range(10), class_percentages, color='skyblue', edgecolor='black')
    plt.xticks(range(10), ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                           'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'])
    plt.title('Class Distribution in Fashion-MNIST')
    plt.xlabel('Classes')
    plt.ylabel('Percentage of Images (%)')
    plt.ylim(0, 100) # Установка пределов по оси Y от 0 до 100%
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Вызов функции для отображения распределения классов
plot_class_distribution(y_train)
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.expand_dims(x_train, axis=-1)
```

```

x_test = np.expand_dims(x_test, axis=-1)
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(x_train)
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax') # 10 классов для классификации
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
                    epochs=20,
                    validation_data=(x_test, y_test))
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_acc:.4f}')
plt.figure(figsize=(12, 4))
# График потерь
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')

```



```

plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# График точности
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()

predictions = model.predict(x_test)
def plot_predictions(images, true_labels, predictions, num=10):
    plt.figure(figsize=(15, 5))
    for i in range(num):
        plt.subplot(2, 5, i + 1)
        plt.imshow(images[i].reshape(28, 28), cmap='gray')
        plt.title(f'True: {true_labels[i]}\nPred: {np.argmax(predictions[i])}')
        plt.axis('off')
    plt.show()

# Отображение первых 10 изображений из тестового набора
plot_predictions(x_test, y_test, predictions, num=10)

model2 = models.Sequential([
    layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5), # Дропаут для борьбы с переобучением

```

```

layers.Dense(10, activation='softmax')
])

# Компиляция и обучение второй модели
model2.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9),
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

history2 = model2.fit(datagen.flow(x_train, y_train, batch_size=32),
                    epochs=20,
                    validation_data=(x_test, y_test))

# Оценка второй модели
test_loss2, test_acc2 = model2.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy of model 2: {test_acc2:.4f}')
plt.figure(figsize=(12, 4))

# График потерь
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# График точности
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

```

```
plt.show()
predictions2 = model2.predict(x_test)
predicted_classes2 = np.argmax(predictions2, axis=1)

# Отчет о классификации
print(classification_report(y_test, predicted_classes2, target_names=[
    'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
    'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'
]))
```

ПРИЛОЖЕНИЕ Б
Дополнительные материалы

Ссылка на файл с кодом: https://github.com/lentyss/SII_MTUCI.git