МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

18.05.23

КУРСОВАЯ РАБОТА

СОЗДАНИЕ СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА ДЛЯ ФОРМИРОВАНИЯ СОБИРАТЕЛЬНОГО ПОРТРЕТА

Работу выполнил _____ (подпись) П.Л. Бандуровский

Направление подготовки 09.03.03 Прикладная информатика

Направленность (профиль) Прикладная информатика в экономике

Научный руководитель
д-р. техн. наук, зав. каф., _____ (подпись) А.В. Коваленко

Нормоконтролер
канд. физ.-мат. наук, доц. _____ (подпись) Г.В. Калайдина

Краснодар
2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

КУРСОВАЯ РАБОТА

СОЗДАНИЕ СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА ДЛЯ ФОРМИРОВАНИЯ СОБИРАТЕЛЬНОГО ПОРТРЕТА

Работу выполнил _____П.Л. Бандуровский
(подпись)

Направление подготовки 09.03.03 Прикладная информатика

Направленность (профиль) Прикладная информатика в экономике

Научный руководитель
д-р. техн. наук, зав. каф., _____ А.В. Коваленко
(подпись)

Нормоконтролер
канд. физ.-мат. наук, доц. _____ Г.В. Калайдина
(подпись)

Краснодар
2023

РЕФЕРАТ

Курсовая работа 37с., 12 рис., 16 источников, 1 приложение.

ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНАЯ НЕЙРОННАЯ СЕТЬ, DCGAN, СОБИРАТЕЛЬНЫЙ ПОРТРЕТ, PYTHON, PYTORCH, СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ, БАЗА ДАННЫХ CELEBA

Объектом исследования является разработка нейронной сети с помощью Python и библиотеки PyTorch.

Цель работы: создание системы искусственного интеллекта для формирования собирательного портрета.

В результате курсовой работы были изложены основные понятия, связанные с данной темой и рассмотрены методы для создания различных нейронных сетей, классификации нейронных сетей. Были изучены основные архитектуры генеративно-состязательной модели нейронной сети и создана нейронная сеть на основе DCGAN архитектуры для формирования собирательного портрета с помощью языка программирования Python и библиотеки PyTorch.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С момента появления искусственных нейронных сетей (ИНС) их применяют для решения различных прикладных задач. ИНС успешно используются в широчайшем спектре приложений, таких как распознавание образов, генерация изображений, прогнозирование, выявление зависимостей и многие другие.

Возможность нейронных сетей работать с нелинейными данными делает их более эффективными, чем обычные программы

Генерация данных нейронными сетями – это процесс создания новых текстов, изображений, видео или аудиофайлов с помощью алгоритмов машинного обучения.

Нейронные сети могут быть обучены на огромных объемах данных, что позволяет им научиться распознавать и повторять стили и образцы, присутствующие в этих данных. Например, нейронные сети могут быть обучены распознавать образцы в языке и генерировать новые изображения, которые могут казаться очень похожими на настоящие.

Генерация данных нейронными сетями может быть полезна во многих областях, например, в маркетинге и рекламе, где создание оригинального и интересного контента является ключевым фактором в привлечении новых клиентов и удержании существующих.

Кроме того, генерация контента нейронными сетями может быть полезна в научных исследованиях, где исследователи могут использовать нейронные сети для генерации новых данных, которые могут помочь в разработке новых технологий и продуктов.

В связи с вышеизложенным, тема курсовой работы, направленная на создание системы искусственного интеллекта для формирования собирательного образа, является актуальной.

Актуальность данной работы обусловлена также широким применением подобных технологий и, вследствие этого, большой популярностью на рынке программ, распознающих и генерирующих образы.

Целью работы является создание системы искусственного интеллекта для формирования собирательного образа, а именно разработка генеративно-состязательной нейронной сети с использованием языка программирования Python и библиотеки PyTorch. Данная цель определила следующие задачи:

– Подготовка и обработка данных для обучения модели.

– Исследование различных архитектур GAN модели нейронных сетей.

– Разработка архитектуры GAN модели, которая будет использоваться для генерации собирательного портрета.

– Обучение GAN модели на обучающей выборке.

Структура курсовой работы состоит из трех глав, а также из одного приложения, в котором представлен код нейронной сети.

В первой главе курсовой работы представлен теоретический анализ предметной области, методы обучения нейронных сетей, рассмотрены основные типы задач нейросетевых технологий, а также различные архитектуры генеративно состязательных нейронных сетей.

Во второй главе описаны основные программные средства реализации нейронных сетей, дана краткая характеристика языка программирования Python, описаны основные библиотеки, рабочая среда, обучающая выборка. Представлены различные архитектуры нейронных сетей, такие как генеративно-состязательные и сверточные нейронные сети. Подробно рассмотрена DCGAN модель нейронной сети. Описаны активационные функции, алгоритм оптимизации Адам.

В третьей главе представлена разработка нейронной сети «FaceForge» для формирования собирательного портрета

В заключении подведен основной итог работы.

1 Теоретический анализ нейронных сетей

Программа, имитирующая модель человеческих нейронных связей, называется искусственной нейронной сетью. Этот тип программы используется для создания обучаемых систем, которые способны распознавать или генерировать контент [1].

Работа нейронной сети не похожа на работу обычной программы. Ей необходимо быть обученной, чтобы выполнять задачи самостоятельно. В результате деятельность программы становится менее предсказуемой, но более вариативной [2].

Благодаря такому подходу современные мощные нейронные сети могут рисовать картины, создавать стихи и отвечать на сложные вопросы. Они широко используются в программных продуктах различных областей, от роботов-помощников до сложных медицинских систем диагностики [3].

Нейронные сети можно классифицировать по разным критериям. Рассмотрим некоторые из них:

1) По типу обучения:

– Обучение с учителем, когда модель настраивает свои веса на минимальную ошибку, используя правильные ответы на входе.

– Обучение без учителя, когда модель самостоятельно находит закономерности в данных без заранее известных правильных ответов [4].

2) По архитектуре нейронные сети делятся на несколько типов:

– Прямые сети, где данные проходят через сеть в одном направлении от входа к выходу без обратных связей. На рисунке 1 показана схема прямой нейронной сети

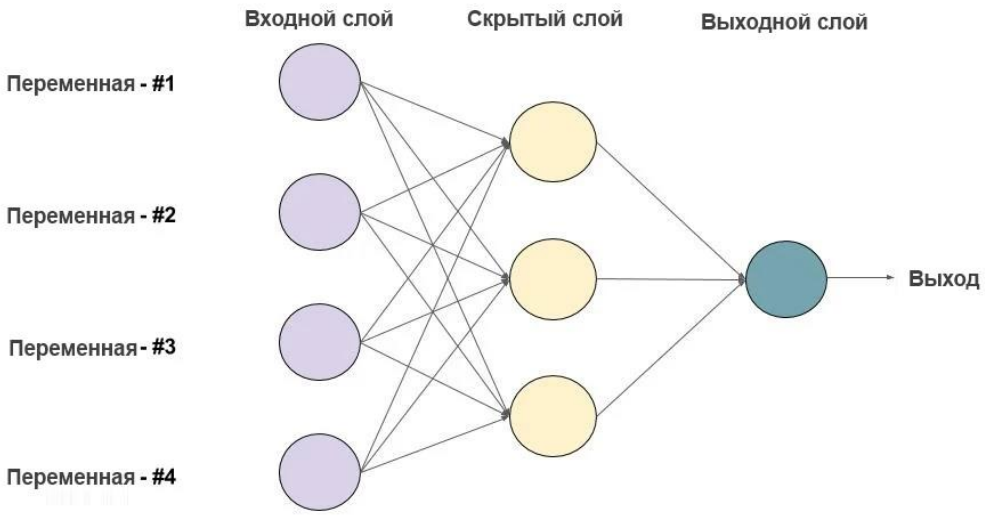


Рисунок 1 – Схема прямой нейронной сети [5]

– Рекуррентные сети, где имеются обратные связи, и информация может циркулировать по сети. Этот тип сетей применяется для анализа последовательностей, таких как тексты, речь или временные ряды. На рисунке 2 показана схема рекуррентной нейронной сети

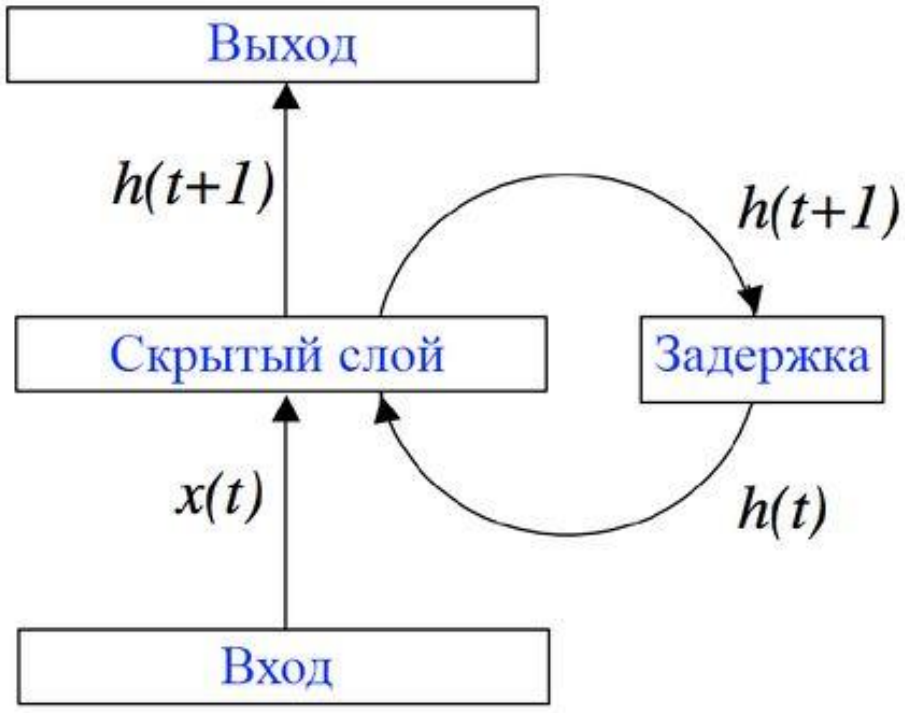


Рисунок 2 – Схема рекуррентной нейронной сети[6]

– Сверточные сети, которые специализированы для обработки изображений и других типов сигналов, таких как звуковые волны. На рисунке 3 показана схема сверточной нейронной сети

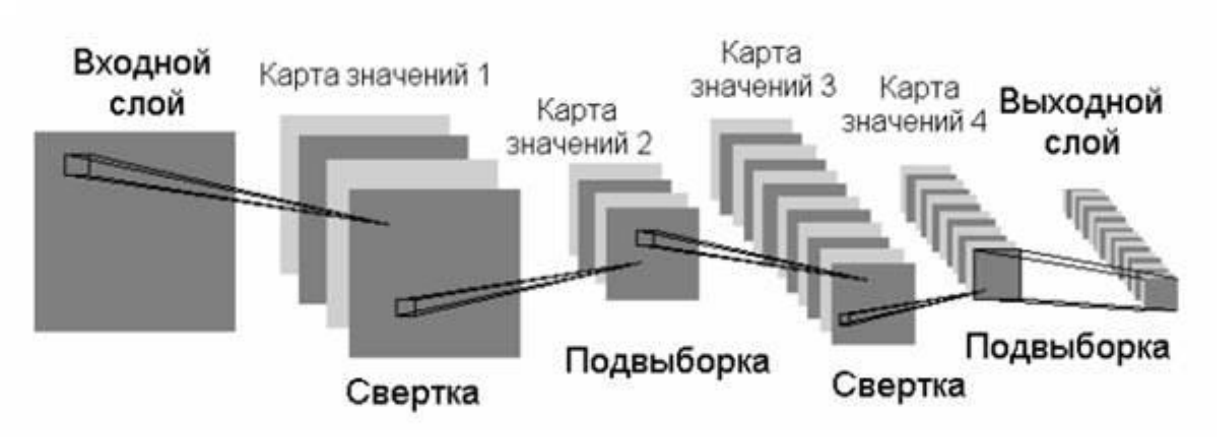


Рисунок 3 – Схема сверточной нейронной сети[6]

3) По количеству слоев:

– Однослойные, где сеть состоит только из одного слоя.

– Многослойные, где сеть состоит из нескольких слоев, которые выполняют последовательную обработку данных. Многослойные сети могут иметь различные архитектуры и использоваться для различных задач.

4) По функции активации:

– Линейные, где функция активации является линейной, и сеть работает как линейный регрессор.

– Нелинейные, где функция активации является нелинейной, и сеть способна моделировать сложные нелинейные зависимости в данных.

Классификация нейронных сетей по признакам помогает в выборе оптимальной модели для конкретной задачи.

а) Задача классификации, которая используется для распознавания различных объектов, таких как лица, животные или эмоции. Кроме того, классификация может быть использована для распознавания образов, то есть

выбора конкретного объекта из предложенного множества, например классификация изображения кошки среди изображений собак.

б) Задача регрессии используется для предсказания численного значения на основе заданных признаков. Она может применяться для оценки возраста на основе фотографии, прогнозирования биржевых курсов, определения стоимости имущества и многих других задач. Регрессионные модели обучаются на обучающих данных, которые содержат признаки и соответствующие значения целевой переменной. После этого модель может использоваться для предсказания значений целевой переменной для новых данных, которые содержат только признаки.

в) Задача кластеризации используется для группировки данных на основе их признаков. Кластеризация может быть полезной, когда количество классов на выходе неизвестно. Например, кластеризация может применяться для выявления групп картинок или для сегментации клиентов по сходству их поведения на сайте. Алгоритмы кластеризации обучаются на данных, не размеченных по классам, и пытаются найти скрытые закономерности и сходства между ними.

г) Задача генерации используется для автоматического создания контента или его модификации. Нейронные сети могут использоваться для генерации уникальных текстов, звуков, видео и даже для раскрашивания черно-белых изображений или изменения фона фотографий.

1.1 GAN

Одной из распространенных моделей для генерации контента является генеративно-состязательная нейронная сеть (GAN). GAN состоит из двух нейросетей: генератора и дискриминатора. Задача генератора состоит в том, чтобы генерировать образы заданной категории, а задача дискриминатора – пытаться распознать, является ли созданный образ реалистичным. Таким

образом, генератор генерирует определенные образы, а дискриминатор пытается определить, являются ли эти образы реальными [7].

Сеть обучается до тех пор, пока генератор не начнет генерировать весьма реалистичные изображения. На рисунке 4 представлена общая схема генеративно-состязательной нейронной сети

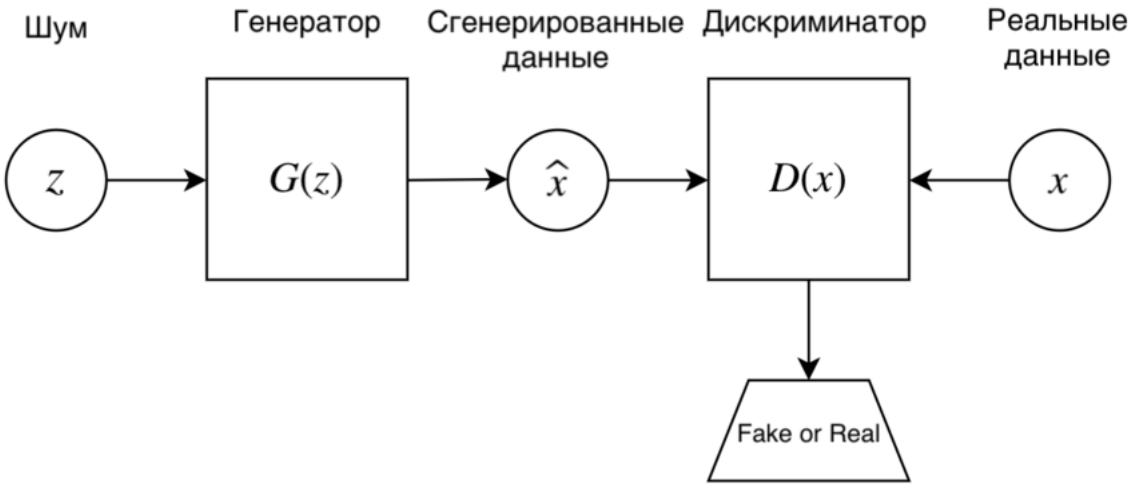


Рисунок 4 – Схема работы GAN архитектуры [7]

Генератор начинает формирование изображения с генерации случайного шума, на который постепенно начинают проступать фрагменты искомого изображения.

Дискриминатор пытается классифицировать входные данные. Учитывая особенности полученных данных, он стараются определить категорию, к которой они относятся. Таким образом, дискриминационные функции сопоставляют образы с категорией и используются только для этой корреляции.

Существует множество разновидностей GAN. Самые популярные из них:

1) DCGAN – использует сверточные слои как в генераторе, так и в дискриминаторе для генерации более качественных изображений.

2) CGAN – позволяет контролировать генерацию изображений, вводя дополнительную информацию в виде условия.

3) CycleGAN – используется для преобразования изображений из одного стиля в другой, например, превращение изображений лошадей в изображения зебр.

4) Pix2Pix – позволяет преобразовывать изображения из одного типа в другой, например, перевод черно-белых изображений в цветные

5) SAGAN – использует механизм внимания для моделирования глобальных зависимостей в изображении.

Каждая из этих разновидностей GAN имеет свои особенности и применяется в различных задачах [7].

Результат работы генеративно-состязательных нейронных сетей упрощает процесс создания контента. С помощью их помощью можно создавать картинки для интернет-магазина, аватаров для игр, видеоклипы, сгенерированных автоматически, исходя из музыкального бита произведения, или даже виртуальных ведущих для ТВ-программ.

Благодаря работе GAN и генеративных моделей возникает синтез данных, на которых потом будут обучаться другие системы.

Таким образом, генеративно-состязательные нейронные сети актуальны на сегодняшний день. Данные, созданные нейронными сетями, применяются в различных сферах деятельности, поэтому тема курсовой работы, посвященная созданию системы искусственного интеллекта для формирования собирательного портрета, является актуальной.

2 Основные программные средства реализации нейронной сети

2.1 Язык программирования Python

Язык программирования Python является одним из самых популярных языков для разработки нейронных сетей, благодаря простоте, лаконичности и выразительности. Именно на Python есть библиотеки и для более простого написания нейронных сетей [8].

2.2 Основные библиотеки для разработки нейронных сетей

PyTorch – это фреймворк для машинного обучения, который разработан на языке программирования Python. Он предоставляет удобный интерфейс для работы с графами вычислений и реализации нейронных сетей. PyTorch предоставляет разработчикам мощный и гибкий инструментарий для создания сложных моделей нейронных сетей, обучения их на больших наборах данных и получения высококачественных результатов [9].

NumPy – это библиотека языка Python, которая предоставляет поддержку многомерных массивов и матриц, а также богатый набор функций для работы с ними. NumPy является одной из наиболее распространенных и эффективных библиотек для работы с научными вычислениями и анализом данных. Он предоставляет множество функций для работы с многомерными массивами данных. Для разработки нейронных сетей NumPy используется для эффективной работы с многомерными массивами данных, которые используются для хранения входных данных, весов и смещений модели, а также результатов промежуточных вычислений [10].

Matplotlib – это библиотека для визуализации данных в языке программирования Python. Она предоставляет широкие возможности для создания графиков, диаграмм, рисунков и других типов визуализации данных.

Это позволяет улучшить понимание процесса обучения и результатов работы нейронных сетей.

2.3 Рабочая среда

В качестве рабочей среды была выбрана среда Google Colaboratory. Это бесплатная среда, чтобы писать код в jupyter notebook. Она функционирует по принципу облака, поэтому над одним проектом могут работать одновременно несколько человек. Программа предоставляет доступ к графическим процессорам GPU и TPU. Благодаря их мощности можно исследовать искусственный интеллект и развивать приложения на основе нейронных сетей[11].

Colab позволяет использовать в одном файле исполняемый код, html-разметку, картинки. Этими файлами можно делиться: разрешать просматривать, редактировать и оставлять комментарии для совместной работы. Сервис помогает и с другими задачами:

1) сортировать данные,
2) строить визуализации,
3) проводить машинное обучение,
4) создавать системы для big data,
5) составлять прогнозы,
6) писать руководства.

2.4 Сверточные слои

Сверточные нейронные сети – это класс нейронных сетей, которые специализируются на обработке и анализе данных с пространственной структурой, таких как изображения, видео, звуковые волны.

Основным компонентом сверточной нейронной сети является сверточный слой, который применяет фильтры (ядро свертки) к входным

данным, выделяя изображение наиболее важные характеристики, такие как границы, текстуры и формы объектов. Затем происходит подвыборка, которая уменьшает размерность данных и сохраняет только наиболее значимые признаки.

Сверточные нейронные сети часто используются в задачах классификации изображений, где требуется определить, что находится на изображении, например, классификация изображений на котов и собак. Они также используются в обработке естественного языка, например, для распознавания и классификации текстов.

В целом, сверточные нейронные сети стали очень популярными и успешными в различных областях машинного обучения, так как они позволяют автоматически извлекать признаки из входных данных, не требуя ручного определения этих признаков.

2.5 Обучающая выборка

В качестве обучающей выборки была использована обучающая выборка CelebA – это широко используемый набор данных, содержащий более 200 000 изображений знаменитостей. Каждое изображение представляет собой цветное фото размером 178x218 пикселей. На рисунке 5 представлены примеры изображений из обучающей выборки CelebA

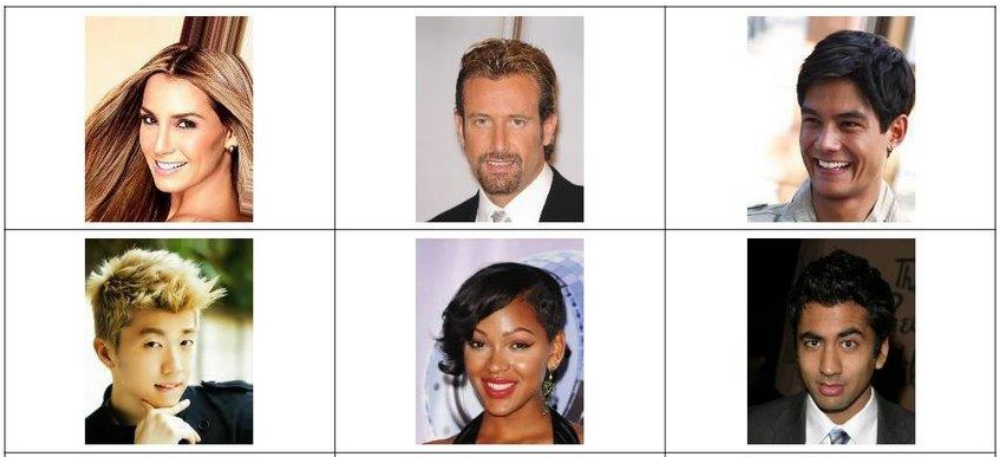


Рисунок 5 – Примеры изображений людей из базы данных CelebA

Каждый образец в CelebA содержит дополнительные метаданные, такие как различные атрибуты знаменитостей, например возраст, пол, цвет волос и глаз, наличие очков, усы и. Всего есть около 40 атрибутов, каждый из которых может быть использован в качестве условия для генерации изображения. Это позволяет использовать CelebA в качестве набора данных для задач условной генерации.

CelebA один из первых и наиболее известных наборов данных, используемых в глубоком обучении для задач генерации изображений. Он остается популярным и широко используется в научных и коммерческих проектах.

2.6 DCGAN

DCGAN– это модель глубокой нейронной сети, которая используется для генерации изображений, прежде всего, в компьютерном зрении. Она была разработана в 2015 году на базе оригинальной модели GAN, которая была представлена несколько лет ранее. На рисунке 6 изображена схема DCGAN.

DCGAN использует сверточные слои в генераторе и дискриминаторе вместо полносвязных слоев, что позволяет учитывать пространственную структуру изображений. В DCGAN генератор работает как обратный декодер, который преобразует скрытое представление в изображение. Дискриминатор же выступает в роли классификатора, который определяет, является ли данное изображение реальным или сгенерированным[13].

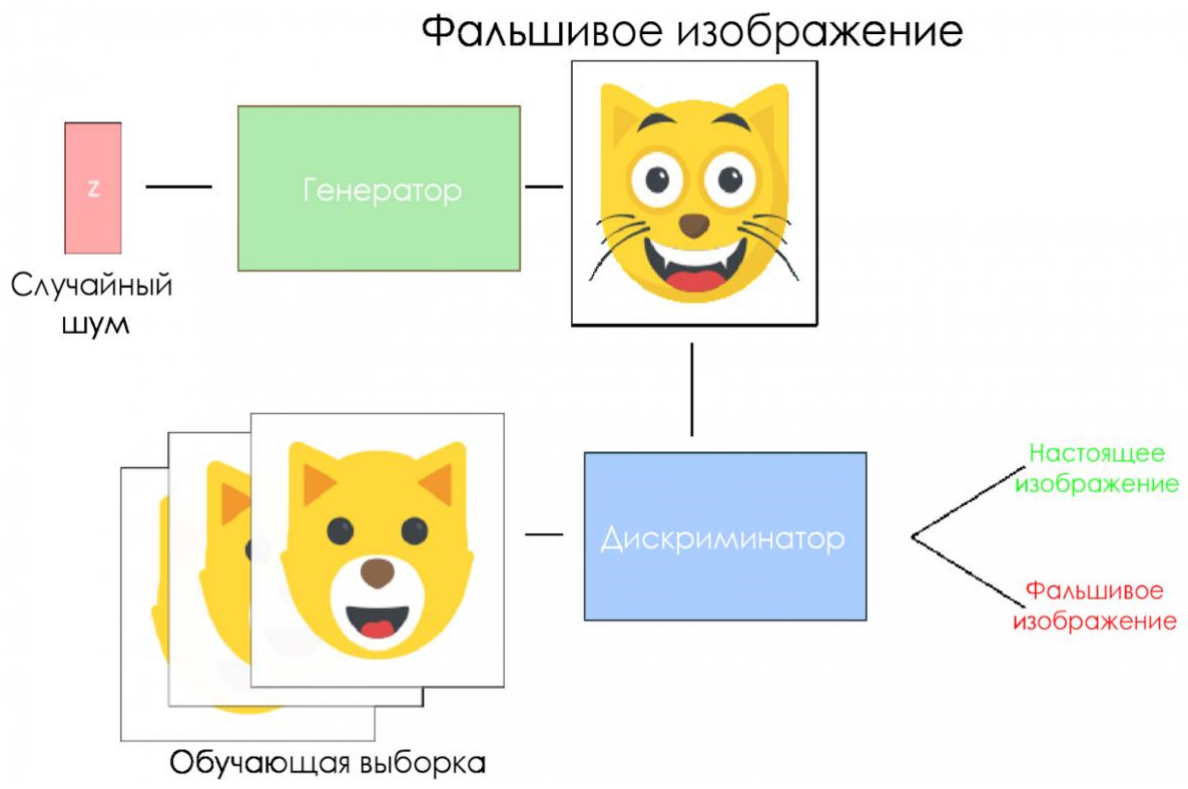


Рисунок 6 – Схема DCGAN

Главным преимуществом DCGAN является то, что она способна генерировать высококачественные изображения с высокой разрешающей способностью. Это связано с тем, что DCGAN использует несколько сверточных слоев в генераторе и дискриминаторе, что позволяет модели изучать более сложные иерархические особенности в данных.

Для улучшения качества сгенерированных изображений в DCGAN используются различные методы, такие как:

− Batch normalization - позволяет ускорить процесс обучения и уменьшить вероятность переобучения

− LeakyReLU - функция активации, которая позволяет избежать проблемы «мертвых нейронов»

Таким образом, DCGAN позволяет создавать высококачественные изображения, которые могут быть использованы в различных задачах, таких как распознавание образов, классификация изображений и машинное зрение в целом.

2.7 Активационные функции

В искусственных нейронных сетях функция активации нейрона определяет выходной сигнал, который определяется входным сигналом или набором входных сигналов. Рассмотрим несколько из активационных функций:

1) Сигмоида. Она нелинейна по своей природе, а комбинация таких функций производит тоже нелинейную функцию. Формула (1) является формулой сигмоиды

$$f(x) = \frac{1}{1+e^{-x}} \quad (1)$$

Еще одно достоинство такой функции – она не бинарна, что делает активацию аналоговой, в отличие от ступенчатой функции. Для сигмоиды также характерен гладкий градиент.

В диапазоне значений X от -2 до 2 значения Y меняется очень быстро. Это означает, что любое малое изменение значения X в этой области влечет существенное изменение значения Y.

2) Гиперболический тангенс. Это еще одна часто используемая активационная функция. Формула (2) является формулой гиперболического тангенса

$$f(x) = \tanh(x) = \frac{2}{1+ e^{-2x}} \quad (2)$$

Гиперболический тангенс очень похож на сигмоиду. И действительно, это скорректированная сигмоидная функция.

Поэтому такая функция имеет те же характеристики, что и у сигмоиды, рассмотренной ранее. Её природа нелинейна, она хорошо подходит для

комбинации слоёв, а диапазон значений функции – (-1, 1). Поэтому активационная функция не перегрузится от больших значений.

3) ReLU. Пользуясь формулой 3, становится понятно, что ReLU возвращает значение x, если x положительно, и 0 в противном случае

$$f(x) = \max(0, x) \quad (3)$$

ReLU нелинейна по своей природе, а комбинация ReLU также нелинейна. ReLU менее требовательно к вычислительным ресурсам, чем гиперболический тангенс или сигмоида, так как производит более простые математические операции. Поэтому имеет смысл использовать ReLU при создании глубоких нейронных сетей.

Однако, ReLU имеет некоторые недостатки.

Во-первых, при использовании ReLU, некоторые нейроны могут «умереть», то есть они могут получить отрицательное значение и оставаться неактивными на всем протяжении обучения.

Во-вторых, ReLU несимметрична относительно нуля, поэтому может возникнуть проблема «расслоения», когда нейроны могут выдавать только положительные значения. Для решения этих проблем могут быть использованы другие функции активации, такие как LeakyReLU.

4) LeakyReLU. Активационная функция, которая используется в нейронных сетях в качестве нелинейной функции активации. Она представляет собой улучшенную версию ReLU функции, которая решает проблему «мертвых» нейронов, которые могут возникнуть при использовании ReLU. [14]

LeakyReLU работает так же, как и ReLU, возвращая ноль, если вход меньше нуля, и возвращая сам вход, если он больше или равен нулю. Однако, в отличие от ReLU, LeakyReLU имеет небольшой отрицательный наклон для отрицательных значений входа, что позволяет избежать «мертвых» нейронов. Формула для LeakyReLU выглядит следующим образом:

$$f(x) = \max(\alpha x, x) \quad (4)$$

где α – отрицательный уклон, который является маленьким положительным числом, например, 0,01.

Преимуществом LeakyReLU является устойчивость к "умиранию" нейронов и лучшая сходимость в процессе обучения, что приводит к более быстрому и точному обучению нейронных сетей.

Использование LeakyReLU может помочь улучшить производительность нейронной сети, особенно при обучении глубоких нейронных сетей.

2.8 Batch normalization

Batch normalization (BN) — это метод нормализации входных данных в нейронных сетях путём вычисления среднего и дисперсии для каждого батча обучающей выборки. Он был впервые предложен в 2015 году и быстро стал широко используемым в современных нейронных сетях.[15]

Целью BN является стабилизация распределения активаций между слоями, что упрощает процесс обучения сети и ускоряет его сходимость. В частности, BN препятствует затуханию градиента и позволяет использовать более высокие значения скорости обучения, что ускоряет процесс обучения.

Принцип работы BN заключается в нормализации каждой активации признакового описания внутри батча, путём вычитания из неё среднего значения батча и деления на стандартное отклонение батча. Затем нормализованное значение подвергается преобразованию (с помощью двух параметров масштабирования и смещения), что позволяет сети использовать нелинейные преобразования на нормализованных данных.

BN применяется к выходу сверточных и полносвязных слоёв, обычно после активации, и может быть использован как на этапе обучения, так и на этапе тестирования.

2.9 Алгоритм оптимизации Адам

Адам – это алгоритм оптимизации, который можно использовать вместо классической процедуры стохастического градиентного спуска для итеративного обновления весов сети на основе обучающих данных.

Вместо того чтобы адаптировать скорости обучения параметров на основе среднего первого момента (среднего), Адам использует среднее значение вторых моментов градиентов (нецентрированная дисперсия).

В частности, алгоритм вычисляет экспоненциальную скользящую среднюю градиента и квадрата градиента, а параметры β_1 и β_2 управляют скоростью затухания этих скользящих средних.[16]

Начальное значение скользящих средних и значений β_1 и β_2, близких к 1,0 (рекомендуется), приводит к смещению оценок моментов в сторону нуля. Это смещение преодолевается сначала вычислением смещенных оценок, а затем вычислением оценок с поправкой на смещение.

Адам является популярным алгоритмом в области глубокого обучения, потому что он быстро достигает хороших результатов.

Эмпирические результаты показывают, что Адам хорошо работает на практике и выгодно отличается от других методов стохастической оптимизации.

Используя большие модели и наборы данных, Адам может эффективно решать практические проблемы глубокого обучения.

Таким образом, язык программирования Python удобен для разработки нейронных сетей, так как содержит мощные алгоритмы для нейросетей и удобные библиотеки.

3 Разработка генерирующей нейронной сети

В качестве результата курсовой работы была разработана нейронная сеть «FaceForge», которая формирует собирательный портрет.

Нейронная сеть была создана на основе DCGAN модели для формирования изображений. Для этого мы обучили модель, используя набор данных CelebA, состоящий из фотографий лиц людей.

В обучающей выборке каждое изображение в CelebA содержит множество меток атрибутов, таких как наличие улыбки, наличие бороды, цвет волос. По мере того, как нейронная сеть изучает распределение данных, она также генерирует собирательные образы. На рисунке 7 представлены изображения размером 32*32 пикселя, на которых обучалась нейронная сеть «FaceForge»

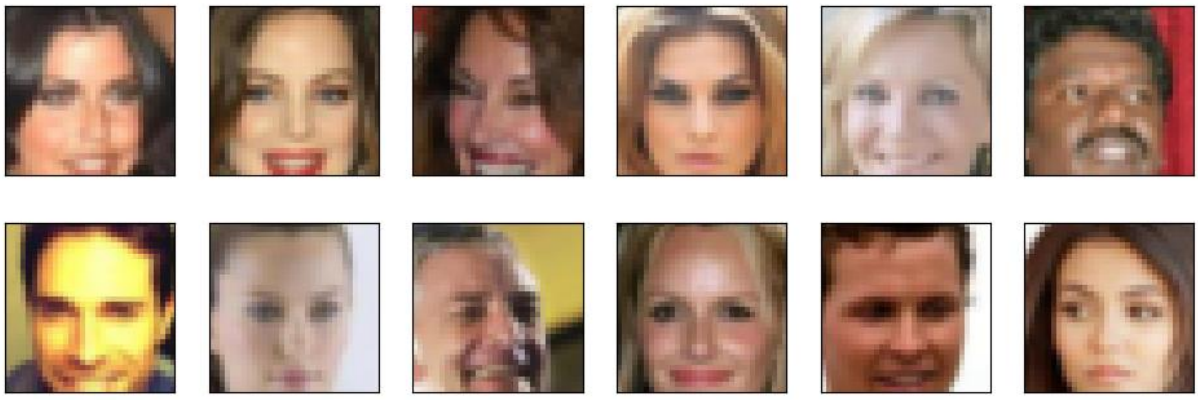


Рисунок 7 – Примеры случайных изображений, на которых обучалась нейронная сеть

3.1 Архитектура генератора

Архитектура генератора состоит из одного полносвязного слоя и трех транспонированных сверточных слоев. На вход генератора поступает случайный вектор шумов, состоящий из 100 элементов, по которому генерируется изображение.

Первый сверточный слой имеет входную размерность 32*4, глубину 32*2 и размер ядра 4.

Следующий сверточный слой имеет входную размерность 32*2, глубину 32 и размер ядра 4.

Третий сверточный слой имеет входную размерность 32, глубину 3 (RGB каналы изображения) и размер ядра 4.

Входной слой генератора – полносвязный слой с размером входных данных 100, который генерирует вектор шума.

Выходной слой генератора – транспонированный сверточный слой с функцией активации tanh.

Функция активации для скрытых слоев генератора – ReLU.Выходными данными генератора является изображение размером 32x32 пикселя. На рисунке 8 изображена архитектура генератора нейронной сети «FaceForge»

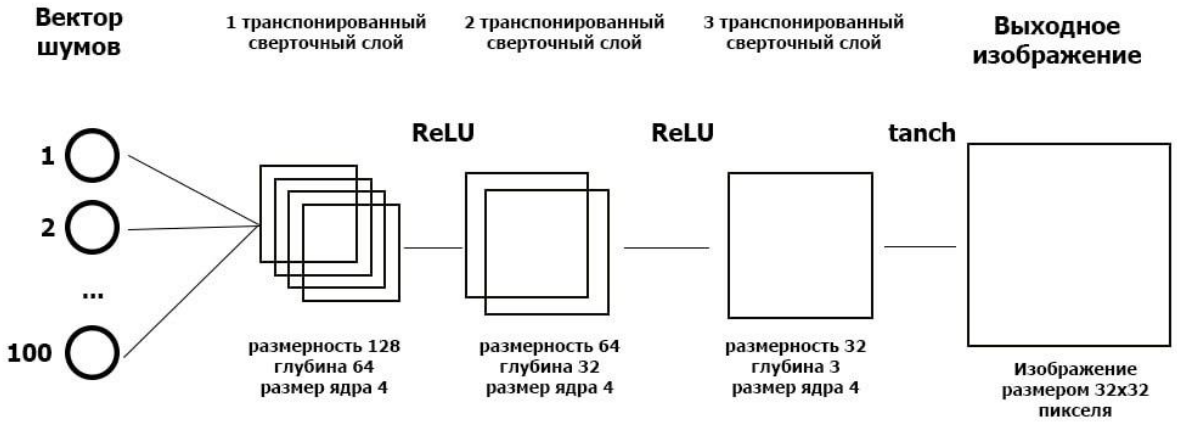


Рисунок 8 – Архитектура генератора

3.2 Архитектура дискриминатора

Архитектура дискриминатора состоит из трех сверточных слоев с последующим полносвязным слоем. На вход дискриминатора поступает изображение размером 32x32 пикселя.

Первый сверточный слой имеет входную размерность 3 (RGB каналы изображения), глубину 32 и размер ядра 4.

Следующий сверточный слой имеет входную размерность 32, глубину 32*2 и размер ядра 4.

Третий сверточный слой имеет входную размерность 32*2, глубину 32*4 и размер ядра 4.

Затем идет выравнивание полученных карт признаков и подача их на полносвязный слой, который имеет один выходной нейрон для бинарной классификации (0 - подлинное изображение, 1 - сгенерированное).

Функция активации для сверточных слоев – LeakyReLU с наклоном 0.2, а для полносвязного слоя не используется функция активации. На рисунке 9 изображена архитектура дискриминатора

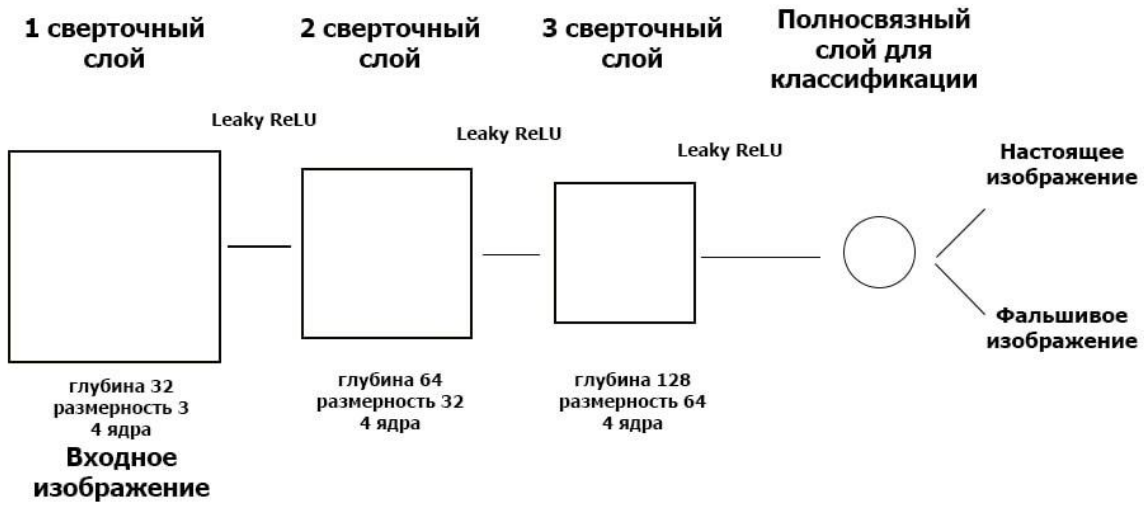


Рисунок 9 – Архитектура дискриминатора

3.3 Функции потерь

Функция потерь используется для расчета ошибки между реальными и полученными ответами. Наша глобальная цель – минимизировать эту ошибку. Для обучения GAN используется две функции потерь: функция потерь генератора (generator loss) и функция потерь дискриминатора (discriminator

loss). В математическом плане процесс обучения GAN заключается в минимаксной игре генератора и дискриминатора, в которой дискриминатор адаптирован для минимизации ошибки различия реального и сгенерированного образца, а генератор адаптирован на максимизацию вероятности того, что дискриминатор допустит ошибку. Таким образом, функция потерь эффективно приближает обучение нейронной сети к этой цели.

Функция потерь генератора – это среднеквадратичное отклонение между выходом генератора и единичным тензором (эталонной картинкой).

Функция потерь дискриминатора – это сумма двух слагаемых: среднеквадратичного отклонения между выходом дискриминатора на настоящих исходных данных и единицами, а также среднеквадратичного отклонения между выходом дискриминатора на сгенерированных данных и нулями. Это задача классификации, где единицы соответствуют реальным изображениям, а нули – сгенерированным.

3.4 Обучение

Теперь, когда мы определили модели для дискриминатора и генератора, мы готовы начать обучение. Нейронная сеть «FaceForge» обучалась на протяжении 50 эпох.

На рисунке 10 указаны результаты обучения дискриминатора и генератора нейронной сети, где на оси x указано количество эпох обучения, а на оси y указано значение функции потерь.

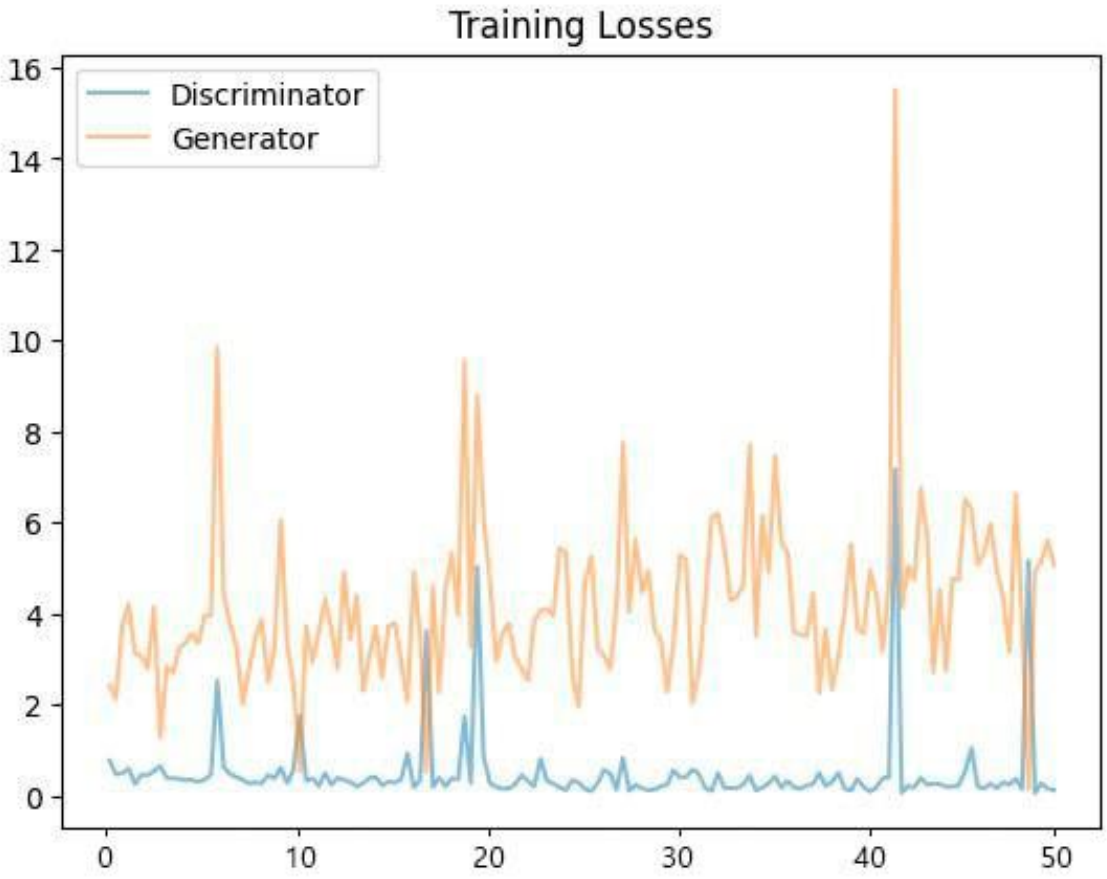


Рисунок 10 – Результаты обучения генератора и дискриминатора

Как мы можем видеть, результаты обучения еще можно улучшить, увеличив время обучения.

Выведем несколько собирательных образа, сгенерированных в результате работы нейронной сети по случайному шуму. На рисунке 11 представлены выходные данные работы программы.

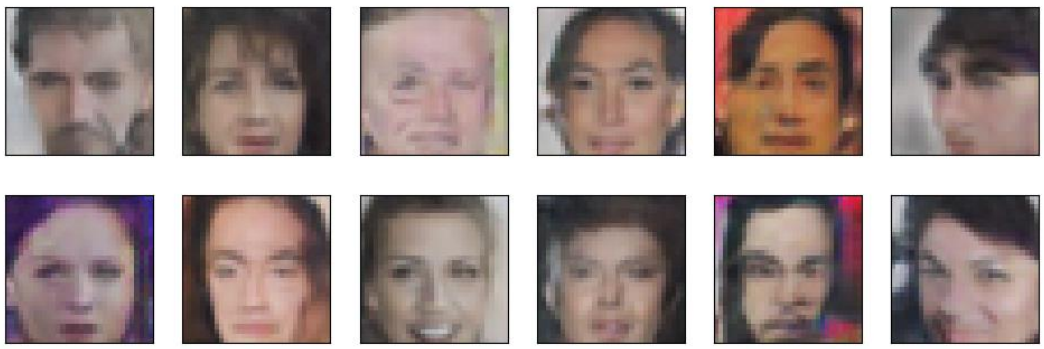


Рисунок 11 – Собирательные образы, сгенерированные через 50 эпох

В начале процесса обучения сгенерированные изображения абсолютно случайны. По мере обучения генератор изучает распределение реальных данных, и примерно через двадцать эпох некоторые сгенерированные изображения лиц уже напоминают реальных людей. На рисунке 12 показаны собирательные образы, сгенерированные через 20 эпох обучения нейронной сети.

Рисунок 12 – Собирательные образы сгенерированные через 20 эпох обучения

Результаты после 50 эпох обучения лучше, чем после 20 эпох, так же после пятидесяти эпох обучения есть несколько портретов, похожих на лица реальных людей. Результаты можно улучшить, проводя более длительное обучение, однако это займет большее количество времени, и требует больших вычислительных мощностей. Полный код программы представлен в приложении А.

Таким образом, нами была разработана нейронная сеть, которая формирует собирательный портрет с использованием Python и PyTorch.

ЗАКЛЮЧЕНИЕ

В заключение подведем основной итог курсовой работы. За время выполнения работы цель была достигнута, а именно: мы разработали систему искусственного интеллекта для формирования собирательного портрета. Также, были выполнены все поставленные задачи:

– Подготовка и обработка данных для обучения модели.

– Исследование различных архитектур GAN модели нейронных сетей.

– Разработка архитектуры GAN модели, которая будет использоваться для генерации собирательного портрета.

– Обучение GAN модели на обучающей выборке.

Для разработки программного продукта были использованы такие средства разработки нейронной сети как Python, PyTorch. Были рассмотрены различные архитектуры GAN модели нейронных сетей, алгоритмы и структуры, используемые при работе с изображениями.

В результате работы была разработана нейронная сеть, которая формировала собирательный портрет человека.

В будущем планируется улучшить нейронную сеть, для повышения качества изображения, точности и увеличения скорости обработки данных, а также добавить возможность генерировать собирательный портрет с указанием используемых атрибутов.

Конечно, на сегодняшний день нейронные сети способны не только различать и генерировать портреты людей, но и другие изображения в более высоком качестве, причем такие сети используют подобный алгоритм.

Данный алгоритм может быть также усовершенствован и использоваться даже для разработки различного рода программного обеспечения для персональных компьютеров.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Нейронные сети. URL: https://sbercloud.ru/ru/services/neural-networks (дата обращения 14.05.2023)

2. Нейронные сети, перцептрон – викиконспекты. URL: https://neerc.ifmo.ru/wiki/index.php?title=Нейронные_сети,_перцептрон (дата обращения 14.05.2023)

3. Что такое нейронные сети и как они работают. URL: https://sky.pro/media/neyronnye-seti/ (дата обращения 14.05.2023)

4. Эндрю Тарсак, Грокаем глубокое обучение / Эндрю Тарсак. СПБ.: Питер, 2021 – 352 с. – ISBN 978-5-4461-1334-7

5. Типы нейронных сетей. Принцип их работы и сфера применения. URL: https://otus.ru/nest/post/1263/ (дата обращения 14.05.2023)

6. Виды и классификация нейронных сетей. URL: https://aisimple.ru/12-klassifikacija-nejronnyh-setej.html/ (дата обращения 14.05.2023)

7. Generative Adversarial Nets (GAN) URL: https://neerc.ifmo.ru/wiki/index.php?title=Generative_Adversarial_Nets_(GAN) (дата обращения 14.05.2023)

8. Пишем нейросеть на Python с нуля. URL: https://proglib.io/p/pishem-neyroset-na-python-s-nulya-2020-10-07 (дата обращения 14.05.2023)

9. Что такое PyTorch и как он работает. URL: https://www.kverner.ru/chto-takoe-pytorch-i-kak-on-rabotaet/ (дата обращения 14.05.2023)

10. NumPy. URL: https://blog.skillfactory.ru/glossary/numpy/ (дата обращения 14.05.2023)

11. Что такое Google Colab и кому он нуженURL: https://blog.skillfactory.ru/chto-takoe-google-colaboratory-i-komu-on-nuzhen/ (дата обращения 14.05.2023)

12. Сверточные нейронные сети. URL: https://neerc.ifmo.ru/wiki/index.php?title=Сверточные_нейронные_сети (дата обращения 14.05.2023)

13. DCGAN (гулбоко сверточные генеративные состязательные сети) URL: https://machinelearningmastery.ru/dcgans-deep-convolutional-generative-adversarial-networks-c7f392c2c8f8/ (дата обращения 14.05.2023)

14. Выбор слоя активации в нейронных сетях.. URL: https://habr.com/ru/articles/727506/ (дата обращения 14.05.2023)

15. Batch normalization. URL: https://neerc.ifmo.ru/wiki/index.php?title=Batchnormalization#:~:text=Пакетная%20нормализация%20(англ.,стабилизировать%20работу%20искусственных%20нейронных%20сетей.сети (дата обращения 14.05.2023)

16. Реализуем и сравниваем оптимизаторы моделей в глубоком обучении. URL:https://habr.com/ru/companies/skillfactory/articles/525214/#:~:text=Adam%20%20один%20из%20самых%20эффективных,идеи%20RMSProp%20и%20оптимизатора%20импульса. (дата обращения 14.05.2023)

ПРИЛОЖЕНИЕ А

Код нейросети «FaceForge»

```
from IPython.display import clear_output

!wget -cq -O problem_unittests.py
https://raw.githubusercontent.com/udacity/deep-learning-v2-
pytorch/master/project-face-generation/problem_unittests.py
!wget -cq -O processed_celeba_small.zip
https://s3.amazonaws.com/video.udacity-
data.com/topher/2018/November/5be7eb6f_processed-celeba-
small/processed-celeba-small.zip
clear_output()
print("Данные загружены")


!unzip -n processed_celeba_small.zip
clear_output()
print("Данные извлечены")

data_dir = 'processed_celeba_small/'

import pickle as pkl
import matplotlib.pyplot as plt
import numpy as np
import problem_unittests as tests

import torch
from torchvision import datasets
from torchvision import transforms
from torch.utils.data import DataLoader

def get_dataloader(batch_size, image_size,
data_dir='processed_celeba_small/'):


    transform = transforms.Compose([transforms.Resize(image_size),
                                    transforms.ToTensor()])


    dataset = datasets.ImageFolder(data_dir, transform)


    data_loader = DataLoader(dataset, batch_size=batch_size,
shuffle=True, num_workers = 4)

    return data_loader

batch_size = 128
```

```
img_size = 32

celeba_train_loader = get_dataloader(batch_size, img_size)



def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

dataiter = iter(celeba_train_loader)
images, _ = next(iter(celeba_train_loader))
fig = plt.figure(figsize=(20, 4))
plot_size=20
for idx in np.arange(plot_size):
    ax = fig.add_subplot(2, int(plot_size/2), idx+1, xticks=[],
yticks=[])
    imshow(images[idx])

def scale(x, feature_range=(-1, 1)):
    min, max = feature_range
    x = x * (max - min) + min
    return x

img = images[0]
scaled_img = scale(img)

print('Min: ', scaled_img.min())
print('Max: ', scaled_img.max())

import torch.nn as nn
import torch.nn.functional as F

def conv(in_channels, out_channels, kernel_size, stride=2, padding=1,
batch_norm=True):
    layers = []
    conv_layer = nn.Conv2d(in_channels, out_channels,
                           kernel_size, stride, padding, bias=False)

    layers.append(conv_layer)

    if batch_norm:
        # добавляем слой batch normalization
        layers.append(nn.BatchNorm2d(out_channels))

    return nn.Sequential(*layers)

# создание дискриминатора
# conv_dim это глубина первого сверточного слоя
```

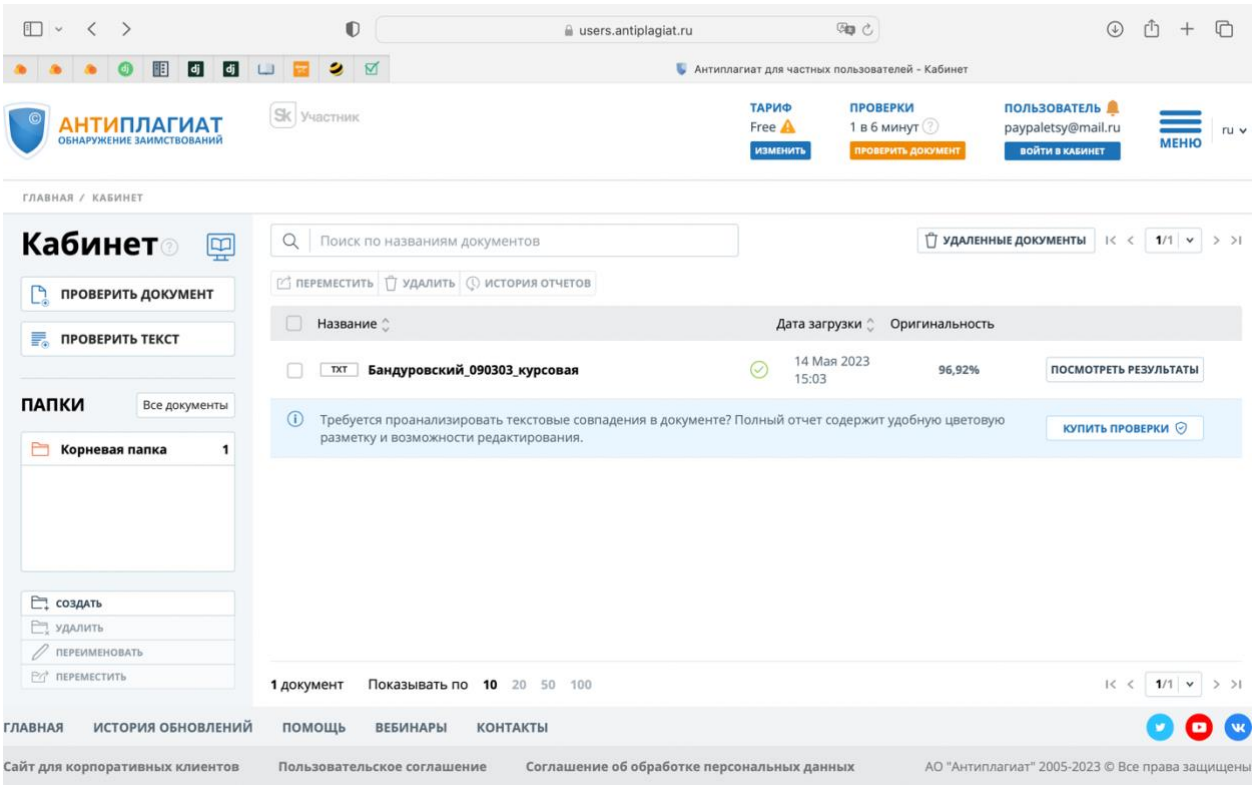
```
class Discriminator(nn.Module):

    def __init__(self, conv_dim):
        super(Discriminator, self).__init__()

        self.conv_dim = conv_dim

        # 32x32 вход
        self.conv1 = conv(3, conv_dim, 4, batch_norm=False)
        # 16x16 выход
        self.conv2 = conv(conv_dim, conv_dim*2, 4)
        # 8x8 выход
        self.conv3 = conv(conv_dim*2, conv_dim*4, 4)
        # 4x4 выход

        # полносзвязный
        self.fc = nn.Linear(conv_dim*4*4*4, 1)



        # прямое распространение нейронной сети
        # x входные данные в нейронную сеть
    def forward(self, x):

        out = F.leaky_relu(self.conv1(x), 0.2)
        out = F.leaky_relu(self.conv2(out), 0.2)
        out = F.leaky_relu(self.conv3(out), 0.2)

        out = out.view(-1, self.conv_dim*4*4*4)

        out = self.fc(out)
        return out

tests.test_discriminator(Discriminator)

# создаем транспонированный сверточный слой с необязательной пакетной
нормализацией
def deconv(in_channels, out_channels, kernel_size, stride=2,
padding=1, batch_norm=True):
    layers = []
    transpose_conv_layer = nn.ConvTranspose2d(in_channels,
out_channels,
                                              kernel_size, stride,
padding, bias=False)
    layers.append(transpose_conv_layer)

    if batch_norm:
        layers.append(nn.BatchNorm2d(out_channels))

    return nn.Sequential(*layers)
```

```
# создаем генератор
# z_size длина входного скрытого вектора
# conv_dim глубина входных данных для последнего транспортируемого
сверточного слоя
class Generator(nn.Module):

    def __init__(self, z_size, conv_dim):

        super(Generator, self).__init__()

        self.conv_dim = conv_dim

        # первый полносвязный слой
        self.fc = nn.Linear(z_size, conv_dim*4*4*4)

        # сверточные слои
        self.t_conv1 = deconv(conv_dim*4, conv_dim*2, 4)
        self.t_conv2 = deconv(conv_dim*2, conv_dim, 4)
        self.t_conv3 = deconv(conv_dim, 3, 4, batch_norm=False)


        # прямое распространение
    def forward(self, x):
        out = self.fc(x)
        out = out.view(-1, self.conv_dim*4, 4, 4) # (batch_size,
depth, 4, 4)

        # скрытые слои, активация - relu
        out = F.relu(self.t_conv1(out))
        out = F.relu(self.t_conv2(out))

        # последний слой активация - tanh
        out = self.t_conv3(out)
        out = torch.tanh(out)

        return out

tests.test_generator(Generator)

def weights_init_normal(m):

    classname = m.__class__.__name__

    if classname.find('Linear') != -1 or classname.find('Convo2d') !=
-1:
        # получение количества входных данных
        n = m.in_features
        y = (1.0/np.sqrt(n))
        m.weight.data.normal_(0, y)
        m.bias.data.fill_(0)
```

```
def build_network(d_conv_dim, g_conv_dim, z_size):
    D = Discriminator(d_conv_dim)
    G = Generator(z_size=z_size, conv_dim=g_conv_dim)
    D.apply(weights_init_normal)
    G.apply(weights_init_normal)

    print(D)
    print()
    print(G)

    return D, G

d_conv_dim = 32
g_conv_dim = 32
z_size = 100

D, G = build_network(d_conv_dim, g_conv_dim, z_size)

import torch

train_on_gpu = torch.cuda.is_available()
if not train_on_gpu:
    print('No GPU found. Please use a GPU to train your neural
network.')
else:
    print('Training on GPU!')

def real_loss(D_out, smooth=False):
    batch_size = D_out.size(0)

    if smooth:

        labels = torch.ones(batch_size)*0.9
    else:
        labels = torch.ones(batch_size) # real labels = 1
    if train_on_gpu:
        labels = labels.cuda()
    criterion = nn.BCEWithLogitsLoss()
    loss = criterion(D_out.squeeze(), labels)
    return loss

def fake_loss(D_out):
    batch_size = D_out.size(0)
    labels = torch.zeros(batch_size) # fake labels = 0

    if train_on_gpu:
        labels = labels.cuda()

    criterion = nn.BCEWithLogitsLoss()
    loss = criterion(D_out.squeeze(), labels)
    return loss
```

```
import torch.optim as optim

lr = 0.002
beta1=0.4
beta2=0.999

d_optimizer = optim.Adam(D.parameters(), lr, [beta1, beta2])
g_optimizer = optim.Adam(G.parameters(), lr, [beta1, beta2])

def train(D, G, n_epochs, print_every=300):

    if train_on_gpu:
        D.cuda()
        G.cuda()

    samples = []
    losses = []
    sample_size=16
    fixed_z = np.random.uniform(-1, 1, size=(sample_size, z_size))
    fixed_z = torch.from_numpy(fixed_z).float()
    if train_on_gpu:
        fixed_z = fixed_z.cuda()

    for epoch in range(n_epochs):
        for batch_i, (real_images, _) in
enumerate(celeba_train_loader):

            batch_size = real_images.size(0)
            real_images = scale(real_images)

            d_optimizer.zero_grad()
            if train_on_gpu:
                real_images = real_images.cuda()

            D_real = D(real_images)
            d_real_loss = real_loss(D_real)
            z = np.random.uniform(-1, 1, size=(batch_size, z_size))
            z = torch.from_numpy(z).float()
            if train_on_gpu:
                z = z.cuda()
            fake_images = G(z)

            D_fake = D(fake_images)
            d_fake_loss = fake_loss(D_fake)
            d_loss = d_real_loss + d_fake_loss
            d_loss.backward()
            d_optimizer.step()
            g_optimizer.zero_grad()
            z = np.random.uniform(-1, 1, size=(batch_size, z_size))
            z = torch.from_numpy(z).float()
```

```
            if train_on_gpu:
                z = z.cuda()
            fake_images = G(z)
            D_fake = D(fake_images)
            g_loss = real_loss(D_fake)


            g_loss.backward()
            g_optimizer.step()

            if batch_i % print_every == 0:
                losses.append((d_loss.item(), g_loss.item()))
                print('Epoch [{:5d}/{:5d}] | d_loss: {:6.4f} | g_loss:
{:6.4f}'.format(
                        epoch+1, n_epochs, d_loss.item(),
g_loss.item()))

        G.eval()
        samples_z = G(fixed_z)
        samples.append(samples_z)
        G.train()

    with open('train_samples.pkl', 'wb') as f:
        pkl.dump(samples, f)

    return losses

n_epochs = 50

losses = train(D, G, n_epochs=n_epochs)

fig, ax = plt.subplots()
losses = np.array(losses)
plt.plot(losses.T[0], label='Discriminator', alpha=0.5)
plt.plot(losses.T[1], label='Generator', alpha=0.5)
plt.title("Training Losses")
plt.legend()
def view_samples(epoch, samples):
    fig, axes = plt.subplots(figsize=(16,4), nrows=2, ncols=8,
sharey=True, sharex=True)
    for ax, img in zip(axes.flatten(), samples[epoch]):
        img = img.detach().cpu().numpy()
        img = np.transpose(img, (1, 2, 0))
        img = ((img + 1)*255 / (2)).astype(np.uint8)
        ax.xaxis.set_visible(False)
        ax.yaxis.set_visible(False)
        im = ax.imshow(img.reshape((32,32,3)))
with open('train_samples.pkl', 'rb') as f:
    samples = pkl.load(f)

_ = view_samples(-1, samples)
```


users.antiplagiat.ru
Антиплагиат для частных пользователей - Кабинет
АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ
Участник
ТАРИФ
Free
ИЗМЕНИТЬ
ПРОВЕРКИ
1 в 6 минут
ПРОВЕРИТЬ ДОКУМЕНТ
ПОЛЬЗОВАТЕЛЬ
paypaletsy@mail.ru
ВОЙТИ В КАБИНЕТ
МЕНЮ
ru
ГЛАВНАЯ / КАБИНЕТ
Кабинет
ПРОВЕРИТЬ ДОКУМЕНТ
ПРОВЕРИТЬ ТЕКСТ
ПАПКИ
Все документы
Корневая папка 1
СОЗДАТЬ
УДАЛИТЬ
ПЕРЕИМЕНОВАТЬ
ПЕРЕМЕСТИТЬ
Поиск по названиям документов
УДАЛЕННЫЕ ДОКУМЕНТЫ
1/1
ПЕРЕМЕСТИТЬ
УДАЛИТЬ
ИСТОРИЯ ОТЧЕТОВ
Название
Дата загрузки
Оригинальность
TXT
Бандуровский_090303_курсовая
14 Мая 2023
15:03
96,92%
ПОСМОТРЕТЬ РЕЗУЛЬТАТЫ
Требуется проанализировать текстовые совпадения в документе? Полный отчет содержит удобную цветовую разметку и возможности редактирования.
КУПИТЬ ПРОВЕРКИ
1 документ
Показывать по 10 20 50 100
1/1
ГЛАВНАЯ
ИСТОРИЯ ОБНОВЛЕНИЙ
ПОМОЩЬ
ВЕБИНАРЫ
КОНТАКТЫ
Сайт для корпоративных клиентов
Пользовательское соглашение
Соглашение об обработке персональных данных
АО "Антиплагиат" 2005-2023 © Все права защищены