МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Шашковой Дарьи Васильевны
(фамилия, имя, отчество)

Институт (факультет) ИРИТ

Кафедра Информатика и системы управления

Группа 20ИС

Дата защиты «____»_____

Индекс

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт радиоэлектроники и информационных технологий
Направление подготовки (специальность) 09.03.01 Информатика и вычислительная техника
Направленность (профиль) образовательной программы Интеллектуальные системы обработки информации и управления
Кафедра Информатика и системы управления

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра

Студента Шашковой Дарьи Васильевны группы 20 ИС

на тему: «Разработка программы распознавания предметов одежды для помощи слабовидящим пользователям»

СТУДЕНТ:
_____Шашкова Д.В._____
(подпись) (фамилия, и., о.)

_____
(дата)

РУКОВОДИТЕЛЬ:
_____Бухнин А.В.__
(подпись) (фамилия, и., о.)

_____
(дата)

РЕЦЕНЗЕНТ:
_____

_____
(подпись) (фамилия, и., о.)

_____
(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ:
_____Тимофеева О.П._
(подпись) (фамилия, и., о.)

_____

_____
(дата)

КОНСУЛЬТАНТЫ:
1. По _____нормоконтролю _____
_____
_____Шагалова П.А. ___
(подпись) (фамилия, и., о.)

_____
(дата)

2. По_____
_____
(подпись) (фамилия, и., о.)

_____
(дата)

3. По _____
_____
_____
_____
(подпись) (фамилия, и., о.)

_____
(дата)

ВКР защищена _____
(дата)

протокол № _____
с оценкой _____

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Кафедра Информатика и системы управления

УТВЕРЖДАЮ

Зав. кафедрой

_____ О.П.Тимофеева

«___» _____ 2024 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника студенту Шашковой Д.В. группы 20-ИС

1. Тема ВКР «Разработка программы распознавания предметов одежды для помощи слабовидящим пользователям»

(утверждена приказом по вузу от 10.04.24 №1111/5)

2.Срок сдачи студентом законченной работы _____

3. Исходные данные к работе: Large-scale Fashion (DeepFashion) Database

4. Содержание расчётно-пояснительной записки (перечень вопросов, подлежащих разработке)
Исследование алгоритмов машинного обучения для задач классификации изображений
Выбор лучшего алгоритма машинного обучения для задач классификации изображений
Алгоритм преобразования текста в звук
Разработка графического интерфейса компьютерного приложения

5. Перечень графического материала
Актуальность работы
Цели и задачи
Средства разработки
Сравнение результатов обучения моделей нейросетей для распознавания категорий одежды
Сравнение результатов обучения модели нейросетей для распознавания аттрибутов
Сравнение результатов обучения по параметрам модели нейросетей для распознавания аттрибутов
Графический интерфейс компьютерного приложения

6. Консультанты по ВКР (с указанием относящихся к ним разделов)

_____

Нормоконтроль _Шагалова П.А._

7. Дата выдачи задания 25.02.24.

Код и содержание Компетенции	Задание	Проектируемый результат	Отметка о выполнении
ОПК-1 Способен применять естественнонаучные и общеинженерные знания, методы математического анализа и моделирования, теоретического и экспериментального исследования в профессиональной деятельности	Изучить методы и подходы к решению задачи.	Выбрать методы и подходы к решению задачи.	Выполнено
ОПК-2 Способен понимать принципы работы современных информационных технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности	Изучить отечественные и зарубежные источники, собрать информацию по исследуемой задаче.	Изучение и перевод зарубежных статей, описание теоретической части по поставленной задаче.	Выполнено
ОПК-3 Способен решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учётом основных требований информационной безопасности;	Разработать информационную систему согласно выявленным требованиям.	Разработка информационной системы с учетом составленных требований.	Выполнено
ОПК-4 Способен участвовать в разработке стандартов, норм и правил, а также технической документации, связанной с профессиональной деятельностью	Написать пояснительную записку на основе результатов работы разработанного программного обеспечения.	Написание пояснительной записки на основе результатов работы разработанного программного обеспечения.	Выполнено
ОПК-5 Способен инсталлировать программное и аппаратное обеспечение для информационных и	Исследовать и выбрать программное обеспечение для	Исследование и выбор программного обеспечения для	Выполнено

автоматизированных систем	выполнения работы.	выполнения работы.	
ОПК-6 Способен разрабатывать бизнес-планы и технические задания на оснащение отделов, лабораторий офисов компьютерным и сетевым оборудованием	Составить требования для информационной системы.	Составление требований для информационной системы.	Выполнено
ОПК-7 Способен участвовать в настройке и наладке программно-аппаратных комплексов	Провести все необходимые настройки в среде разработки.	Проведение всех необходимых настроек в среде разработки.	Выполнено
ОПК-8 Способен разрабатывать алгоритмы и программы, пригодные для практического применения	Разработать программное обеспечение для оптимизации задачи распределения ресурсов на основе генетического алгоритма.	Разработка программного обеспечения для оптимизации задачи распределения ресурсов на основе генетического алгоритма.	Выполнено
ОПК-9 Способен осваивать методики использования программных средств для решения практических задач	Изучить технические материалы по используемым программным средствам.	Изучение технических материалов по используемым программным средствам.	Выполнено
ПКС-1 Способен разрабатывать, тестировать и сопровождать программное обеспечение автоматизированных систем обработки информации и управления	Протестировать разработанное программное обеспечение.	Тестирование разработанного программного обеспечения.	Выполнено
ПКС-2 Способен проектировать и обеспечивать функционирование автоматизированных систем обработки информации и управления	Спроектировать необходимые алгоритмы для решения поставленной задачи.	Проектирование необходимых алгоритмов для решения поставленной задачи.	Выполнено
ПКС-3 Способен использовать математические методы обработки и анализа информации при решении профессиональных задач	Выбрать методы решения задачи.	Выбор методов решения задачи.	Выполнено

УК-1 Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	Изучить и провести сравнительный анализ средств разработки.	Изучение и проведение сравнительного анализ средств разработки.	Выполнено
УК-2 Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	Выявить задачи, которые необходимо выполнить для достижения поставленной цели.	Выявление задач, которые необходимо выполнить для достижения поставленной цели.	Выполнено
УК-3 Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде	Изучить основы организации социального взаимодействия.	Изучение основ организации социального взаимодействия.	Выполнено
УК-4 Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах)	Уметь использовать государственный и иностранный язык для осуществления деловой коммуникации.	Умение использовать государственный и иностранный язык для осуществления деловой.	Выполнено
УК-5 Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах	Изучить основы организации социального взаимодействия.	Изучение основ организации социального взаимодействия.	Выполнено
УК-6 Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	Составить индивидуальный план работ по выполнению ВКР.	Составление индивидуального плана работ по выполнению ВКР.	Выполнено
УК-7 Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности	Уметь организовывать контроль для поддержания уровня физической подготовки организма.	Умение организовывать контроль для поддержания уровня физической подготовки организма.	Выполнено
УК-8 Способен создавать и поддерживать безопасные условия жизнедеятельности, в том числе при возникновении	Уметь создавать и поддерживать безопасные условия	Умение создавать и поддерживать безопасные условия	Выполнено

чрезвычайных ситуаций	жизнедеятельности.	жизнедеятельности.	
УК-9 Способен принимать обоснованные экономические решения в различных областях жизнедеятельности	Уметь самостоятельно принимать экономические решения.	Умение самостоятельно принимать экономические решения.	Выполнено
УК-10 Способен формировать нетерпимое отношение к коррупционному поведению	Изучить действующие правовые нормы по обеспечению борьбы с коррупцией.	Изучение действующие правовых нормы по обеспечению борьбы с коррупцией.	Выполнено

Руководитель _____А.В. Бухнин
(подпись)

Задание принял к исполнению_____

(дата)

Студент_____Д.В. Шашкова

(подпись)

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

АННОТАЦИЯ

к выпускной квалификационной работе

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника студента Шашковой Д.В. группы 20-ИС
по теме «Разработка программы распознавания предметов одежды для помощи слабовидящим пользователям»

Выпускная квалификационная работа выполнена на 68 страницах, содержит 38 рисунков,
6 таблиц, библиографический список из 9 источников, 1 приложений.
Актуальность: выбранная задача является актуальной и практически значимой, так как разработка программы распознавания предметов одежды позволит значительно упростить процесс выбора одежды на дому, делая его более доступным и удобным т.к. это поможет слабовидящим пользователям преодолеть трудности с определением формы, стиля или бренда одежды без необходимости помощи других людей.

Объект исследования: Приложение
Предмет исследования: Проектирование, разработка и отладка приложения
Цель исследования: Разработка приложения распознавания предметов одежды для помощи слабовидящим пользователям.

Задачи исследования: разработка функционала необходимого для ввода фото на вход приложения, выбор алгоритма распознавания одежды - нейронной сети, разработка алгоритма перевода выходных данных нейросети в описательный текст одежды присутствующей на фото, разработка программы для перевода текстового описания в звуковое описание, написание графического интерфейса компьютерного приложения.

Методы исследования: язык программирования Python

Структура работы:

Во введении приведена актуальность поставленной задачи, выбор архитектуры системы и используемого инструментария.

В 1 разделе «Техническое задание» рассматриваются основные цели, задачи и используемые технические средства при разработке программы

Во 2 разделе «Разработка моделей нейросети» собираются и подготавливаются данные, иссле́дуются возможности предобученных нейросетей и происходит их настройка и обучение.

В 3 разделе «Разработка компьютерного приложения» с помощью библиотек PyQt5 реализуется компьютерное приложение и конвертируется в формат .exe

В заключении проанализирована проделанная работа и сделаны выводы.

Выводы:

1. Изучены виды, параметры предобученных нейронных сетей
2. Обучены модели нейросетей для распознавания категорий и аттрибутов
3. Реализовано приложение распознавания категорий и аттрибутов одежды для помощи слабовидящим пользователям

_____/Шашкова Д.В. «____» _____ 20_____ г.

подпись студента /расшифровка подписи

Содержание

Введение

Нейросети, мощные алгоритмы машинного обучения, способные имитировать работу человеческого мозга, стали неотъемлемой частью нашего повседневного мира. Их влияние на различные сферы жизни невозможно переоценить, начиная от автоматического распознавания речи до прогнозирования погоды. Однако особое место среди применений нейросетей занимает их роль в улучшении качества жизни людей с ограниченными возможностями.

Специфика взаимодействия человека с окружающим миром требует адаптации технологий под индивидуальные потребности каждого пользователя. Для людей с ограниченными возможностями, таких как слепые или слабовидящие, доступ к информации и услугам становится особенно сложным. Нейросети предлагают решение этой проблемы, предоставляя инструменты для преобразования текста в речь, распознавания образов и даже навигацию без необходимости использования зрения. Эти технологии не только улучшают качество жизни инвалидов, но и способствуют их интеграции в общество, обеспечивая равный доступ к информационным ресурсам и сервисам.

Важность разработки и распространения приложений на основе нейросетей заключается в их универсальности и доступности. Приложения, использующие нейросети, могут быть загружены на практически любое устройство, что позволяет людям с ограниченными возможностями получать необходимую помощь в любом месте и в любой момент времени. Такие решения не только упрощают процесс получения информации, но и способствуют социализации пользователей, позволяя им участвовать в общественной жизни на равных условиях.

Своей выпускной квалификационной работой я хотела бы облегчить жизнь слепым и слабовидящим пользователям в повседневном выборе одежды например перед тем как пойти на работу, на встречу, отдых или в любое другое место. Для этого было решено написать несколько алгоритмов глубокого обучения и внедрить их в компьютерное приложение, чтобы обеспечить максимальный комфорт в использовании пользователями с ограниченными возможностями.

1 Техническое задание

1.1 Назначение разработки

Реализованный проект должен представлять собой функционирующее компьютерное приложение для считывания изображений с камеры, которое в последствие распознает категорию и атрибуты(узор, длину, текстуру) одежды и озвучит получившееся описание. Должен быть реализован удобный пользовательский интерфейс, помогающий слепым и слабовидящим пользователям в использовании приложения.

1.2 Задачи работы

Задачами работы являются - разработка и обучение нейросети для распознавания категорий(например платье, футболка) и атрибутов(узор, длину, текстуру) одежды, написание подпрограммы для создания описания из полученных на выходе нейросети данных, написание подпрограммы для озвучки данного описания, написание интерфейса приложения с возможностью - сделать фотографию, сгенерировать описание, озвучить последнее описание, выйти из приложения.

1.3 Используемые технические средства

- Язык программирования Python
- Графический интерфейс Anaconda Navigator
- Открытая программная библиотека для машинного обучения TensorFlow
- Фреймворк машинного обучения для языка Python с открытым исходным кодом, созданный на базе Torch. Используется для решения различных задач: компьютерное зрение, обработка естественного языка - PyTorch
- Библиотека машинного обучения, используемая для задач глубокого обучения - Fastai
- Набор расширений графического фреймворка Qt для языка программирования Python, выполненный в виде расширения Python - PyQt5

2 Разработка моделей нейросети

2.1 Исследование потребностей людей с ограниченными возможностями

Прежде чем выбрать тему проекта было решено изучить потребности слепых и слабовидящих пользователей. Были написаны письма в различные организации и просмотрены видеоролики со слепыми людьми, где рассказывалось о том, в чем такие люди нуждаются больше всего

1. Согласно [1]. большинство незрячих людей хотели бы больше всего уметь различать цвета, а также увидеть как со временем изменяются лица их близких и друзей
2. Организация “Видеть мир душой” утверждает, что наибольшим спросом бы пользовался хороший навигатор, который бы указывал точное направление для человека, например идти прямо, а не ориентировался бы на компас.
3. Организация “Жизнь незрячих” также сказала о необходимости навигационного приложение, которое например водило бы по метрополитену с подробными описаниями станций, выходом к переходам и тд, а также имело бы возможность онлайн попросить об оказании помощи у сотрудников службы сопровождения метрополитена.
4. Организация “ТИФЛО LIFE” ещё раз подтвердило о необходимости хорошего навигационного приложения для слепых.
5. Председатель Нижегородской организации "общество слепых" Ирина Сумарокова считает, что было бы полезным иметь приложение для описания одежды и обуви, или описания товаров на полке продуктовых магазинов или такое приложение, которое бы предупреждало пользователей о наличие рядом хрупких и опасных для пользователей предметов - например посуды или фарфора.

Были также отправлены письма во Всероссийское общество слепых и Общероссийскую общественную организации инвалидов «Всероссийское ордена Трудового Красного Знамени общество слепых», однако впоследствии я была перенаправлена в Нижегородский филиал ВОС.

Собрав все вышеперечисленное и проанализировав свои возможности, было решено написать приложение по распознаванию одежды. Это поможет пользователям самостоятельно одеваться дома, не боясь выглядеть нелепо или неприемлемо для общества.

2.2 Поиск и сбор данных

Первостепенной задачей в разработке нейросети является поиск и сбор данных необходимых для обучения.

Для этого можно использовать три основных подхода: ручной поиск изображений, написание программы для сбора изображений из интернета и использование готовых наборов данных из интернет-ресурсов.

Ручной поиск изображений подразумевает поиск и сбор изображений одежды вручную через интернет. Этот метод требует значительного времени и усилий, так как каждое изображение нужно выбирать и загружать вручную. Однако, это позволяет получить очень точный и специфический набор данных, который точно соответствует установленным требованиям. Ручной поиск подходит для небольших проектов или когда требуется очень специфический набор данных.

Написание программы для автоматического сбора изображений из интернета — это более эффективный способ, чем ручной поиск. Программа может использовать Application Programming Interface (API) различных сервисов, таких как Google Images, Bing, Yandex, чтобы искать и скачивать изображения, соответствующие заданным критериям. Этот метод позволяет собрать большой объем данных за короткое время, но требует знаний программирования и понимания того, как работают API для поиска изображений.

Использование готовых наборов данных из интернет-ресурсов — это самый быстрый и простой способ получить данные для классификации изображений одежды. Многие организации и исследователи публикуют свои наборы данных в открытом доступе, что позволяет другим использовать эти данные для своих проектов.

Было решено воспользоваться последним методом в качестве основного источника изображений. На просторах интернета существует множество различных наборов, вот некоторые из них:

- Fashion-MNIST: это набор данных изображений статей Заландо, состоящий из обучающего набора из 60 000 примеров и тестового набора из 10 000 примеров. Каждый пример представляет собой изображение в оттенках серого размером 28x28, связанное с меткой из 10 классов. Fashion-MNIST является упрощённой версией оригинального набора данных MNIST, но вместо рукописных цифр содержит изображения различных видов одежды.

- Clothing1M: Это один из самых больших наборов данных, специально созданных для задач классификации одежды. Содержит 1M изображений одежды в 14 классах. Это набор данных с зашумленными метками, поскольку данные собираются с нескольких вебсайтов интернет-магазинов и включают множество неправильно маркированных образцов. Этот набор данных также содержит 50 000, 14 000 и 10 000 изображений с чистыми метками для обучения, проверки и тестирования соответственно. Clothing1M предлагает большую разнообразие и сложность по сравнению с другими наборами данных, делая его идеальным для продвинутых исследований в области глубокого обучения.

- DeepFashion крупномасштабная база данных одежды. Она содержит более 800 000 разнообразных изображений моды, от хорошо поставленных изображений магазинов до непринуждённых фотографий потребителей, что составляет крупнейшую базу данных визуального анализа моды. Содержит обширную информацию о предметах одежды. Каждое изображение в этом наборе данных помечено 50 категориями, 1000 описательными атрибутами, ограничивающей рамкой и ориентирами одежды. А также содержит более 300 000 пар изображений в разных позах и разных областях. [2]

С использованием базы данных DeepFashion разрабатываются четыре теста, включая прогнозирование атрибутов, прогнозирование категорий, поиск одежды на изображении, сегментация изображения одежды.

Так как задача спрогнозировать описание не только категории одежды, но и атрибутов, было решено воспользоваться набором данных DeepFashion, позволяющим обучить нейросеть и на тех и на других данных.

2.3 Загрузка набора

Данные набора данных DeepFashion были загружены с официального сайта (см список литературы 1.) и представляют из себя следующее:

- Изображения (Img/img.zip) 289 222 различных изображений одежды.
- Аннотации к рамкам (Anno/list_bbox.txt) надписи на рамках.
- Аннотации к модным знакам (Anno/list_landmarks.txt)
- Аннотации к категориям (Anno/list_category_cloth.txt & Anno/list_category_img.txt) ярлыки категорий одежды
- Аннотации к атрибутам (Anno/list_attr_cloth.txt & Anno/list_attr_img.txt) - ярлыки атрибутов одежды.
- Оценочные разделы (Eval/list_eval_partition.txt) - названия изображений для тренировочной, валидационной и тестовой выборки соответственно.

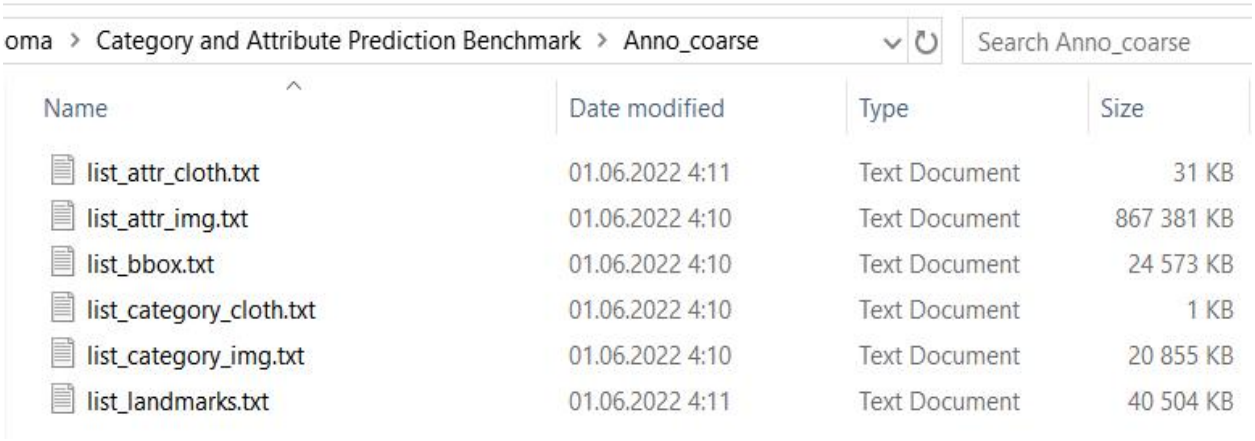


Рисунок 1. - Anno Coarse

Есть отдельная папка с данными разбитыми для тренировочной, валидационной и тестовой выборок. Однако это сокращённый набор для обучения, который содержит всего по 14 000 изображений на каждую выборку.

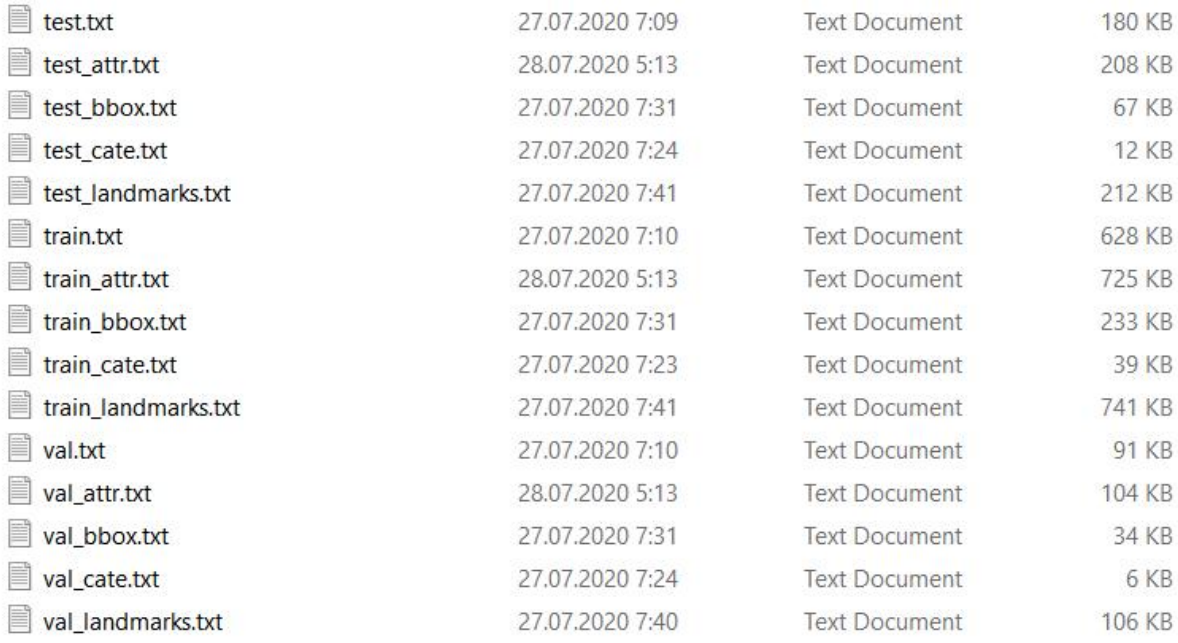


Рисунок 2. - Anno File

2.4 Приведение набора в формат удобный для обучения модели

Первая строка - загрузка необходимых библиотек.

```
import numpy as np
import pandas as pd
```

NumPy (Numerical Python) — это библиотека Python, которая предоставляет поддержку для больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими

массивами. NumPy позволяет эффективно работать с большими объёмами данных, обеспечивая высокую производительность.

Pandas — это библиотека Python, построенная на основе NumPy, которая предоставляет структуры данных и инструменты для анализа данных. Pandas включает в себя структуры данных, такие как Series и DataFrame, которые обеспечивают удобный способ организации и манипулирования данными. Pandas особенно полезен для работы с табличными данными, предоставляя мощные инструменты для очистки, трансформации и анализа данных. Pandas также поддерживает импорт данных из различных источников, таких как CSV, Excel, SQL и JSON

Загрузка первого файла с путями до изображений и их категориями.

list_category_img = pd.read_csv('D:/diploma/Category and Attribute Prediction Benchmark/Anno_coarse/list_category_img.txt', on_bad_lines='skip',skiprows=1, sep='\s+', engine='python')

Несмотря на то, чтобы функция read_csv предназначена в первую очередь для чтения файла формата csv она также подходит и для чтения формата txt.

Для чтения из файла использовались следующие параметры:

Во-первых из-за того что первая строка txt была для указания размера датафрейма, необходимо её пропустить с помощью skiprows=1.

Во-вторых путь до изображения и категорию разделяет около 20 пробелов и обычным разделителем ‘ ‘ или ‘\t‘ считать её не представляется возможным, воспользуемся sep='\s+'

В-третьих из-за чрезмерно большого объёма данных пришлось использовать другой движок engine='python', а также параметр on_bad_lines='skip'

Проверка корректности считывания файла:

```
list_category_img.head()
```

	image_name	category_label
0	img/Sheer_Pleated-Front_Blouse/img_00000001.jpg	3
1	img/Sheer_Pleated-Front_Blouse/img_00000002.jpg	3
2	img/Sheer_Pleated-Front_Blouse/img_00000003.jpg	3
3	img/Sheer_Pleated-Front_Blouse/img_00000004.jpg	3
4	img/Sheer_Pleated-Front_Blouse/img_00000005.jpg	3

Рисунок 3. - Проверка list_category_img

Аналогичным образом был загружен файл с путями до изображения и его атрибутами. Разделитель sep='\s*\.jpg\s*' обусловлен тем что расстояние между image_name и attribute_labels различное. Также было решено принять его за символы до и включая jpg и все что после.

list_attr_img = pd.read_csv('D:/diploma/Category and Attribute Prediction Benchmark/Anno_coarse/list_attr_img.txt', on_bad_lines='skip',skiprows=2, sep='\s*\.jpg\s*', names=['image_name', 'attribute_labels'], engine='python'

Проверка корректности считывания файла:

```
list_attr_img.head()
```

	image_name	attribute_labels
0	img/Sheer_Pleated-Front_Blouse/img_00000001.jpg	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -...
1	img/Sheer_Pleated-Front_Blouse/img_00000002.jpg	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -...
2	img/Sheer_Pleated-Front_Blouse/img_00000003.jpg	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -...
3	img/Sheer_Pleated-Front_Blouse/img_00000004.jpg	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -...
4	img/Sheer_Pleated-Front_Blouse/img_00000005.jpg	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -...

Рисунок 4. - Проверка list_attr_img

Новая задача - изменение всех путей до изображений все пути до изображения на полные, а также изменение названий самих изображений.

Следующей подпрограммой все изображения перемещаются в 1 папку с файлами:

```
source_folder = 'D:/diploma/img'
target_folder = 'D:/diploma/img_highres'
if not os.path.exists(target_folder):
    os.makedirs(target_folder)
for subdir, dirs, files in os.walk(source_folder):
    for file in tqdm(files, desc=f"Processing {os.path.basename(subdir)}"):
        if file.endswith(('.jpg', '.jpeg', '.png')):
            source_file_path = os.path.join(subdir, file)
            target_file_path = os.path.join(target_folder, os.path.basename(subdir) + '__' +
file)
            shutil.copy2(source_file_path, target_file_path)
```

Изменение пути до изображений для категориального массива:

```
list_category_img['image_name'] = list_category_img['image_name'].str.replace('/img_', '__img_')
list_category_img['image_name'] = list_category_img['image_name'].str.replace('img', 'C:/Users/USRx/Desktop/Daria/img_highres', 1)
```

Проверка корректности изменений:

```
list_category_img.iloc[0]['image_name']
```

```
'C:/Users/USRx/Desktop/Daria/img_highres/Sheer_Pleated-Front_Blouse__img_00000001.jpg'
```

Рисунок 5. - Проверка изменений list_category_img

Изменение пути до изображений для массива атрибутов:

```
list_attr_img['image_name'] = list_attr_img['image_name'].str.replace('/img_', '__img_')
list_attr_img['image_name'] = list_attr_img['image_name'].str.replace('img', 'C:/Users/USRx/Desktop/Daria/img_highres', 1)
```

Проверка корректности изменений:

```
list_attr_img.iloc[0]['image_name']
```

```
'C:/Users/USRx/Desktop/Daria/img_highres/Sheer_Pleated-Front_Blouse__img_00000001.jpg'
```

Рисунок 6. - Проверка изменений list_attr_img

Преобразование массива pandas в массив numpy.

```
list_category_img = list_category_img.to_numpy()
list_attr_img = list_attr_img.to_numpy()
```

Сохранение файлов в формате npy.

```
np.save('category_img_new.npy',list_category_img)
np.save('attr_img_new.npy',list_attr_img)
```

2.5 Подготовка среды разработки

Перед началом работы с нейросетями необходимо установить виртуальную среду разработки.

Виртуальная среда помогает решать конфликты зависимостей проекта, создавая изолированные среды. Виртуальные среды включают новую копию двоичных файлов

Python и автономную копию всей стандартной библиотеки Python. Поэтому она может работать сама по себе.

Для этого необходимо установить Anaconda Navigator и создать в ней виртуальную среду, выбрав версию python.

В течение выпускной квалификационной работы было создано 2 виртуальные среды разработки нейросетей: на основе TensorFlow и на основе PyTorch и Fastai.

2.5.1 Подготовка виртуальной среды с TensorFlow

Создание виртуальной среды используя встроенную консоль Anaconda [3]

Выбор версии python - 3.9.18

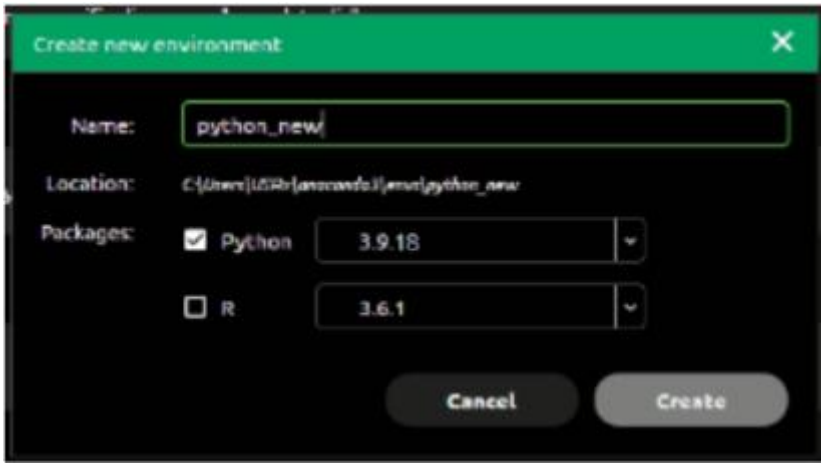


Рисунок 7. - Создание виртуального окружения

Установка необходимых библиотек:

- tensorflow == 2.6.0
- numpy == 1.23.4
- pandas == 2.2.1
- pillow == 10.3.0

Также необходимо установить ipykernel для возможности использования тетрадок Jupyter и tensorflow-gpu, чтобы иметь возможность отправить вычислительные операции с центрального процессора CPU на графический процессор GPU. Что значительно ускорит обучения нейросетей.

Остальные необходимые библиотеки Anaconda Navigator подтягивает самостоятельно. Версии библиотек приходилось подбирать, чтобы они не конфликтовали друг с другом, например numpy был понижен до 1.23.4.

Следующий этап - открытие среды разработки VSCode и выбор виртуальной среды среди предлагаемых.

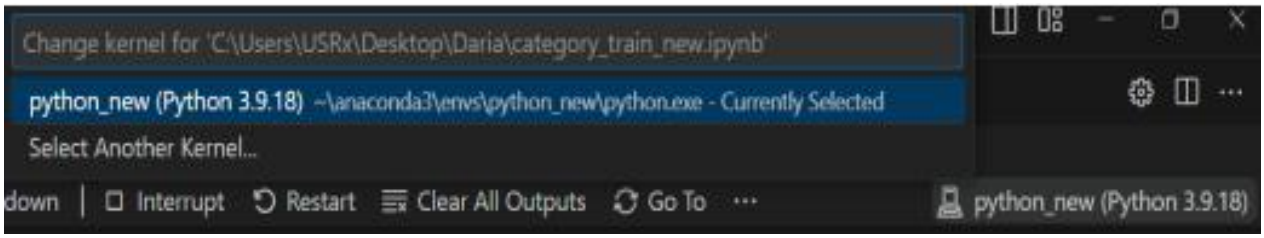


Рисунок 8. - Выбор виртуального окружения

2.5.2 Подготовка виртуальной среды с PyTorch и Fastai

Эта виртуальная среда создавалась с использованием встроенной консоли Anaconda. Это необходимо, так как некоторые библиотеки, в том числе Fastai можно установить только таким способом. [4]

Пустая виртуальная среда было создана аналогично предыдущей, но использовалась версия python 3.9.19.

Была открыта консоль виртуальной среды и введены следующие команды:

- Установка cudatoolkit необходимые для подключения GPU к нашей программе:

conda install cudatoolkit -c anaconda -y

- Установка поддержки Cuda для PyTorch. Важно чтобы версия Cuda совпадала с pytorch-cuda. conda install pytorch-cuda=12.1 -c pytorch -c nvidia -y
- Установка библиотек PyTorch для работы с изображениями и звуком.

conda install pytorch torchvision torchaudio -c pytorch -c nvidia -y

- Установка Fastai

conda install fastai -c fastai -c pytorch -c nvidia -y

Оставшиеся библиотеки numpy и pandas устанавливаются обычным способом через Anaconda Navigator.

2.6 Разработка нейросети для распознавания категорий

Существует множество различных моделей работающих с изображениями. Франсуа Шолле приводит по крайней мере 6 моделей, среди которых есть такие как VGG16, ResNet,MobileNet,EfficientNet. [5]

В работе были использованы следующие предобученные нейросети:

- из библиотеки TensorFlow Keras такие модели как VGG16, MobileNet50v2, ResNet50v2

- из библиотеки Fastai такие модели как VGG16, ResNet34, EfficientNet

Далее выбирается лучшая модель и улучшается её эффективность с помощью изменений параметров, а также рассмотрены результаты работы на данных, на которых она не обучалась.

2.6.1 Обзор моделей нейросетей

- Модель VGG16 — это архитектура сверточной нейронной сети (CNN), разработанная командой Visual Geometry Group из Оксфордского университета. Она была представлена на конференции ICLR в 2015 году как часть серии моделей VGG, включая VGG8, VGG16 и VGG19.

Структура VGG16

VGG16 состоит из 16 слоями, включая 13 сверточных слоёв, 3 полносвязных слоя и один выходной слой. Архитектура модели организована таким образом, чтобы постепенно увеличивать сложность признаков:

Сверточные слои: В начале модели идут несколько сверточных слоёв с небольшим количеством фильтров (например, 64 или 128). Эти слои используются для обнаружения простых признаков, таких как края и текстуры.

Пуллинг слои: После сверточных слоёв следуют пуллинг-слои (max pooling), которые уменьшают размерность пространства признаков, сохраняя при этом наиболее важную информацию.

Полносвязные слои: Затем модель переходит к более глубоким слоям, где используются полносвязные слои для интеграции информации из различных частей изображения и формирования более сложных признаков.

Выходной слой: На последнем этапе модель выдаёт предсказание, используя softmax функцию для определения вероятностей принадлежности изображения к каждому из классов.

Особенности VGG16

Глубина: Одной из ключевых особенностей VGG16 является её глубина. Модель использует большое количество сверточных слоёв, что позволяет ей эффективно обучаться на сложных задачах классификации изображений.

Использование стандартных блоков: VGG16 использует стандартные блоки (такие как conv + relu + maxpool) без использования специализированных операций, таких как batch normalization или dropout, что упрощает понимание и реализацию модели.

- Модель MobileNet50 представляет собой лёгкую сверточную нейронную сеть (CNN), разработанную для мобильных приложений и устройств с ограниченными ресурсами. Она была впервые представлена как часть TensorFlow и открытого программного обеспечения Google. Основной особенностью MobileNet является использование глубоких разделённых свёрток (depthwise separable convolutions), что позволяет значительно уменьшить количество параметров по сравнению с другими сетями, имеющими обычные свёртки и одинаковую глубину. Это приводит к созданию более легковесных глубоких нейронных сетей, сохраняющих высокую точность классификации.

Глубокие разделённые свёртки состоят из двух операций: глубокой свёртки (depthwise convolution) и свёртки типа pointwise (pointwise convolution). Глубокая свёртка применяется к каждому каналу входного изображения независимо, а затем результаты объединяются в одной свёртке типа pointwise. Этот подход позволяет сократить количество параметров и вычислений, делая модель более эффективной с точки зрения использования памяти и вычислительных ресурсов.

Кроме того, MobileNet поддерживает два глобальных гиперпараметра для дальнейшего снижения вычислительных затрат: ширины мультиплеера (width multiplier) и мультиплеера разрешения (resolution wise multiplier). Эти параметры позволяют адаптировать архитектуру под конкретные задачи и ограничения устройства, изменяя количество слоёв и их размерности без необходимости изменения самой структуры сети.

- Модель ResNet50v2 - также известная как ResNet-50, является одним из наиболее известных архитектур сверточных нейронных сетей (CNN), которая достигла наивысших результатов на датасете ImageNet в 2015 году. Эта модель состоит из 50 слоёв и включает в себя 16 резидуальных блоков, каждый из которых содержит несколько сверточных слоёв с резидуальными связями. Архитектура также включает в себя слои пулинга, полносвязные слои и слой выхода softmax для классификации.

Основные особенности и компоненты архитектуры ResNet50:

Резидуальные блоки: Резидуальный блок — это основной строительный блок архитектуры ResNet50, который используется для поддержания одного и того же размера входа и выхода. Резидуальный блок состоит из трёх сверточных слоёв, каждый из которых следует за нормализацией пакета и функцией активации ReLU. Вход добавляется к выходу третьего сверточного слоя.

Проекционные блоки: Проекционный блок используется, когда размеры входа и выхода различаются. Этот блок включает в себя сверточный слой со скользящим шагом (2, 2) для downsampling входа и сверточный слой с размером фильтра (1, 1) для изменения глубины входа, чтобы соответствовать выходу.

Слои пулинга: Слой пулинга используется для уменьшения пространственных размеров карт признаков, произведенных сверточными слоями, сохраняя при этом самые важные информацию. Это может помочь уменьшить вычислительную стоимость сети и предотвратить переобучение.

Полносвязные слои: Также известные как плотные слои, они представляют собой тип слоя нейронной сети, где все нейроны в слое подключены ко всем нейронам в предыдущем слое. Эти слои обычно используются в качестве последних слоев в нейронной сети и отвечают за окончательные прогнозы.

Нормализация пакета: Нормализация пакета обычно применяется после сверточных или полностью связанных слоёв в нейронной сети, но перед функцией активации. Это широко используемая техника в глубоком обучении, которая показала улучшение производительности многих типов нейронных сетей.

ResNet50v2 отличается от оригинальной модели ResNet50 тем, что в нем были внесены изменения в проекционные блоки для улучшения производительности и эффективности обучения.

- Модель ResNet34 - является одной из версий архитектуры ResNet, разработанной для решения задач классификации изображений. ResNet34 состоит из 34 слоёв и использует концепцию резидуальных связей для облегчения обучения глубоких нейронных сетей.

Основные особенности и компоненты архитектуры ResNet34 включают:

Конволюция и пулинг: Модель начинается с блока Conv1, включающего в себя операцию конволюции с ядром размером 7x7, нормализацию пакета и операцию максимального пулинга. Это позволяет уменьшить размер входного изображения до (112x112) и увеличить число каналов до 64.

Блоки: Каждый слой ResNet34 состоит из нескольких блоков, которые могут быть базовыми (с двумя операциями) или узкими (с тремя операциями). Операции включают в себя сверточные слои, нормализацию пакета и функцию активации ReLU, кроме последнего слоя в блоке, который не имеет функции активации ReLU.

Резидуальные связи: Главная особенность ResNet34 заключается в использовании резидуальных связей, которые позволяют пропускать вход данных через некоторые слои сети напрямую, соединяя их с выходом предыдущего слоя. Это помогает решить проблему исчезающего градиента в глубоких сетях и улучшает процесс обучения.

Узкие блоки: Узкий блок в ResNet34 состоит из трёх операций: две сверточные операции с размером фильтра 3x3 и одна сверточная операция с размером фильтра 1x1. Это позволяет эффективно увеличивать глубину сети без значительного увеличения количества параметров.

Повторение слоёв: После первого блока Conv1, сеть повторяет паттерн слоёв, где каждый слой выполняет 3x3 конволюцию с фиксированным размером карты признаков (64, 128, 256, 512 соответственно) и пропускает вход через каждые два сверточных слоя. Размеры ширины и высоты остаются постоянными на протяжении всего слоя.

Изменение размера: Изменение размера входного объёма происходит за счёт увеличения шага в первом сверточном слое каждого слоя с 1 до 2, вместо использования операции пулинга.

- Модель EfficientNet представляет собой архитектуру сверточной нейронной сети (CNN), разработанную для достижения высокой точности классификации изображений с минимальным количеством параметров и вычислений. Эта модель использует метод масштабирования всех измерений глубины/ширины/разрешения с помощью составного коэффициента, что позволяет ей эффективно использовать ресурсы и обеспечивать высокую производительность.

Основные особенности и компоненты архитектуры EfficientNet включают:

Мобильные инвертированные бутстрепы (MBConv): Это фундаментальная строительная блок архитектуры EfficientNet, вдохновлённая инвертированными резидуальными блоками из MobileNetV2, но с некоторыми модификациями. MBConv начинается с глубоковой свёртки, за которой следует свёртка типа point-wise (1x1 свёртка), расширяющая количество каналов, и ещё одна 1x1 свёртка, уменьшающая каналы обратно до исходного числа. Этот дизайн "бутылочного горлышка" позволяет модели эффективно учиться, сохраняя при этом высокую степень представительной мощности.

Блоки Squeeze-and-Excitation (SE): Эти блоки помогают модели научиться сосредотачиваться на основных признаках и подавлять менее релевантные. Блок SE использует глобальное среднее пулинг для уменьшения пространственных размеров карты признаков до одного канала, за которым следуют два полностью связанных слоя.

Эти слои позволяют модели учиться зависимостям между каналами и создавать веса внимания, которые умножаются на исходную карту признаков, акцентируя внимание на важной информации.

Принцип масштабирования: EfficientNet использует составной коэффициент φ для равномерного масштабирования ширины, глубины и разрешения сети в принципиально правильном порядке. Этот подход обоснован интуитивно понятным предположением, что если входное изображение больше, то сети требуется больше слоёв для увеличения поля восприятия и больше каналов для захвата более тонко-гранулярных шаблонов на большем изображении.

Таким образом, каждая из моделей имеет свои особенности в архитектуре, что позволяет ей быть эффективнее других на тех или иных задачах, поэтому было решено использовать каждую из них, для большей уверенности в результате.

2.6.2 Алгоритм обучения на TensorFlow

На этом этапе необходимо выбрать виртуальную среду в соответствии с пунктом 2.4.1, в которой установлена библиотека TensorFlow

Импортирование необходимых библиотеки:

```
import tensorflow as tf
from tensorflow import keras
```

Далее произведена проверка подключения к GPU, так как обучение нейросети представляет из себя множество вычислительных процессов, которые центральный процессор не в состоянии обработать достаточно быстро.

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print(tf.test.is_built_with_cuda())
```

✓ 3.2s

```
Num GPUs Available:  1
True
```

Рисунок 9. - Проверка подключения GPU

Загрузка набора библиотек отвечающего за загрузку и обработку табличных данных и изображений.

```
import numpy as np
import pandas as pd
```

```
import os
import PIL.Image
from PIL.Image import Resampling
PIL.Image.MAX_IMAGE_PIXELS = None


from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model
from keras import layers
from tensorflow.keras import losses
```

Загрузка в переменную path_labels ранее созданного файла category_img_new:

В функции np.load необходимо использовать настройку allow_pickle=True, которая позволит загружать данные из файлов npy несмотря на угрозу безопасности.

```
path_labels =
np.load('C:/Users/USRx/Desktop/Daria/category_img_new.npy',allow_pickle=True)
```

Формирование dataframe, разделяя данные на входные признаки - path и целевой label

```
df = pd.DataFrame(path_labels, columns=['path','label'])
```

Входной признак: Это отдельные независимые переменные. Модели прогнозирования используют признаки для формирования прогнозов. В нашем случае это изображения, которые приводятся к нужному размеру, цвету и формату, чтобы их можно было подать на вход нейросети.

Целевой признак: является результатом входных переменных - прогноз нашей модели. В нашем случае это отдельные классы - категории, на которые входные переменные могут быть сопоставлены в задачи классификации.

Так как набор данных очень большой и загрузить в оперативную память все 300 000 фотографий невозможно, предобработку необходимо будет совершать во время обучения с помощью ImageDataGenerator. Однако этот генератор принимает на вход строковые значения. Поэтому приведём столбец label к строковому типу

```
df['label'] = df['label'].astype(str)
```

Для обучения модели также необходимо сформировать тренировочную, валидационную и тестовую выборки.

Тренировочная выборка: Используется для обучения модели. Это набор данных, на котором модель настраивает свои параметры для минимизации ошибки. Тренировочная выборка позволяет модели научиться распознавать шаблоны и делать прогнозы на основе входных данных.

Валидационная выборка: Используется для ранней остановки процесса обучения. Валидационная выборка позволяет оценить, насколько хорошо модель работает на новых данных, которые она ещё не видела во время обучения. Если ошибка на валидационной выборке начинает расти, это может указывать на переобучение, и обучение может быть прекращено, чтобы избежать дальнейшего ухудшения качества модели.

Тестовая выборка: Используется после окончания обучения для окончательной оценки качества модели. Тестовая выборка представляет собой набор данных, который модель никогда не видела во время обучения или валидации. Результаты на тестовой выборке дают объективную оценку того, насколько хорошо модель справляется с новыми, неизвестными данными. Это важно для понимания, насколько надёжно модель будет работать в реальных условиях

Создание пустых датафреймов pandas, с указанием названия столбцов.

```
train_df = pd.DataFrame(columns=['path', 'label'])
val_df = pd.DataFrame(columns=['path', 'label'])
test_df = pd.DataFrame(columns=['path', 'label'])
```

Отбор уникальных значений меток, для определения точного количества классов представленных на данном наборе данных.

```
unique_labels = df['label'].unique()
```

Была написана функция, которая разделяет данные в соотношении 3:1:1. Для этого перебирая значения меток, были срезаны данные со всеми одинаковыми метками. Для тестовой выборки оставим 80% от всех данных, а для валидационной и тестовой по 10%. Затем датафреймы объединяются с полученными срезами.

```
for label in unique_labels:
    label_data = df[df['label'] == label]
    train_size = int(len(label_data)*0.8)
    val_size = int(len(label_data) * 0.1)
    train_data = label_data[:train_size]
    val_data = label_data[train_size:train_size+val_size]
```

```
test_data = label_data[train_size+val_size:]


train_df = pd.concat([train_df, train_data])
val_df = pd.concat([val_df, val_data])
test_df = pd.concat([test_df, test_data])
```

Проверка разбиения на выборки:

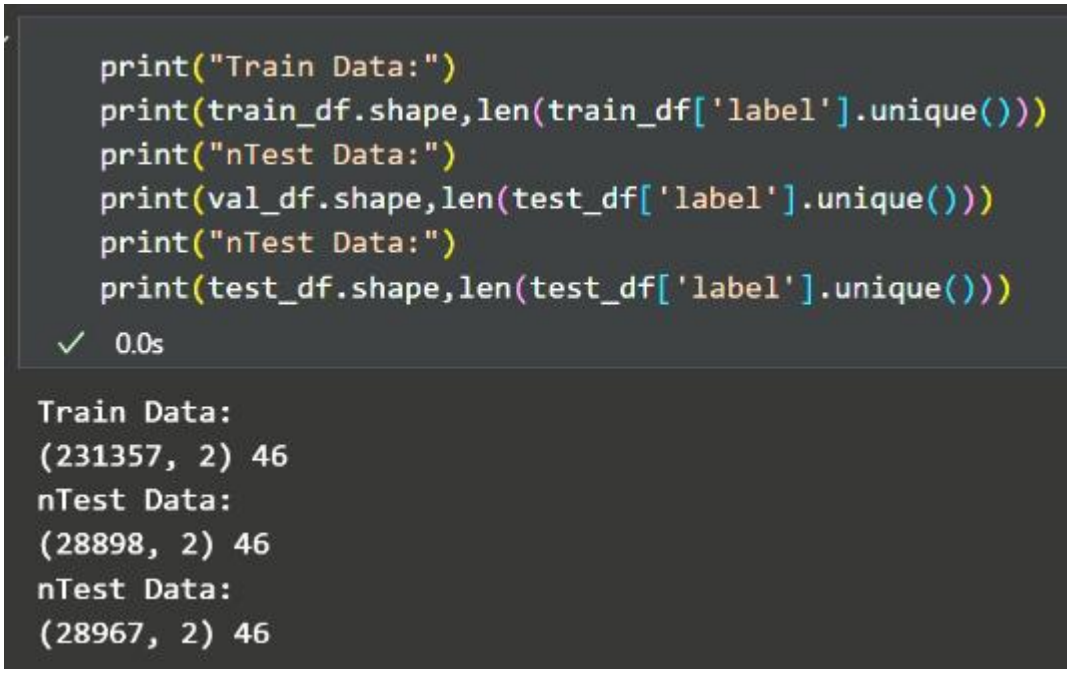


Рисунок 10. - Распределение изображений по выборкам

Отсюда видно, что в каждом датафрейме имеется 46 классов. Это интересно так как изначально заявлено о 50 классах. На самом же деле, некоторые категории данные в датасете представляют из себя атрибуты - такие как flannel и button-down то есть фланелевая ткань и пуговицы по всей длине. Поэтому самих классов немного меньше чем предполагается. Остальных выборок касается тоже самое.

В тренировочной выборке осталось 231357 изображений, в валидационной 28898 изображений, а в тестовой 28967 изображений.

Для того чтобы нейросеть обучалась более равномерно, получившиеся датафреймы были перемешаны.

```
train_df = train_df.sample(frac=1).reset_index(drop=True)
val_df = val_df.sample(frac=1).reset_index(drop=True)
test_df = test_df.sample(frac=1).reset_index(drop=True)
```

Создание генератора, который помимо всего прочего будет нормировать данные с помощью параметра rescale и обогащать их поворачивая под небольшим градусом horizontal_flip

```
datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
)
```

Был определён формат, который будет подаваться на вход нейросети. Все нейросети, которые используются будут принимать на вход изображения в формате 224,244,3. То есть размер изображения 224x224 и 3 канала RGB. Изображения были приведены к заданной форме с помощью target_size.

Было определённо из какого датафрейма необходимо загружать данные dataframe, выбраны входные и целевой столбец с помощью x_col и y_col, определён размер входных пакетов подаваемых на вход нейросети batch_size и вид классификации. Так как имеется 46 категорий, то это многоклассовая классификация, а не бинарная, следовательно class_mode = ‘categorial’

Генератор для тренировочной выборки:

```
train_generator = datagen.flow_from_dataframe(
    dataframe=train_df,
    directory=None,
    x_col='path',
    y_col='label',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

На выходе было получено следующее сообщение:

```
Found 231357 validated image filenames belonging to 46 classes.
```

Рисунок 11. - Результаты создания генератора

В нем говорится о том, что было найдено 231377 изображений принадлежащих к 46 уникальным классам.

Генератор для валидационной выборки:

```
val_generator = datagen.flow_from_dataframe(
    dataframe=val_df,
    directory=None,
    x_col='path',
    y_col='label',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

Генератор для тестовой выборки:

```
test_generator = datagen.flow_from_dataframe(
    dataframe=test_df,
    directory=None,
    x_col='path',
    y_col='label',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical')
```

Установка библиотек необходимых для построения нейросети.

```
from keras_applications.vgg16 import VGG16, preprocess_input
from keras_applications.mobilenet_v2 import MobileNetV2
from keras_applications.resnet_v2 import ResNet50V2
from tensorflow.keras.optimizers import Adam
```

Здесь находятся вышеупомянутые VGG16, MobileNetV2 и ResNet50V2, которые были предобучены на наборе данных ImageNet. Этот набор включает небольшое количество изображений с одеждой, но в нашей задаче больше интересует сама архитектура нейросети нежели её прежнее обучение. Также имеется preprocess_input, который предобрабатывает изображения перед подачей на вход VGG16, и оптимизатор Adam

Был создан базовый слой на основе первой выбранной нейросети и установлен необходимый входной размер для изображения 224 на 224 с тремя каналами RGB.

Параметр include_top установлен на False, чтобы нейросеть обучилась для новых 46 классах, а conv_base.trainable = False, говорит о том, что веса модели на которых она обучалась ранее будут заморожены, чтобы нейросеть уже имела некоторое представление о входных данных.

```
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)
conv_base.trainable = False
```

Была создана основная структура новой нейросети.

- Входной слой, на который будут подаваться значения размером (224,224,3).
- Слой предобработки для VGG16.
- Слой VGG16
- Слой Flatten, который снижает размерности до одной, так как на выходе vgg16 имеется tensor размером 5,5,512, а необходимо получить 46.
- Выходной слой Dense с размерностью 1 длинной 46 - равным количеству классов с функцией активации softmax. Слой активации необходим для получения вероятности принадлежности к каждому классу. Так softmax преобразует вектор числовых значений в вектор вероятностей, суммирующихся до единицы.

```
inputs = keras.Input(shape=(224, 224, 3))
x = keras.applications.vgg16.preprocess_input(inputs)
x = conv_base(inputs)
x = layers.Flatten()(x)
outputs = layers.Dense(46, activation="softmax")(x)
model = keras.Model(inputs, outputs)
```

Для компиляции модели необходимо заполнить следующие параметры компилирования:

Optimizer: Оптимизатор отвечает за обновление весов модели во время процесса обучения. Он выбирает направление и скорость изменения весов на основе текущего состояния модели и её производных. Adam является одним из наиболее популярных оптимизаторов, который автоматически адаптируется к изменяющимся

условиям обучения, используя механизмы адаптивного градиента и моментов

Loss Function (Функция потерь): Функция потерь измеряет разницу между прогнозами модели и истинными значениями. Она используется для определения, насколько хорошо модель справляется с задачей. Categorical_crossentropy рассчитывается как отрицательное ожидание логарифма истинных вероятностей классов, умноженных на соответствующие предсказанные вероятности. Это выражение позволяет модели минимизировать ошибку классификации, учитывая все возможные классы одновременно.

Metrics (Метрики): Метрики используются для мониторинга производительности модели во время обучения и оценки её качества после обучения. Они могут быть использованы для сравнения различных моделей или для отслеживания улучшения одной и той же модели во времени. Accuracy рассчитывается как отношения правильно предсказанных значений, ко всем предсказанным значениям.

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

Также является важным сохранять лучшую модель на основе валидационных данных, для этого используется класс ModelCheckpoint, который сохраняет модель по указанному пути каждый раз когда она достигла наилучших результатов - в нашем случае, лучшего значения accuracy на валидационной выборке.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
    filepath="resnet50.keras",
    save_best_only=True,
    monitor="val_accuracy"
)]
```

Последний этап - обучение модели. Было решено обучать все модели TensorFlow на 35 эпохах, что более чем достаточно для использования предобученных моделей.

```
history = model.fit(
    train_generator,
    epochs=35,
```

```
    validation_data=val_generator,
    callbacks=callbacks
)
```

После обучения модель VGG16 показала следующие результаты:

```
Epoch 30/40
7230/7230 [==============================] - 1537s 213ms/step - loss: 2.5172 - accuracy: 0.3124 - val_loss: 2.5560 - val_accuracy: 0.3202
Epoch 31/40
7230/7230 [==============================] - 1530s 212ms/step - loss: 2.5192 - accuracy: 0.3116 - val_loss: 2.5363 - val_accuracy: 0.3171
Epoch 32/40
7230/7230 [==============================] - 1533s 212ms/step - loss: 2.5152 - accuracy: 0.3121 - val_loss: 2.6020 - val_accuracy: 0.3246
Epoch 33/40
7230/7230 [==============================] - 1550s 214ms/step - loss: 2.5107 - accuracy: 0.3133 - val_loss: 2.5970 - val_accuracy: 0.3183
Epoch 34/40
7230/7230 [==============================] - 1518s 210ms/step - loss: 2.5133 - accuracy: 0.3133 - val_loss: 2.6823 - val_accuracy: 0.3171
Epoch 35/40
7230/7230 [==============================] - 1512s 209ms/step - loss: 2.5144 - accuracy: 0.3125 - val_loss: 2.6594 - val_accuracy: 0.3003
```

Рисунок 12. - Результаты обучения VGG16

За 35 эпох модель смогла обучиться всего лишь до 30% правильных ответов и при этом согласно нестабильным оценкам на валидации происходит переобучение модели. Такой расклад означает то, что модель VGG16 плохо подходит для классификации изображений одежды и требуется более сложная модель.

Обучение следующей модели ResNet50v2. Для этого был изменён слой с предобученной моделью на ResNet50v2

```
conv_base = keras.applications.resnet_v2.ResNet50V2(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)
conv_base.trainable = False
```

И изменена основная структура модели с удалением слоя предобработки для vgg16.

```
inputs = keras.Input(shape=(224, 224, 3))
x = conv_base(inputs)
x = layers.Flatten()(x)
outputs = layers.Dense(46, activation="softmax")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="categorical_crossentropy",
    optimizer=Adam(),
    metrics=["accuracy"])
```

Изменение пути для сохранения лучшей модели

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
    filepath="resnet50v2.keras",
    save_best_only=True,
    monitor="val_accuracy")
]
```

Модель ResNet50V2 показала следующие результаты:

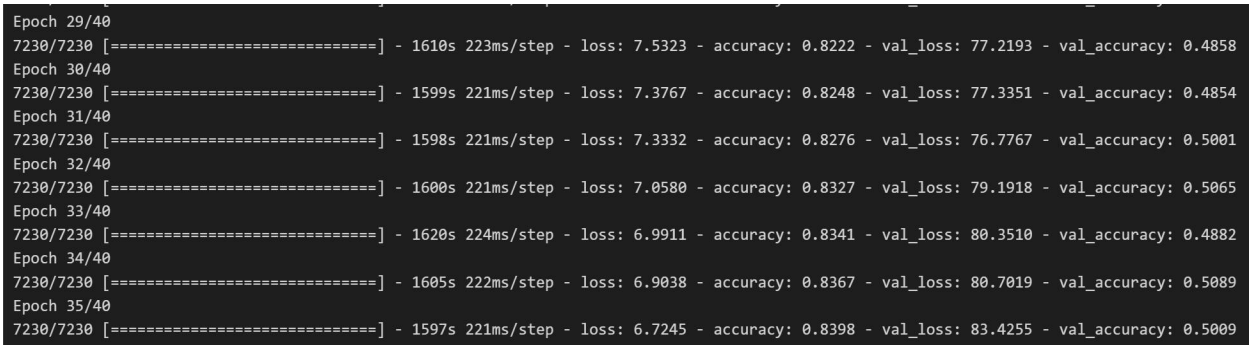


Рисунок 13. - Результаты обучения модели ResNet50V2

За 35 эпох модель смогла обучиться всего лишь до 83% правильных ответов, однако на валидационной выборке результат составил всего 50%, модель также демонстрирует признаки переобучения, а значит нет смысла продолжать её обучать.

Результат для ResNet50v2 куда лучше чем на VGG16 однако все ещё далёк от идеала, ведь нам важен результат на валидационной и тестовой выборках, что означает что результат в 50% нас не удовлетворяет.

Обучение следующей модели MobileNetv2. Для этого был изменён слой с предобученной моделью на MobileNetv2.

```
conv_base = keras.applications.mobilenet_v2.MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)
conv_base.trainable = False
```

Основная структура модели осталась той же.

```
inputs = keras.Input(shape=(224, 224, 3))
x = conv_base(inputs)
x = layers.Flatten()(x)
outputs = layers.Dense(46, activation="softmax")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="categorical_crossentropy",
    optimizer=Adam(),
    metrics=["accuracy"])
```

Изменение пути для сохранения лучшей модели

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
    filepath="mobilenet.keras",
    save_best_only=True,
    monitor="val_accuracy")
]
```

Модель MobileNetV2 показала следующие результаты:

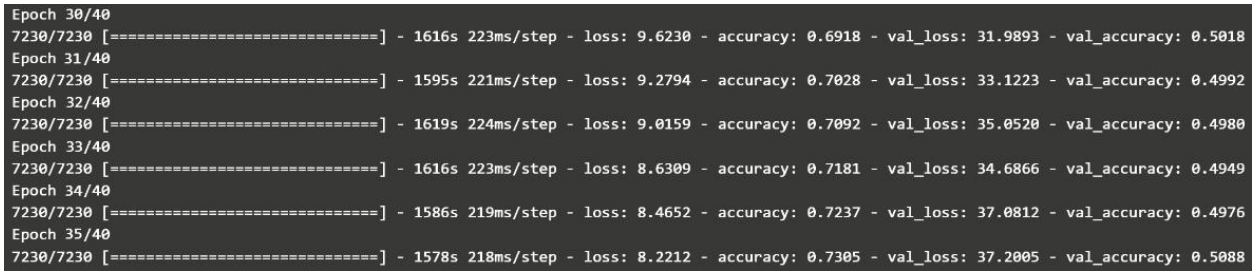


Рисунок 14. - Результаты обучения модели MobileNetV2

За 35 эпох модель смогла обучиться до 73% правильных ответов, а на валидационной выборке результат составил 51%. При этом ошибка оказалась почти в 2 раза меньше на валидации чем у ResNet50v2

Результат для MobileNetV2 лучше чем на VGG16 и ResNet50V2 и все ещё далёк от хороших результатов. Для внедрения модели в приложения необходим результат хотя бы в 70% правильных ответов.

Итого среди моделей предоставленных библиотекой TensorFlow лучшие результаты дала MobileNetV2 . Если модели на Fastai не покажут лучших результатов, необходимо будет пробовать улучшить результаты на MobileNetV2 .

2.6.3 Алгоритм обучения нейросетей на PyTorch и Fastai

Для этого этапа необходимо активировать вторую виртуальную среду разработки в соответствии с пунктом 2.4.2. с установленными библиотеками PyTorch и Fastai

Импорт необходимых библиотек:

```
import torch
from fastai.vision.data import ImageDataLoaders
from fastai.vision.all import *
from fastai.imports import *
import gc
import pandas as pd
import numpy as np
%matplotlib inline
```

Помимо основных библиотек torch и fastai были также импортированы библиотеки gc и matplotlib. Первая библиотека поможет очистить кэш программы, что очень важно для с точки зрения экономии памяти компьютера. Вторая же библиотека отвечает за построение графиков, которые понадобятся позже для анализа эффективности и улучшения обучения нейросети.

В первую очередь необходимо проверить подключение GPU к нашей среде разработки:

```
if torch.cuda.is_available():
    print("GPU доступен.")

    num_gpus = torch.cuda.device_count()
    print(f"Количество доступных GPU: {num_gpus}")

else:
    print("GPU недоступен.")
```

```
GPU доступен.
Количество доступных GPU: 1
```

Рисунок 15. - Проверка подключения GPU с Torch

GPU доступен, значит можно начать работу.

В первую очередь необходимо загрузить данные и разделить их аналогично предыдущей программе за тем исключением, что нет необходимости разделять данные на валидационную выборку, а также перемешивать таблицы вручную. С этим справятся функции из библиотеки fastai:

```
path_labels =
np.load('C:/Users/USRx/Desktop/Daria/category_img_new.npy',allow_pickle=True)
df = pd.DataFrame(path_labels, columns=['path','label'])
train_df = pd.DataFrame(columns=['path', 'label'])
test_df = pd.DataFrame(columns=['path', 'label'])
unique_labels = df['label'].unique()
for label in unique_labels:
    label_data = df[df['label'] == label]
    train_size = int(len(label_data)*0.8)
    train_data = label_data[:train_size]
    test_data = label_data[train_size:]
    train_df = pd.concat([train_df, train_data])
test_df = pd.concat([test_df, test_data])
```

Проверка разделённых данных:

```
print("Train Data:")
print(train_df.shape,len(train_df['label'].unique()))
print("nTest Data:")
print(test_df.shape,len(test_df['label'].unique()))
```

```
Train Data:
(231357, 2) 46
nTest Data:
(57865, 2) 46
```

Рисунок 16. - Проверка разделения данных на выборки

Для того чтобы обрабатывать изображения во время обучения нейросети, был использован аналог функции ImageDataGenerator из библиотеки TensorFlow - ImageDataLoaders из библиотеки Fastai.

Генератор для тренировочной выборки:

```
data = ImageDataLoaders.from_df(df=train_df, bs=64, seed=42, shuffle=True,
item_tfms=Resize(224), batch_tfms=aug_transforms(min_scale=0.9), valid_pct=0.1)
```

Помимо уже очевидных df=train_df и bs=64, появились новые параметры. Так seed позволяет каждый раз перемешивать или разделять данные строго определенным образом, это удобно при обучении моделей. Shuffle позволяет перемешивать выборку перед тем

как подать её на вход модели. item_tfms применяет функции к изображениям, в нашем случае достаточно просто изменить размер изображения. valid_pct отвечает за то, какую часть от тренировочной выборки взять в качестве валидационной. batch_tfms применяет различные функции для обогащения данных, в нашем случае изменение размера изображения минимально от 90%

Генератор для тестовой выборки:

```
data = ImageDataLoaders.from_df(df=test_df, bs=64, item_tfms=Resize(224))
```

Теперь можно переходить сразу к обучению нейросетей, так как функция visual_learner в fastai самостоятельно определяет сколько классов будет на выходе в обучении, также в ней по умолчанию установлены оптимизатор Adam и функция категориальной кросс-энтропии для измерения потерь, поэтому нет смысла вводить их самостоятельно.

Код для обучения модели VGG16

```
learn = vision_learner(data, vgg16, metrics=accuracy,
weights='VGG16_Weights.DEFAULT')
learn.fine_tune(10)
```

Модель VGG16 показала следующие результаты:

epoch	train_loss	valid_loss	accuracy	time
0	1.611596	1.443953	0.573720	52:07
1	1.460089	1.324826	0.611671	52:45
2	1.429225	1.260854	0.629133	52:37
3	1.342570	1.209502	0.643354	52:55
4	1.258745	1.165765	0.659434	52:21
5	1.242280	1.144846	0.667041	52:29
6	1.203891	1.121049	0.668295	52:46
7	1.200859	1.107230	0.677675	53:46
8	1.138486	1.088581	0.681262	52:29
9	1.113598	1.081953	0.683164	51:56

Рисунок 17. - Результаты обучения модели VGG16

Модель дала точность в 68,3% причём потери продолжают уменьшаться как на тренировочной выборке так и на валидации, где результат даже лучше. Это значит, что VGG16 на fastai лучше чем все предыдущие модели.

Код для обучения модели EfficientNet. Мы будем использовать версию b2 с чуть более сложной структурой, чтобы модель могла воспринимать более сложные образы.

learn = vision_learner(data, efficientnet_b2, metrics=accuracy,weights=EfficientNet_B2_Weights.DEFAULT)

learn.fine_tune(10)

Модель EfficientNet показала следующие результаты:

epoch	train_loss	valid_loss	accuracy	time
0	1.521417	1.413306	0.584439	47:12
1	1.437437	1.301098	0.617203	46:53
2	1.363957	1.214382	0.641755	47:30
3	1.308365	1.159044	0.656711	47:11
4	1.241770	1.124838	0.668857	47:11
5	1.168729	1.102826	0.672660	46:56
6	1.156606	1.092353	0.677545	48:00
7	1.121612	1.084452	0.678539	48:08
8	1.165025	1.074702	0.679793	47:33
9	1.128305	1.076408	0.680225	47:22

Рисунок 18. - Результаты обучения модели EfficientNet

Модель EfficientNet показала результаты хуже чем VGG16 и смогла обучиться на 68% что чуть хуже. Потери также показывают результаты немного хуже. Так на тренировочной выборке они составили 1.12, против 1.11 у vgg16.

Код для обучения модели ResNet34

learn = vision_learner(data, resnet34, metrics=accuracy, weights='ResNet34_Weights.DEFAULT')

learn.fine_tune(10)

После обучения модель ResNet34 показала следующие результаты:

epoch	train_loss	valid_loss	accuracy	time
0	1.291395	1.160646	0.657273	44:39
1	1.270610	1.177075	0.651480	44:33
2	1.219316	1.107528	0.675081	44:38
3	1.141825	1.064988	0.688178	44:40
4	1.097458	1.042702	0.693408	44:43
5	0.996158	1.017572	0.703307	44:49
6	0.910841	1.008294	0.708537	44:50
7	0.863683	1.004868	0.714113	44:49
8	0.815881	1.008290	0.713637	44:47
9	0.766834	1.008782	0.714761	44:53

Рисунок 19. - Результаты обучения модели ResNet34

Модель ResNet34 после 10 эпох показала результат в 71%, и потери 0.76 для тренировочной и 1 на валидационной, причём признаки переобучения она показала уже на 7 итерации, после чего потери начали увеличиваться.

Все результаты обучения моделей были сведены в единую таблицу, чтобы принять окончательное решение о том, какую модель следует применять и улучшать.

Таблица 1. - Сравнение результатов обучения моделей

TensorFlow				
Модели	train - accuracy	train- loss	valid - accuracy	valid - loss
VGG16	0.31	2.5	0.3	2.7
ResNet50v2	0.83	6.7	0.5	83.4
MobileNetv2	0.73	8.2	0.5	37.2
Fastai и Torch				
Модели	train - accuracy	train- loss	valid - accuracy	valid - loss
VGG16	0.683	1.11	>0.683	1.08
EfficientNet	0.680	1.12	>0.680	1.07

ResNet34	0.71	0.76	<0.71 но >0.68	1.0

Таким образом, лучшей моделью оказалась ResNet34, которая показала лучшие результаты и по точности и по потерям на валидационной выборки.

Для улучшения результатов обучения модели, был использован анализ предоставленный в книге Джереми Форда Глубокое обучение в fastai и pytorch [6]

```
learn = vision_learner(data, resnet34, metrics=accuracy, pretrained=True
learn.fine_tune(2)
```

epoch	train_loss	valid_loss	accuracy	time
0	1.590717	1.428768	0.583315	45:06

epoch	train_loss	valid_loss	accuracy	time
0	1.288283	1.158631	0.663540	44:35
1	1.150757	1.040408	0.694186	44:40

Рисунок 20. - Первые итерации обучения ResNet34

Когда вызывается функция learn.fine_tune(), вся сеть замораживается и обучается в течение определённой эпохи только на случайно инициализированных весах вновь созданного слоя. Затем сеть размораживается и обучаются все слои вместе в течение указанного количества эпох. Поэтому при вызове функции learn.fine_tune() видно результат для трёх эпох, а не для двух.

Далее была построена матрица ошибок для нашей модели. Эта матрица по диагонали показывает правильные предсказания. То есть класс 1 предсказан как класс 1. Остальные же элементы матрицы показывают неправильные предсказания. Но они могут показать то, какие классы с какими модель путает чаще всего. Так судя по матрице анорак(1 класс) часто путается с ветровкой(11 класс)

```
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix(figsize=(15,15), dpi=60)
```

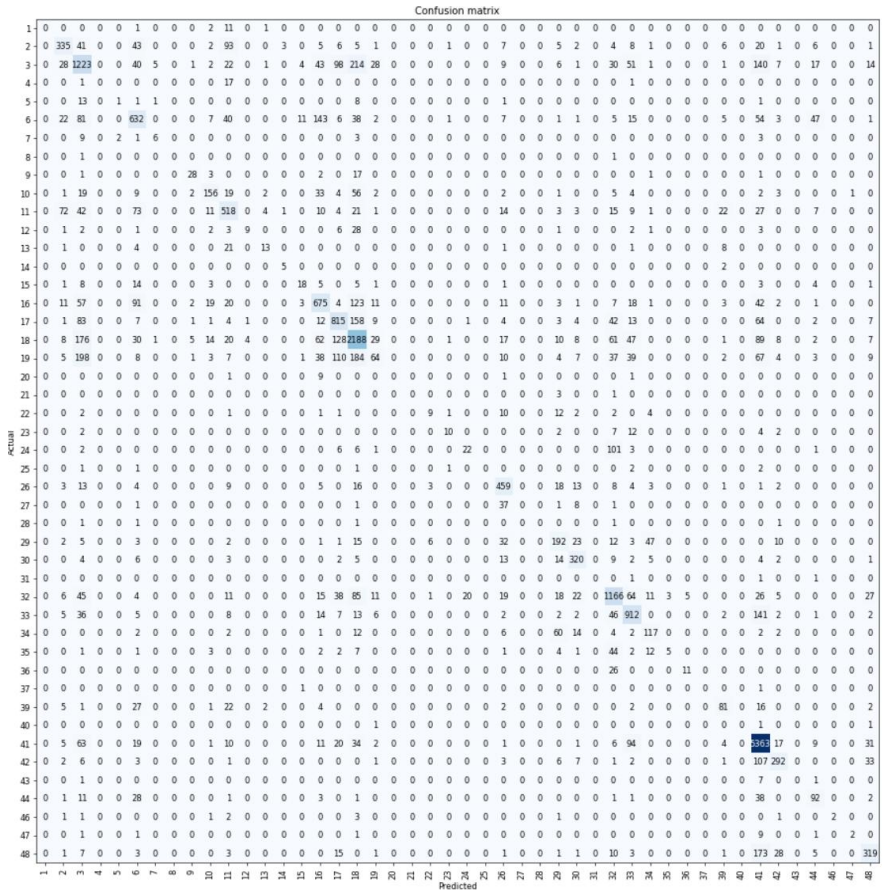


Рисунок 21. - Матрица ошибок для ResNet34

Выберем классы которые путаются наиболее часто, в качестве минимального порога выберем 50 ошибок:

interp.most_confused(min_val=50)

```
[(3, 18, 214),
 (19, 3, 198),
 (19, 18, 184),
 (18, 3, 176),
 (48, 41, 173),
 (17, 18, 158),
 (6, 16, 143),
 (33, 41, 141),
 (3, 41, 140),
 (18, 17, 128),
 (16, 18, 123),
 (19, 17, 110),
 (42, 41, 107),
 (24, 32, 101),
 (3, 17, 98),
 (41, 33, 94),
 (2, 11, 93),
 (16, 6, 91),
 (18, 41, 89),
 (32, 18, 85),
 (17, 3, 83),
 (6, 3, 81),
 (11, 6, 73),
 (11, 2, 72),
 (19, 41, 67),
 (17, 41, 64),
 (32, 33, 64),
 (41, 3, 63),
 (18, 16, 62),
 (18, 32, 61),
```

Рисунок 22. - Наиболее частые ошибки для ResNet34

Эту таблицу следует читать следующим образом: первый столбец - реальный класс, второй столбец - предсказанный класс, 3 столбец - количество ошибок.

Так если рассматривать первую строчку 3 класс (блузка) неправильно предсказан как 18 класс (футболка) целых 214 раз или как 41 класс (платье) 140 раз. Причём такие предсказания довольно ‘человечны’ так как не всегда по части фотографии человек может понять разницу между блузкой или платьем с воротничком из такой же ткани.

Для улучшения показателей было рассмотрено на одной эпохе как обучается модель и изменяется скорость её обучения.

```
learn_2 = vision_learner(data, resnet34, metrics=accuracy)
lr_range = learn_2.lr_find()
```

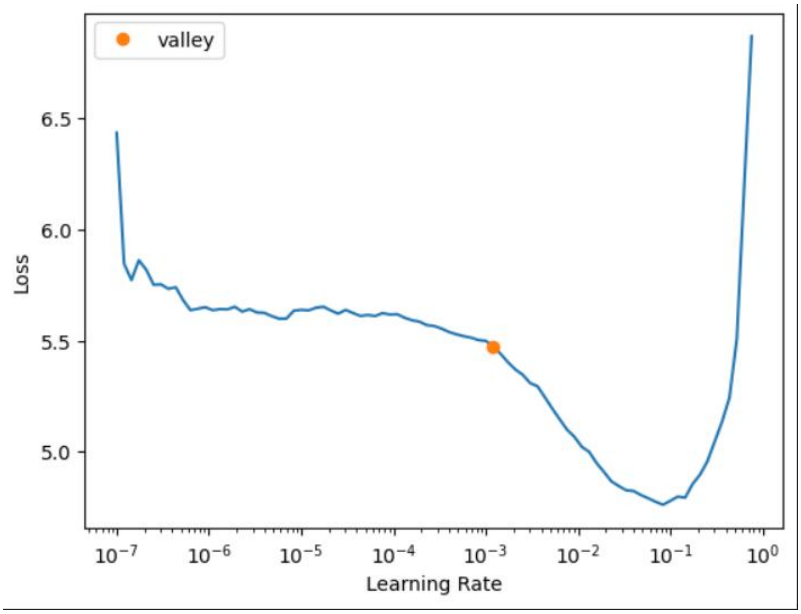


Рисунок 23. - Скорость обучения модели ResNet34 на первой эпохи

Необходимо найти наиболее эффективную скорость обучения, которая позволит сети быстрее объединяться. Эта точка является самым крутым наклоном кривой потерь. Точки экстремумов и плоские участки кривой соответствуют темпам обучения, которые не позволяют сети обучаться, поскольку потери в этих точках не увеличиваются.

Точка на кривой показывает наилучшее значение то есть около 10^-3, но было испробовано несколько значений для большей объективности.

```
learn_3 = vision_learner(data, resnet34, metrics=accuracy)
learn_3.fine_tune(3, base_lr=0.0012)
learn_4 = vision_learner(data, resnet34, metrics=accuracy)
learn_4.fine_tune(3, base_lr=0.0025)
```

```
learn_5 = vision_learner(data, resnet34, metrics=accuracy)
learn_5.fine_tune(3, base_lr=0.0032)
learn_6 = vision_learner(data, resnet34, metrics=accuracy)
learn_6.fine_tune(3, base_lr=0.0018)
```

Таблица 2. - Сравнение обучения с разной скоростью модели ResNet34

№	1	2	3	4
Скорость	0.0012	0.0025	0.0032	0.0018
Accuracy	0.696	0.703	0.704	0.702

Отсюда видно что предложенное значение было наиболее хорошим. Поэтому воспользуемся learn_5 со скоростью обучения 0.0032

После обучения на всех уровнях нам необходимо ещё раз пересмотреть скорость обучения, так как после нескольких серий обучения с относительно высокой скоростью обучения прежняя скорость обучения больше не подходит и должна быть снижена по мере замедления обучения.

```
learn_5.unfreeze()
learn_5.lr_find()
```

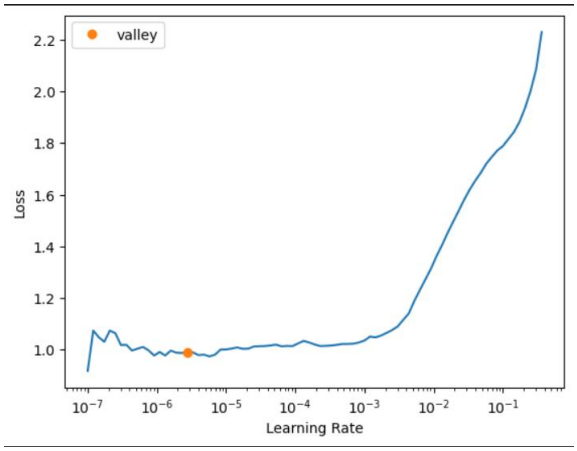


Рисунок 24. - Скорость обучения модели после трёх эпох.

Кривая потерь выглядит иначе, так как веса сети больше не являются случайными. Нет такого резкого снижения, связанного с моментом, когда веса были изменены со случайных на однократные, что уменьшает потери. Форма кривой выглядит более плоской, и необходимо взять диапазон значений от точки уменьшения до точки, где потери снова начнут расти.

Перенесённые из предварительно подготовленной модели, уже хорошо распознают основные визуальные концепции и не требуют особого обучения. В то время как более глубокие слои, которые отвечают за распознавание сложных форм, характерных для нашего проекта, по-прежнему выиграют от более высокой скорости обучения. Поэтому необходимо использовать меньшую скорость обучения для первых слоёв и большую скорость обучения для последних, чтобы они могли настраиваться быстрее, чем предыдущие слои.

```
learn_5.fit_one_cycle(6, lr_max=slice(1e-7, 1e-5))
learn_5.recorder.plot_loss()
```

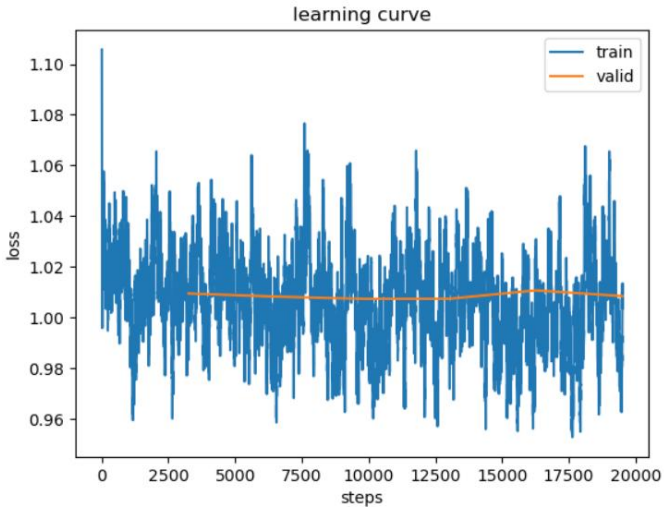


Рисунок 25. - Сравнение потерь на тренировочной выборки и валидации

Построение графика потерь при обучении и валидации помогает оценить, нужно ли продолжать обучение. Потери при валидации уже не так сильно улучшаются, или даже ухудшаются, хотя потери при обучении все ещё улучшаются. Продолжая обучение, мы увеличим разрыв между обучением и потерей валидации,что плохо скажется на нашей модели. Поэтому нам лучше прекратить обучение сейчас.

Сохранение модели таким образом, чтобы дальнейшие предсказания проходили уже на центральном процессоре.

```
learn_5.export("cat_resnet34_export.pkl")
```

Проведение предсказания на данных, которые модель ещё не видела.

```
accuracy_full = 0
total_images = len(test_df)
for i in range(total_images):
```

```
    img = Image.open(test_df.iloc[i]['path']).convert('RGB').resize((224, 224))
    with learn.no_bar(), learn.no_logging():
        prediction = learn.predict(img)
    predicted_class = int(prediction[0])
    actual_label = int(test_df.iloc[i]['label'])
    if actual_label == predicted_class:
        accuracy_full += 1
average_accuracy = accuracy_full / total_images
print(f"Average Accuracy: {average_accuracy * 100:.2f}%")
```

```
Average Accuracy: 70.78%
```

Рисунок 26. - Предсказание категории на тестовой выборке.

И на одной фотографии.

```
img=Image.open(r'C:\Users\USRx\Desktop\Daria\sweater.jpg')
img.thumbnail((224,224),Image.LANCZOS)
prediction = learn.predict(img)
print(prediction[0],'sweater=16')
display(img)
```

Результат предсказания:

Рисунок 27. - Предсказание категории на 1 фотографии.

2.7 Разработка нейросети для распознавания атрибутов.

2.7.1 Подготовка данных

Прежде чем начать обучение моделей, необходимо максимально упростить работу для неё.

Сейчас метки для каждого изображения представляют из себя 1000 значений из -1/0/1. Как сказано в пояснениям к данным, -1 отвечает за отсутствие данного аттрибута, 1 за присутствие данного аттрибута.

Достаточно оставить единицы, которые отвечают за строгое наличие данного аттрибута и более того, удалю все -1, преобразовав строку из 1000 элементов в примерный десяток индексов расположений единиц. Например ‘-1 -1 1 -1 -1’ станет ‘2’. Также нужно учесть что расстояние между -1 и 1 два пробела, поэтому сначала нужно будет заменить его на 1. А для тех фотографий которые не принадлежат ни к одному аттрибуту введём новый аттрибут 1000.

Загрузка данных:

```
path_labels = np.load(r'D:\jupyter\Diploma\edit_base\attr_img_new.npy',allow_pickle=True)
df = pd.DataFrame(path_labels, columns=['path','label'])
df['label'] = df['label'].str.replace('  ', ' ')
```

Цикл для преобразования единиц в индексы:

```
for i in range(df.shape[0]):
    numbers = df.loc[i, 'label'].split(' ')
    positive_one_indices = []
    for j, num in enumerate(numbers):
        if num.strip():
            if int(num) == 1:
                positive_one_indices.append(j)
    result_string = ','.join(map(str, positive_one_indices))
    df.loc[i, 'label'] = result_string
df['label'] = df['label'].replace('', '1000')
```

Проверка корректности преобразования:

	path	label
0	C:/Users/USRx/Desktop/Daria/img_highres/Sheer_...	718,820
1	C:/Users/USRx/Desktop/Daria/img_highres/Sheer_...	718,820
2	C:/Users/USRx/Desktop/Daria/img_highres/Sheer_...	142,719,840,960
3	C:/Users/USRx/Desktop/Daria/img_highres/Sheer_...	717
4	C:/Users/USRx/Desktop/Daria/img_highres/Sheer_...	350,407,720,814

Рисунок 28. - Проверка преобразования массива атрибутов

Преобразование в numpy массив и сохранение файла

```
df = df.to_numpy()
np.save(r'D:\jupyter\Diploma\FinalFolder\attr_img_new_shorted.npy',df)
```

Далее были рассмотрены полученные данные. Для этого был подготовлен файл с атрибутами и переведён на русский язык в программе Excel, а затем сохранен в формат csv с использованием Unicode.

```
path_labels =
np.load(r'D:\jupyter\Diploma\FinalFolder\attr_img_new_shorted.npy',allow_pickle=True)
df = pd.DataFrame(path_labels, columns=['path','label'])
dict_df = pd.read_csv(r'D:\jupyter\Diploma\attr_cloth.csv',sep=';')
list_my = df.loc[3454,'label'].split(',')
list_my = list(map(int, list_my))
```

```
print(dict_df.iloc[list_my])
```
✓ 0.0s

	english	russian
73	bird	узор с птицами
74	bird print	принт с птицами
730	print	некоторый принт
836	sleeve	рукава

Рисунок 29. - Проверка аттрибутов случайной фотографии

Отсюда видно что для одной фотографий есть очень похожие аттрибуты, то есть в разметке есть шум. Это обусловлено тем, что данные размечались тремя независимыми людьми, отчего одной фотографии присваиваются примерно одни и те же аттрибуты.

То есть аттрибуты нуждаются в чистке. Было решено провести её следующим образом: в ручную перебрать каждый аттрибут, смотря на то, какие ещё аттрибуты есть на фото. Если есть аналогичные, как например если бы рассматривался аттрибут bird, а были ещё bird print, оставляется аттрибут bird. В таблице csv перевод удалённых аттрибутов был заменён на ‘пропуск’, а затем с помощью уже другой функции удалены из списка аттрибутов каждой фотографии.

Функция для поиска дубликатов аттрибутов:

```
search_number = 1000
for i in range(15000, df.shape[0]):
    list_my = list(map(int, df.loc[i,'label'].split(',')))
    if search_number in list_my:
        print(i)
        print(dict_df.iloc[list_my])
        break
```

После сохранения csv были загружены данные и с помощью следующего кода создан список из аттрибутов подлежащих удалению.

```
delete_attr_list = dict_df[dict_df['russian']=='пропуск']
delete_attr_list = delete_attr_list.index.tolist()
delete_attr_list = [str(number) for number in delete_attr_list]
```

Следующей функцией удаляются все аттрибуты подлежащие удалению:

```
for i in range(df.shape[0]):
    new_list = []
    list = df.loc[i,'label'].split(',')
    for item in list:
        if item not in delete_attr_list:
            new_list.append(item)
    df.loc[i,'label'] = ','.join(new_list)
df['label'] = df['label'].replace('', '1000')
```

Всего было удалено около 340 шумных меток, а также после очистки обнаружилось около 14000 пустых фотографий, что составляет всего 4.6% для 300 тысяч фотографий. Это приемлемый результат, кроме того их можно использовать чтобы протестировать модель перед сохранением.

2.7.2 Алгоритм обучения нейросети на PyTorch и Fastai

Как и в прошлый раз будет обучаться несколько моделей и оптимизировать работу лучшей. В этот раз выбор пал на такие модели как ResNet34, Resnet50 и MobileNet:

Импорт необходимых библиотек:

```
import torch
from fastai.vision.all import *
import gc
import numpy as np
import pandas as pd
import os
from PIL import Image
```

Загрузка данных для обучения:

```
path_labels =
np.load('C:/Users/USRx/Desktop/Daria/attr_img_shorted_delattr.npy',allow_pickle=True)
df = pd.DataFrame(path_labels, columns=['path','label'])
```

Необходимо привести столбец path в нужный вид. Для этого достаточно добавить .jpg в конец пути, чтобы обращение происходило в изображениям формата jpg. А также изменить путь до изображения к формату Windows.

```
df['path'] = df['path'].apply(lambda x: os.path.normpath(x))
df['path'] = df['path']+'.jpg'
```

Для загрузки данных во время обучения воспользуемся классом DataBlock, в качестве блоков установим ImageBlock для изображений и MultiCategoryBlock для меток.

RandomSplitter позволит создать валидационные данные сразу во время обучения. Также изменим размер поступаемых изображений до 224.

```
dblock = DataBlock(
    blocks=(ImageBlock,MultiCategoryBlock),
    splitter=RandomSplitter(valid_pct=0.3,seed=42),
    get_x=ColReader('path'),
    get_y=ColReader('label',label_delim=','),
    item_tfms = Resize(224),
    batch_tfms=aug_transforms())
```

dls = dblock.dataloaders(df,num_workers=5,bs=32,shuffle=True)

Проверка корректности полученных блоков:

Рисунок 30. - Проверка блоков для моделей аттрибутов

В этот раз также необходимо будет изменить метрики. Раннее для категорий использовался accuracy. Все ещё возможно воспользоваться ей, однако она будет показывать результаты близкие к идеальным.

Это происходит по той причине, что accuracy показывает общую долю правильно классифицированных экземпляров, но не раскрывает детали о том, как модель справляется с каждым классом отдельно. Это может быть недостаточно информативно, особенно, когда важно знать, как модель работает с каждым классом по отдельности.

Accuracy также может быть искажена в случаях, когда классы несбалансированны (то есть, то есть один класс значительно доминирует над другими). В таких случаях модель может показывать высокую аккуратность, просто хорошо справляясь с доминирующим классом, но плохо справляться с остальными.

Поэтому была применена другая метрика под названием F-мера - это среднее гармоническое между precision и recall, его значение лежит в диапазоне [0,1]. F-мера достигает максимального значения, когда исходные метрики равны единице. Если хотя бы одна из исходных метрик близка к нулю, то и F-мера тоже стремится к нулю. Однако в работе применяется её модификация F-взвешенная. Она имеет дополнительный параметр бета, который позволяет склонить предпочтение в ту или иную сторону.

Для определения precision и recall обратимся к курсу [7]

Precision - показывает точность, с которой модель присваивает объектам класс 1, то есть получает результат типа Positive. Иными словами, precision определяет, не слишком ли часто модель выставляет класс 1 объектам класса 0.

Чем выше эта метрика, тем меньше таких случаев.

Recall - обратная precision. Она измеряет, смогла ли модель классификации присвоить класс 1 всем объектам этого класса. Предсказания False Negative(единицы принятые за 0) рассматриваются метрикой как «неучтённые» True Positive(единицы принятые за 1). Чем выше recall, тем больше прогнозов Positive(единичных) модель смогла дать верно.

Было решено, что recall в данном случае важнее, чтобы модель могла предсказывать все имеющиеся аттрибуты, поэтому, поэтому параметр beta должен быть больше нуля. Он был принят за 2. В случае если аудитория приложения посчитает иначе, этот параметр при наличии необходимых технических мощностей можно будет оперативно скорректировать.

Также в качестве ещё одного параметра F-меры был применён порог равный 0.2. Это значит что если модель с вероятностью 20% и больше уверена что это 1, она определится как 1.

Параметр average=samples настраивает вариант агрегации означает, что оценка F-меры рассчитывается для каждого образца в наборе данных, а затем берется среднее значение этих оценок

В качестве оптимизатора был выбран Ranger. Он представляет собой синергетический оптимизатор, который объединяет в себе несколько ключевых компонентов для улучшения процесса обучения нейронных сетей. Основные составляющие Ranger включают:

RAdam (Rectified Adam): Это вариация алгоритма Adam, которая включает механизм коррекции шага обучения, чтобы уменьшить эффекты накопления градиентов в процессе обучения.

Gradient Centralization: Этот компонент направлен на уменьшение дисперсии(разброс) градиентов, что может помочь улучшить стабильность обучения и ускорить сходимость.

LookAhead: LookAhead оптимизатор предлагает стратегию обучения, которая позволяет использовать более крупные шаги обучения для первых нескольких эпох, а затем постепенно уменьшать эти шаги. Это может помочь улучшить качество обучения и ускорить сходимость.

И была применена к нему настройка weight decay, которая предотвращает переобучение модели, накладывая высокие штрафы на высокие веса. Это сделает модель проще.

wd = 5e-7

opt_func = partial(ranger, wd=wd)

metrics=[FBetaMulti(2.0, 0.2, average='samples'), partial(accuracy_multi, thresh=0.2)]

Обучение модели. Было использовано несколько предобученных моделей, точнее resnet34, resnet50 и regnet.

Обучение ResNet34:

learn = vision_learner(dls, resnet34, loss_func=BCEWithLogitsLossFlat(thresh=0.2), metrics=metrics, opt_func=opt_func).to_fp16()

learn.fine_tune(10)

Результаты обучения модели:

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.021798	0.020755	0.126233	0.995264	1:01:32

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.020513	0.019446	0.179253	0.995225	1:02:38
1	0.020265	0.018931	0.213286	0.995090	1:02:37
2	0.019775	0.018552	0.231096	0.995070	1:01:28
3	0.019553	0.018297	0.245078	0.995076	1:03:06
4	0.018989	0.018086	0.256903	0.995096	1:02:30
5	0.018479	0.017931	0.261260	0.995145	1:02:46
6	0.017961	0.017753	0.271897	0.995130	1:02:23
7	0.018152	0.017668	0.275632	0.995126	1:02:21
8	0.017328	0.017609	0.281768	0.995089	1:02:15
9	0.017757	0.017619	0.281911	0.995092	1:02:10

Рисунок 31. - Результаты обучения модели аттрибутов ResNet34

На первой модели ResNet34 видно что несмотря на то что accuracy очень высокая, F-мера показывает достаточно низкие значения. Так для этой модели после 10 итераций обучения она составила всего 28%.

Обучение ResNet50:

learn = vision_learner(dls, resnet34, loss_func=BCEWithLogitsLossFlat(thresh=0.2), metrics=metrics, opt_func=opt_func).to_fp16()

learn.fine_tune(10)

Результаты обучения модели:

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.022208	0.021149	0.111762	0.995295	1:02:36

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.020632	0.019708	0.164142	0.995259	1:03:04
1	0.019816	0.018884	0.211681	0.995123	1:03:24
2	0.019216	0.018411	0.233966	0.995173	1:03:23
3	0.019238	0.018101	0.249580	0.995189	1:03:07
4	0.018445	0.017814	0.262712	0.995188	1:03:54
5	0.018589	0.017647	0.274475	0.995133	1:03:12
6	0.018073	0.017499	0.284271	0.995089	1:03:17
7	0.017618	0.017395	0.287307	0.995125	1:03:11
8	0.017560	0.017363	0.290061	0.995119	1:03:47
9	0.017623	0.017367	0.290444	0.995111	1:03:14

Рисунок 32. - Результаты обучения модели аттрибутов ResNet50

Для модели ResNet50 результаты оказались немного лучше 29%, но этого все ещё недостаточно, чтобы сохранять модель и использовать её в приложении.

Обучение MobileNet:

```
learn = vision_learner(dls, mobilenet_v3_large,
loss_func=BCEWithLogitsLossFlat(thresh=0.2), metrics=metrics, opt_func=opt_func).to_fp16()
learn.fine_tune(10)
```

Результаты обучения модели:

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.022926	0.021417	0.113747	0.995234	57:25

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.021615	0.020629	0.126581	0.995288	57:54
1	0.020960	0.020110	0.144832	0.995245	58:10
2	0.020556	0.019726	0.152016	0.995297	58:26
3	0.020323	0.019398	0.175629	0.995167	58:17
4	0.020219	0.019208	0.180437	0.995222	58:25
5	0.019863	0.019083	0.185756	0.995223	58:22
6	0.019595	0.018963	0.193515	0.995201	58:25
7	0.019617	0.018929	0.193022	0.995224	58:35
8	0.019532	0.018887	0.195464	0.995208	58:29
9	0.019471	0.018886	0.196178	0.995208	58:32

Рисунок 33. - Результаты обучения модели аттрибутов MobileNet

Модель MobileNet показала наихудшие результаты среди всех моделей, всего 19%.

Для наглядности сведём все данные в таблицу:

Таблица 3. - Результаты обучения моделей атрибутов

Модели	ResNet34	ResNet50	MobileNet
F-мера	0.281	0.29	0.196
Потери на валидации	0.0176	0.0173	0.0188

Таким образом лучшей оказалась модель ResNet50. Для улучшения качества обучения было решено попробовать такие оптимизаторы как Adam, RMSProp, а также различны функции потерь FocalLoss, BCEWithLogits-LossFlat, а также те функции которые подходят для бинарной классификации переделаны для многоклассовой классификации.

Импорт дополнительных классов:

```
import torch
import torch.nn.functional as F
from torch import nn
```

Функция для многоклассовой кросс-энтропии:

```
class CustomCrossEntropyLoss(nn.Module):
    def __init__(self):
        super(CustomCrossEntropyLoss, self).__init__()

    def forward(self, preds, targets):
        # Targets are in one-hot encoding
        log_preds = F.log_softmax(preds, dim=1)
        loss = -torch.sum(log_preds * targets, dim=1).mean()
        return loss
```

Это функция потерь для многометочной классификации со сглаживанием весов. Часть кода взята с Kaggle [8]:

```
class CustomLabelSmoothing(BCEWithLogitsLossFlat):
    def __init__(self, smoothing: float = 0.1, thresh: float = 0.2, **kwargs):
        super().__init__(thresh=thresh, **kwargs)
        self.smoothing = smoothing
        self.thresh = thresh

    def __call__(self, inp: torch.Tensor, targ: torch.Tensor, **kwargs) -> torch.Tensor:
        targ_smooth = targ.float() * (1. - self.smoothing) + 0.5 * self.smoothing
        loss = super().__call__(inp, targ_smooth, **kwargs)
        return loss

    def predict(self, inp: torch.Tensor) -> torch.Tensor:
        return (torch.sigmoid(inp) > self.thresh).float()

    def __repr__(self) -> str:
        return f"{self.__class__.__name__}(smoothing={self.smoothing},
thresh={self.thresh})"
```

Каждую модель была обучена на 3 итерациях. Обучение составило около 40 часов. Результаты обучения сведены в таблице 4:

Таблица 4. - F-мера после обучения с различными оптимизаторами в функциями потерь.

Функция потерь			BCEWithLogitsLossFlat	FocalLoss	CustomCrossEntropyLoss	CustomLabelSmoothing
Оптимизатор	Ranger	F-мера	0.2247	0.1317	0.1351	0.1699
		accuracy	0.9951	0.8626	0.8586	0.9952
	Adam	F-мера	0.2204	0.1877	0.1885	0.1605
		accuracy	0.9952	0.9107	0.9121	0.9952
	RMSProp	F-мера	0.2579	0.2041	0.2032	0.2108
		accuracy	0.9952	0.9224	0.9214	0.9952

Согласно таблице, лучшей оказалась модель с функцией потерь BCEWithLogitsLossFlat и оптимизатором RMSProp. Она имеет наибольшие значения как у метрики F-мера так и у accuracy и составляет соответственно 25.8% и 99.5%

Чтобы улучшить качество модели, были также изменены параметры wd и thresh, которые отвечают за размер штрафа при обучении модели и определении того, с какой вероятностью считать предсказание за класс 1.

Было решено рассмотреть следующий диапазон значений для wd [5e-8,5e-7,5e-6,5e-5,5e-4] и thresh [0.1,0.15,0.2,0.25,0.05,0.075,0.125]. Как оказалось значения F-меры с увеличением thresh уменьшались, поэтому после значения порога в 0.25 было решено попробовать несколько вариантов в окрестности 0.1.

Сначала рассмотрим работу модели с различным порогом (таблица 5), выберем лучшую, а затем исследуем изменение метрик на wd (таблица 6) и также выберем лучшую. Обучение будет проводиться на 2 итерациях, для сокращения времени обучения. По результатам будет выбрана окончательная лучшая модель для которой будет выбрана скорость обучения в соответствии с пунктом 2.5.2 и окончательно обучать нейросеть.

Таблица 5. - Результаты обучения модели с различным порогом thresh

Порог thresh	0.1	0.15	0.2	0.25	0.05	0.075	0.125
F-мера	0.3305	0.2919	0.2616	0.2274	0.3513	0.3480	0.3118
accuracy	0.9932	0.9946	0.9951	0.9955	0.9877	0.9914	0.9941

Согласно полученным результатам, наибольшая оценка была получена при пороге в 0.05, однако модель делает слишком много неправильных предсказаний, то есть слишком смело даёт единицы, поэтому было решено повысить порог, приняв его за 0.125, которая даёт значение F-меры больше 0.3, однако позволит сделать модель более точной.

Теперь выбрав хороший порог, были рассмотрены результаты обучения модели с различным размером штрафа, параметром wd. Он добавляет к функции потерь дополнительный член, который штрафует большие значения весов модели, делая их более устойчивыми и помогая улучшить обобщающую способность модели. Он помогает контролировать сложность модели и уменьшить вероятность переобучения.

Таблица 6. - Результаты обучения модели с различным размером штрафа wd

wd	5e-9	5e-8	5e-7	5e-6	5e-5
F-мера	0.3127	0.3128	0.3118	0.3120	0.3125
accuracy	0.9941	0.9941	0.9941	0.9940	0.9941

Согласно этой таблице, значение метрик практически не меняется в зависимости от параметра wd. Это происходит из-за того что обучение происходило всего лишь на трёх итерациях, и модель просто не успела обучиться достаточно чтобы возник риск переобучения. Тем не менее было решено остановить выбор на wd = $5*10^{-8}$, поскольку модель достаточно сложная и необходимо избежать её переобучения.

Таким образом итоговая модель выглядит так:

```
BETA = 2
THRESH = 0.125
WD = 5e-8
metrics=[FBetaMulti(BETA, THRESH, average='samples'), partial(accuracy_multi, thresh=THRESH)]
learn_final = vision_learner(dls, resnet50, loss_func=BCEWithLogitsLossFlat(thresh=THRESH), metrics=metrics, opt_func=partial(RMSProp, wd=WD)).to_fp16()
learn_final.fine_tune(3)
```

Также для более высокого качества работы модели было применено обогащение данных при создании загрузчика изображений. В transforms, модели было позволено отражение изображения по вертикали и увеличение света и контрастности изображения:

```
dblock = DataBlock(
    blocks=(ImageBlock,MultiCategoryBlock),
    splitter=RandomSplitter(valid_pct=0.3,seed=42),
    get_x=ColReader('path'),
    get_y=ColReader('label',label_delim=','),
    item_tfms = Resize(224),
```

```
    batch_tfms=aug_transforms(size=224,
                              do_flip=True,
                              flip_vert=False,
                              max_lighting=0.2,
                              max_warp=0.2,
                              p_lighting=0.75)
)
dls = dblock.dataloaders(df, bs=64,shuffle=True)
```

Изначально создаётся объект vision_learner, который представляет собой модель ResNet50. В этом моменте все слои модели автоматически блокируются (замораживаются) для предотвращения изменения весов этих слоёв во время обучения.

Затем при вызове learn_final.fine_tune(3) начинается процесс fine-tuning, который включает обучение модели уже моем наборе данных. На этом этапе все слои модели остаются заблокированными (замороженными). Fine-tuning лишь немного корректирует уже обученную модель.

Финальные результаты работы модели с пока ещё замороженными весами:

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.022025	0.020800	0.179008	0.994225	57:12

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.019717	0.018927	0.253194	0.994359	57:10
1	0.018950	0.018036	0.296731	0.994221	57:24
2	0.018160	0.017780	0.310898	0.994124	57:32

Рисунок 34. - Результаты обучения финальной модели аттрибутов

Чтобы модель начала полноценно обучаться слои модели размораживаются и на 1 итерации рассматривается изменение скорости обучения.

```
learn_final.unfreeze()
lr_range = learn_final.lr_find()
lr_range
```

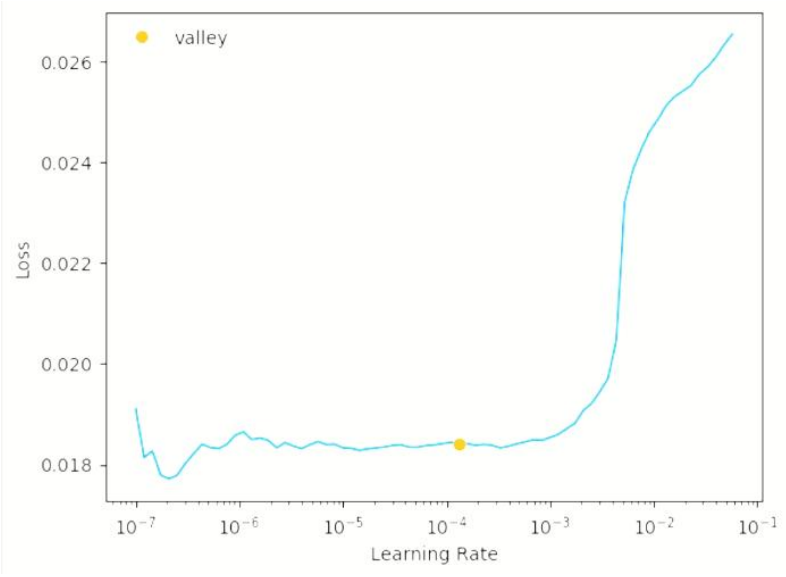


Рисунок 35. - Изменение скорости обучения финальной модели аттрибутов

Так как планируется улучшить уже обученную модель, фокусируясь на весах между точками минимума и роста потерь. Предварительно обученные слои хорошо справляются с базовыми визуальными понятиями, в то время как глубокие слои, отвечающие за сложные формы, предпочитают более высокую скорость обучения. Таким образом, будут использоваться разные скорости обучения для различных слоёв: меньшую для первых слоёв и большую для последних, чтобы обеспечить более эффективное настройку весов.

Продолжим обучение с помощью learn_final.fit_one_cycle(8) с параметром lr_max, которая устанавливает максимальную скорость во время обучения.

```
learn_final.fit_one_cycle(8, lr_max=slice(5e-7, 1e-4))
```

Финальные результаты обучения модели:

epoch	train_loss	valid_loss	fbeta_score	accuracy_multi	time
0	0.018392	0.017735	0.314406	0.994044	57:52
1	0.018160	0.017692	0.316317	0.994046	57:44
2	0.018040	0.017646	0.318400	0.994096	57:53
3	0.017895	0.017608	0.320373	0.994090	57:51
4	0.017820	0.017580	0.320977	0.994112	57:42
5	0.017977	0.017552	0.323314	0.994083	57:21
6	0.017754	0.017540	0.324085	0.994033	57:33
7	0.017637	0.017545	0.323660	0.994058	57:27

Рисунок 36. - Результаты обучения финальной модели аттрибутов с fit_one_cycle

Был построен график зависимости потерь и шагов обучения, для него были построены линии предсказаний на тренировочной и валидационной выборках. Этот график поможет понять есть ли необходимость продолжать обучение. Ведь если потери на валидации не улучшаются, а на тренировочной продолжают расти, возникает опасность переобучения модели. Если же показатели улучшаются на обоих выборках, обучение стоит продолжить.

learn_final.recorder.plot_loss()

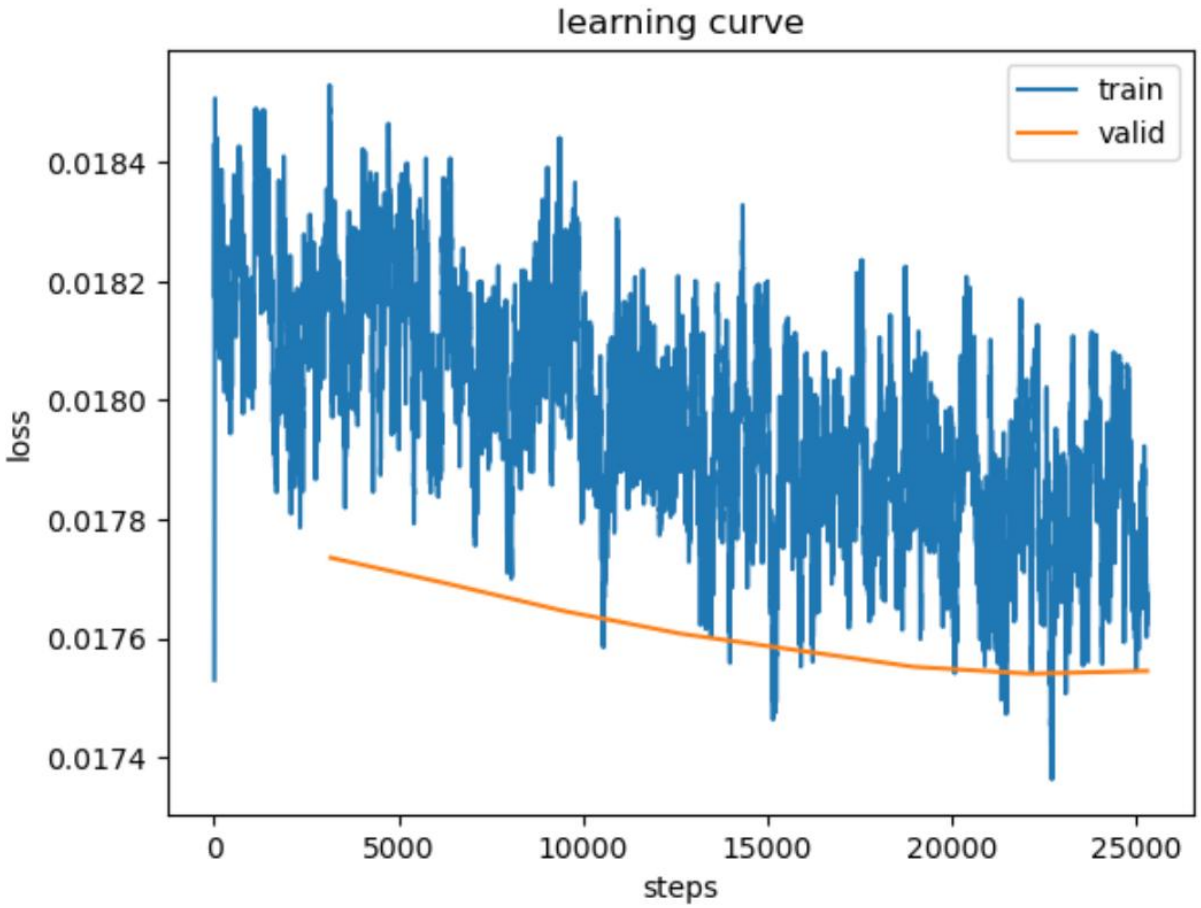


Рисунок 36. - График потерь для финальной модели аттрибутов

Отсюда видно, что потери на валидации начинают расти, хотя на тренировочной выборке все ещё могут продолжать уменьшаться. В таком случае следует прекратить обучение так как нет смысла рисковать и переобучать модель, когда она не даст лучшие предсказания на валидации.

Тестирование работы модели:

learn_final.show_results()

Рисунок 37. - Тест финальной модели аттрибутов

Этот тест демонстрирует, что модель называет практически все аттрибуты которые имеют изображения, но также и добавляет свои. Так у платья аттрибуты 365,596,717,730 модель полностью угадала, однако добавила и свои. Например 358 - расширение к краю, очень даже подходит для данного платья, 839 приталенный крой, также отлично подходит для этого платья с чётко очерченной талией, а вот 90 трапециевидная форма и 892 слово лето уже не так хорошо относятся к данному платью. Таким образом чтобы улучшить работу модели, следует повысить порог предсказаний, чтобы модель не предсказывала лишние аттрибуты, либо её можно оставить как есть. Было решено сохранить модель как есть. Так как она даёт достаточно близкие к реальности предсказания.

Сохраним нашу модель:

```
model = learn_final.model
model.cpu()
torch.save(model, 'attr_resnet50.pkl')
learn_final.export("attr_resnet50_export.pkl")
with open('attr_resnet50_open.pkl', 'wb') as file:
    pickle.dump(learn_final, file)
```

3 Разработка компьютерного приложения

Прежде чем приступить к написанию приложения, необходимо выбрать библиотеки для его написания. Было решено использовать библиотеки python, чтобы обеспечить полную совместимость с реализованными нейросетями. Среди таких:

PySide2: Это официальные привязки Python к Qt, предоставляющие доступ к широкому спектру инструментов и библиотек для быстрого и гибкого создания

пользовательских интерфейсов. PySide2 позволяет разработчикам использовать API Qt в своих приложениях Python, что делает его мощным инструментом для создания сложных GUI 1.

PyQt5: Одной из самых популярных библиотек для создания GUI на Python является PyQt5, которая построена вокруг фреймворка Qt. PyQt5 предлагает модули QtGUI и QtDesigner, позволяя разработчикам реализовывать визуальные элементы с помощью перетаскивания или кодирования. Эта библиотека поддерживает множество платформ, включая Mac, Windows, Linux, iOS и Android, что делает её универсальным решением для разработки приложений различных масштабов 14.

wxPython: Эта библиотека упрощает процесс создания нативного интерфейса без добавления лишнего веса к приложению. С помощью wxPython можно создавать приложения для Linux, macOS, UNIX и Windows. Библиотека включает большое количество виджетов, которые выглядят отлично на всех поддерживаемых платформах без необходимости их настройки 4.

PyForms: Это открытая и кроссплатформенная библиотека, которая позволяет легко создавать приложения для нескольких платформ без значительных изменений в коде. PyForms может быть разделен на три разных раздела: PyForms-GUI, PyForms-Web и PyForms-Terminal, каждый из которых обеспечивает выполнение приложения PyForms как в Windows, так и в веб-режиме или терминале. Это делает PyForms удобным выбором для разработки интерактивных интерфейсов для различных сред.

Выбор был остановлен на PyQt5, так как помимо кроссплатформенности оно предоставляет обширное сообщество и дополнительные инструменты для упрощения написания кода.

Прежде всего для разработки компьютерного приложения необходимо было создать новую виртуальную среду с теми же версиями библиотек pytorch и fastai и python, а также дополнительно установить gtts, pygame, pyqt5, pyqt5-sip

3.1 Разработка интерфейса компьютерного приложения

Для разработки приложения на компьютере была использована в качестве основной библиотеки PyQt5. Это простая библиотека с набором виджетов, которая позволяет быстро и просто написать небольшие приложения.

Импорт библиотек:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QLabel
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt,QTimer
from PyQt5.QtMultimedia import QCamera, QCameraImageCapture
from PyQt5.QtMultimediaWidgets import QCameraViewfinder
import pygame
import gtts
import torch.serialization
import fastai.callback.progress
import fastai.callback.fp16
import fastai.vision.learner
from fastai.learner import load_learner
from PIL import Image
import pandas as pd
import os
```

Среди библиотек приложения здесь также есть pygame и gtts которые отвечают за воспроизведение аудио и его сохранение. Также библиотека os которая позволяет работать с операционной системой. В нашем случае она была нужна для замены старого аудио на новое.

Следующее - класс отвечающий за само приложение.

Он будет состоять из основного окна, раздела с камерой и фото и тремя кнопками - сделать снимок, описать снимок и выход из приложения.

```
class CameraApp(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
        self.clickCount = 0
        self.clickExitCount = 0
        self.clickDesCount=0
        self.currentView = None
```

```
def initUI(self):
    self.setWindowTitle('Приложение для захвата фотографии')
    self.setGeometry(100, 100, 1100, 800)

    self.captureButton = QPushButton('Сделать \nснимок', self)
    self.captureButton.clicked.connect(self.captureImage)
    self.captureButton.setFixedSize(270,100)
    self.captureButton.move(820,260)

    self.describeButton = QPushButton('Описать \nснимок', self)
    self.describeButton.clicked.connect(self.describeCloth)
    self.describeButton.setFixedSize(270,100)
    self.describeButton.move(820,370)

    self.exitButton = QPushButton("Выход", self)
    self.exitButton.clicked.connect(self.exitDef)
    self.exitButton.setFixedSize(270,100)
    self.exitButton.move(820,480)

    self.cameraLabel = QLabel(self)
    self.cameraLabel.setGeometry(800, 780, 800, 780)
    self.cameraLabel.move(10,10)
    self.cameraLabel.setAlignment(Qt.AlignCenter)

    self.photoLabel = QLabel(self)
    self.photoLabel.setGeometry(800, 780, 800, 780)
    self.photoLabel.move(10,10)
    self.photoLabel.setAlignment(Qt.AlignCenter)

    # Установка цвета рамки
    self.setStyleSheet("""QPushButton
                {background-color: #4080FF;
                color: #ffffff;
```

```
                font-size: 40px;
                font-weight: bold;
                border-radius: 10px;
                }""")

        self.camera = QCamera()
        self.imageCapture = QCameraImageCapture(self.camera)
        self.imageCapture.imageCaptured.connect(self.displayImage)
        self.viewfinder = QCameraViewfinder(self.cameraLabel)
        self.viewfinder.setFixedSize(self.cameraLabel.size())
        self.camera.setViewfinder(self.viewfinder)
        self.camera.start()
```

Также с помощью функции self.setStyleSheet() и языка CSS для кнопок было сделано оформление, назначен цвет и размер текста, хотя для слепых людей это не важно, это важно для слабовидящих.

Метод captureImage отвечает за захват и изображения с камеры

```
 def captureImage(self):
        self.imageCapture.capture()
```

Метод displayImage сохраняет изображение и размещает его в главном окне. Также заранее были подготовлены аудио, которое озвучивает нажатие на кнопки и выполнение операции.

```
 def displayImage(self, id, preview):
        if self.clickCount == 0:
            play_text('voice')
            self.clickCount = 1
        else:
            image_jpg = QPixmap(preview)
            image_jpg.save('captured_image.jpg')
            self.photoLabel.setPixmap(image_jpg)
            play_text('itog')
            QTimer.singleShot(2000, lambda: self.photoLabel.clear())
            self.clickCount=0
```

```
            self.camera.start()
```

Метод describeCloth вызывает функцию предсказания для нейросетей, а также сохранения нового аудио и воспроизведение его. Этот процесс самый долгий из всех и может занять у программы несколько секунд, поэтому после нажатия кнопки пользователь будет осведомлен о начале работы приложения.

```
    def describeCloth(self):
        if self.clickDesCount == 0:
            play_text('des_but')
            self.clickDesCount=1
        else:
            play_text('neuro_start')
            neuro_output()
            play_text('last_des')
            self.clickDesCount=0
```

Последний метод класса exitDef по нажатию кнопки выходит из приложения.

```
    def exitDef(self):
        if self.clickExitCount == 0:
            play_text('exit_but')
            self.clickExitCount=1
        else:
            play_text('you_exit')
            self.clickExitCount=0
            self.close()
```

Переходим к прочим функциям необходимым для работы:

Функция save_voice переводит текст в аудио и сохраняет его в формате mp3 с заданным именем.

```
    def save_voice(text,name):
        if os.path.exists(name+'.mp3'):
            os.remove(name+'.mp3')
        language = 'ru'
        speech = gtts.gTTS(text=text, lang=language, slow=False)
        speech.save(name+'.mp3')
```

Функция play_text воспроизводит текст по заданному пути

```
def play_text(path):
    pygame.mixer.init()
    pygame.mixer.music.load(path+'.mp3')
    pygame.mixer.music.play()
```

Главная функция neuro_output сначала открывает сохранённое ранее изображение, затем приводит его в необходимый для модели формат, загружает модели с категориями и аттрибутами, делает предсказание для этого фото и отправляет на вход функции output_to_test предсказанные значения аттрибутов и категорию, возвращая полноценный текст, с достаточно подробным описанием, затем она сохраняет его в формате mp3.

```
def neuro_output():
    img=Image.open(r'captured_image.jpg')
    img.thumbnail((224,224),Image.LANCZOS)
    cat_model = load_learner(r'cat_resnet34_export.pkl')
    attr_model = load_learner(r'D:\jupyter\Diploma\FinalFolder\cat_resnet34_export.pkl')
    category = int(cat_model.predict(img)[0])-1
    attributs = attr_model.predict(img)[0]
    attributs = [int(attr) for attr in attributs.split(',')]
    decription = output_to_test(category,attributs)
    save_voice(decription,'last_des')
```

Последняя функция перевода категорий и аттрибутов в текст. Она подтягивает csv файлы с описанием предсказанных значений, а по заданным числам ищет перевод на русский язык (при желании можно оставить английский). После чего склеивает все в единое предложение. Также оно удаляет пустой аттрибут 1000 если его предсказывает модель и удаляет повторяющиеся аттрибуты. Например такие как stripe и stripes.

```
def output_to_test(category,attributs):
    attr_cloth = pd.read_csv('attr_cloth.csv', sep=';')
    category_cloth = pd.read_csv('category_cloth.csv',sep=';')
    decription_cloth = 'Это '
    decription_cloth += str(category_cloth.loc[category, 'russian'])
    decription_cloth+=' имеет '
    for attr in attributs:
```

```
        if int(attr)==1000:
            continue
        decription_cloth+=str(attr_cloth.loc[int(attr), 'russian'])
    phrases = decription_cloth.split(', ')
    unique_phrases = list(set(phrases))
    unique_phrases.sort(key=phrases.index)
    decription_cloth = ', '.join(unique_phrases)
    decription_cloth+=', '    decription_cloth = decription_cloth[:-2] + '.'
    return decription_cloth
```

Приложение написано. Вывод приложения смотрите в Приложении 1

3.2 Конвертация компьютерного приложения в формат .exe

Для того чтобы конвертировать наш ру файл в полноценное приложение необходимо немного изменить программу. Воспользуемся инструкцией [9].

Во первых, после импорта библиотек добавляется дополнительная строка, которая подключит файл с ресурсами для нашего приложения:

```
import resources
```

Во вторых, в той же папке, где находится исходный код необходимо создать текстовый файл и прописать в нем следующие строки:

```
<RCC>
  <qresource prefix="newPrefix">
    <file>attr_cloth.csv</file>
    <file>category_cloth.csv</file>
    <file>captured_image.jpg</file>
    <file>cat_resnet34_export.pkl</file>
    <file>attr_resnet50_export.pkl</file>
    <file>voice.mp3</file>
    <file>itog.mp3</file>
    <file>exit_but.mp3</file>
    <file>you_exit.mp3</file>
    <file>des_but.mp3</file>
    <file>neuro_start.mp3</file>
    <file>last_des.mp3</file>
```

```
  </qresource>
</RCC>
```

Так приложение поймёт какие именно файлы необходимо загружать. Также, чтобы наше App приложение смогло прочитать этот файл rcc, необходимо в терминале прописать следующую строку,чтобы конвертировать её в формат py:

```
pyrcc5 -o resources.py resources.qrc
```

Наконец, предварительно загрузив в виртуальную среду библиотеку pyinstaller, с помощью команды создадим приложение на основе нашего App.py файла.

```
pyinstaller --onefile App.py
```

Эта библиотека создаст 2 папки dist и build. В первой из которых будет лежать наше приложение. Теперь его можно использовать в работе, однако необходимо помнить, что в папке с приложением должны лежать все связанные файлы.

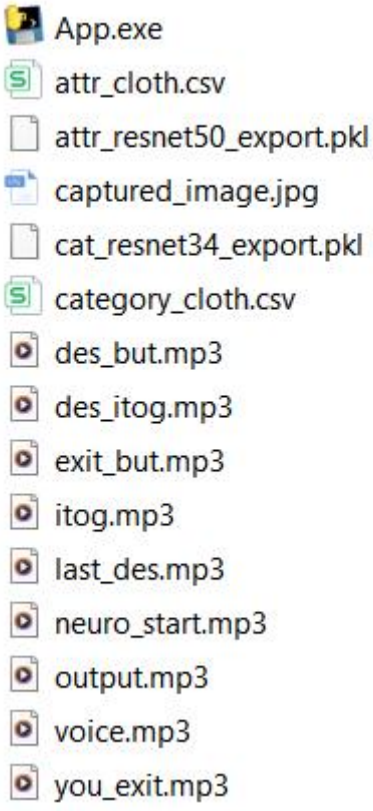


Рисунок 38. - Компьютерное приложение

Заключение

В процессе выполнения выпускной квалификационной работы было разработано компьютерного приложения для распознавания категории и аттрибутов одежды на изображении, были выполнены следующие задачи:

- Обучена нейросеть для распознавания категорий одежды
- Обучена нейросеть для распознавания аттрибутов одежды
- Написано компьютерное приложение включающее обученные нейросети
- Реализованы функции озвучки кнопок и выхода с моделей нейросетей
- Python приложение было сконвертировано в формат .exe

Список литературы

1. YouTube. Blind People Tell Us If They Wish They Could See | Blind People Describe. [Электронный ресурс]. – URL: https://www.youtube.com/watch?v=RBgr0GfkIsM (дата обращения: 07.06.2024).

2. CUHK Lab. DeepFashion: An Enhanced Database for Human Pose Estimation in Casual Clothing. [Электронный ресурс]. – URL: https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html (дата обращения: 07.06.2024).

3. YouTube. How to Train a Neural Network for Fashion Classification. [Видео]. – URL: https://www.youtube.com/watch?v=r31jnE7pR-g (дата обращения: 21.05.2024).

4. YouTube. Introduction to Convolutional Neural Networks (CNNs). [Видео]. – URL: https://www.youtube.com/watch?v=_z6WPasIK8U (дата обращения: 21.05.2024).

5. Шолле Франсуа. Глубокое обучение на Python. 2-е межд. издание. — СПб.: Питер, 2023. — 576 с.: ил. — (Серия «Библиотека программиста»).

6. Ховард Джереми, Гуггер Сильвейн. Глубокое обучение с fastai и PyTorch: минимум формул, минимум кода, максимум эффективности. — СПб.: Питер, 2022. — 624 с.: ил. — (Серия «Бестселлеры O'Reilly»).

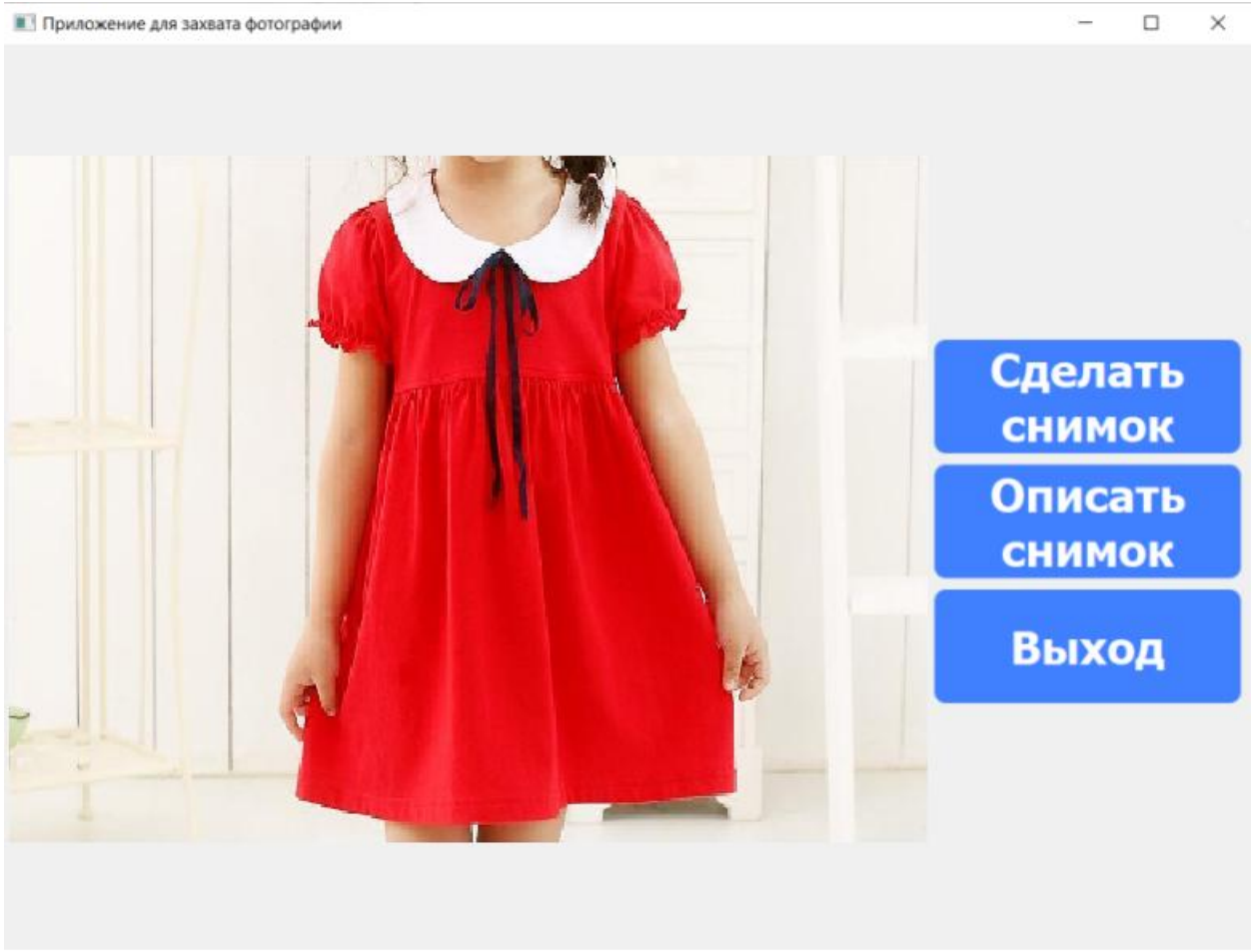
7. Курс «Специалист по Data Science» // Яндекс Практикум [Электронный ресурс]. – URL: https://practicum.yandex.ru/data-scientist/?from=catalog (дата обращения: 09.06.2024).

8. Kaggle. Pytorch Label Smoothing Implementation - Help Needed! [Электронный ресурс]. – URL: https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/166833#929222 (дата обращения: 06.06.2024)

9. Хабр Q&A. Как конвертировать .py (pyqt5) в .exe с изображениями? [Электронный ресурс]. – URL: https://qna.habr.com/q/1187316 (дата обращения: 07.06.2024)

Приложение 1

Интерфейс desktop приложения



Это Платье имеет, ткань шифон, воротник, кружево, длинный воротник с закруглёнными концами, красный цвет, часть являющейся рубашкой, рукава, бант.