МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ ИМЕНИ ПАТРИСА ЛУМУМБЫ»
Факультет физико-математических и естественных наук
Математический институт им. Никольского

«Допустить к защите»

Директор Математического
института им. С.М. Никольского
Муравник А.Б.
« » 2024 г.

Выпускная квалификационная работа бакалавра

Направление/Специальность 01.03.02 «Прикладная математика и информатика»

ТЕМА Создание алгоритма для создания музыкального (аудио) сопровождения к видеоклипам.

Выполнил студент Гамаюнов Никита Ефимович

Группа НПМбд-01-20
Студ. билет №1032201719

Руководитель выпускной
квалификационной работы
Карандашев Я.М. доцент, к.ф.-м.н.

Автор_____

г. Москва
2024г.

Содержание

Введение

Генеративные нейросетевые алгоритмы сегодня стремительно развиваются, привлекают всё больше внимания и меняют наше отношение к информации. Результаты нейронной генерации значительно улучшились за последнее десятилетие: в 2014 году была предложена архитектура GAN [1] для создания изображений, в 2017 – трансформерная архитектура [2], сделавшая прорыв в задачах генерации «последовательность-последовательность» (Seq2Seq), таких, как машинный перевод и обусловленная генерация текста. Модификации обеих идей всё ещё используются в современных подходах: например, DALLE [3], который создаёт изображения на основе входного текста, использует обе: его можно условно разбить на две части: первая – трансформер, который генерирует из текста токены для второй части – VQ-VAE, в последствии часто заменяемую на VQ-GAN [4], которая на основе них создаёт изображения.

Прорывы в решении генеративных задач стали возможны благодаря двум аспектам: развитию аппаратного обеспечения для выполнения сложных вычислений и доступности тренировочных данных. После появления графических ускорителей типа GPU и такого программно-аппаратного обеспечения, как CUDA [5], обучение нейросетевых моделей стало быстрее и дешевле, и доступ к нейросетевой генерации получило гораздо больше исследователей, чем раньше. Что же касается данных, для тренировки большинства генеративных моделей используются пары «результат генерации – условие» (например, это могут быть изображения и их описания), которые могут быть автоматически загружены из Интернета [6].

Большинство алгоритмов для генерации фокусируется на создании текста или изображений – во многом из-за большего ажиотажа и возможности монетизировать готовый продукт, но одна из немаловажных причин для этого – простота получения большого количества пар «изображение – описание» и

«текстовый запрос – текстовый ответ». Для обучения нейросети DALLE было использовано 650 млн. таких пар. Получить похожее количество пар чистых данных вида «звук – описание» гораздо сложнее, например, размер набора данных (датасета) для тренировки «звукового аналога» DALLE AudioGen равен примерно 3 млн. пар. [7, с.1]

Однако с исследовательской точки зрения задача генерации звука является не менее интересной, чем аналоги для изображений и текста. Практически каждый год появляются новые статьи, предлагающие уникальные алгоритмы для генерации и решающие проблемы, связанные с особенностями звуковых данных. Такие работы привлекают меньшее внимание общественности в силу того, что большим спросом пользуются диалоговые чат-модели или генераторы визуальных данных, которые могут выступать в роли личных ассистентов или дизайнеров, заменять системы поиска в Интернете или платные фотобанки. Но возможность быстро получать сгенерированные аудиосэмплы определённо может пригодиться в разных сферах – например, в независимом кинематографе, где бюджеты для съёмок, как правило, невелики при том, что необходимость качественной озвучки действий на экране огромна.

Большинство работ в области генерации аудио выбирают входным условием генерации текст – с точки зрения контроля над результатом этот вариант очень полезен, ведь любые детали, которые должны быть включены в выходные данные алгоритма, можно описать словами. Кроме того, один из главных результатов, полученный авторами модели DALLE-3 – улучшение качества генерации при тренировке на более подробных описаниях [8, с.8]. Несмотря на то, что эта модель используется для создания изображений, принцип её работы схож с принципом работы её аналогов для звука, что позволяет говорить о применимости полученного результата.

Один из алгоритмов, решающих задачу генерации звука по входному тексту, – AudioGen (2023) [9]. Подавляющая часть современных генеративных

моделей не являются монолитными, а состоит из нескольких нейронных сетей, каждая из которых выполняет отдельные функции. AudioGen – не исключение, его можно представить в виде трёх основных компонентов. Первый из них – автокодировщик, который способен кодировать данные в меньшее представление, содержащее основную информацию о них, и декодировать в изначальный вид с наименьшими потерями. Второй компонент алгоритма – звуковая языковая модель [10], обученная для генерации кодов автокодировщика. Третья часть AudioGen – предобученный кодировщик текста, который моделирует наиболее удобное семантическое представление входной информации для языковой модели.

Разделение задач внутри алгоритма позволяет авторам заниматься генерацией меньшего представления (часто называемого латентным), которое можно декодировать с помощью первой модели, вместо большого массива данных в оригинальном виде. Все решения, обсуждаемые в этом блоке, используют похожие методы кодирования и декодирования звуковой информации для облегчения вычислительных мощностей, поскольку современные подходы к нейросетевой компрессии [11] позволяют делать это без заметных потерь в качестве.

Похожий подход используется для построения AudioLDM 2 [12] – ещё одного алгоритма, который специализируется на генерации, обусловленной текстом. Авторы этой работы решают, вообще говоря, обобщённую задачу: предлагают использовать специальное представление звука, называемое «язык аудио» (Language of audio, LOA), получаемое с помощью точной настройки модели GPT-2 [13]. На основании языка аудио, как и на основании кодов текста в AudioGen, создаются аудиофайлы. Главное отличие здесь в том, что для генерации используется диффузионная модель [14][15]. Кроме того, особенность LOA – унификация звуковой информации. Другие генеративные модели для работы с аудио обычно фокусируются на одной из трёх задач – генерации речи, музыки или «звуковых эффектов» – любых других звуков.

Очевидно, эти задачи имеют не очень много сходств. Например, музыка будет достаточно сильно отличаться от любых других звуков как семантически, так и по смыслу. Но AudioLDM 2 удаётся показывать достойные результаты для каждого типа звука – благодаря универсальности информации, содержащейся в LOA.

В статье также рассматривается идея о том, что в LOA можно перевести не только текст, но и информацию «любой модальности», используя разные кодировщики [12, с.5]. Приводятся примеры с генерацией, обусловленной текстом, аудио и фото, но обучение проводилось с использованием пар «текст-аудио», и результаты с вычислением актуальных метрик машинного обучения даны только для работы с текстом. Исходя из этого, AudioLDM 2 можно считать моделью, разработанной для работы с текстовыми условиями.

Названные выше модели являются авторегрессивными. Это означает, что при генерации последовательности каждая новая её часть моделируется на основе частей, сгенерированных ранее (или запросов с условиями, если генерация только началась) [16]. Существует подход, отказывающийся от идеи авторегрессии – MAGNet [17]. Главная его идея в том, что во время генерации части выходной последовательности проходят через стадию переоценки отдельной моделью, и в случае, если не признаются подходящими – скрываются маской, чтобы вместо них были сгенерированы новые. В остальном же устройство сильно напоминает уже рассмотренные выше – входная информация переводится в латентное представление, и на основе него с помощью трансформерной модели создаются коды, которые затем декодируются в выходные данные.

Однако не все модели для создания аудио обучаются исключительно на текстовых условиях. Так, для модели Im2Wav [7] было предложено интересное решение проблемы со сложностью добычи данных: использование видео как входного условия для генерации. Видеофайлы по умолчанию содержат в себе согласованные пары вида «последовательность кадров –

звук», и Интернет содержит достаточное количество видео для составления датасетов любых размеров: только на YouTube находится более 14,5 млрд файлов [18].

Данная модель использует схожую с остальными архитектуру, главное отличие здесь в том, что в качестве декодировщика используется VQ-VAE 2 [19], принимающий на вход два набора кодов, каждый из которых генерируется с помощью отдельной трансформерной модели. Видео, используемые для обучения, представляются в виде последовательности кадров, каждый из которых кодируется независимо от других, затем все полученные коды усредняются и подаются на вход трансформеров в качестве условий.

И трансформеры, и диффузионные модели, используемые в описанных статьях для генерации, требуют больших вычислительных мощностей для обучения и использования. Не во всех работах приводятся точные данные о конфигурации оборудования, но такая информация есть для AudioGen и Music LDM.

Авторы первого алгоритма обучили две модели. Базовая (285 млн. параметров) обучалась на 64 графических ускорителях A100 в течении приблизительно 5 дней. Обучение большой (1 млрд. параметров) заняло примерно неделю на 128 ускорителях A100. Число шагов обучения одинаковое – 200 тысяч [9, с.6].

В статье об AudioLDM 2 не приводится время, ушедшее на обучение, но упоминается аппаратная конфигурация – 8 ускорителей A100 [12, с.7].

Учитывая большой объём компьютерного оборудования, использовавшийся в описанных выше работах, построение алгоритмов схожего масштаба в рамках данной выпускной работы не представляется возможным. Однако построение алгоритма, оперирующего на оборудовании меньшей мощности, всё ещё является доступной опцией.

Я бы хотел сфокусироваться на генерации звуковых эффектов, обусловленной входным видео или изображением. Причина этому – небольшое количество исследований в данном направлении и потенциал в создании качественного аудио сопровождения к видеороликам с помощью генеративных алгоритмов.

Итак, задача данной выпускной работы – исследовать возможности разработки алгоритма для нейросетевой генерации аудио, обусловленной входным изображением или последовательностью изображений (видео) с использованием ограниченных вычислительных ресурсов, и создать подобный алгоритм, адаптировав его архитектуру для работы на небольших мощностях.

Глава 1. Методы

1.1 Нейронная сеть

Контекст данной выпускной работы подразумевает изучение и использование широкого спектра различных нейросетевых моделей. Однако исследования будут направлены не на разбор базовых понятий и принципов работы с ними, но на работу с достаточно продвинутыми и сложными вариантами сетей, требующими длительного изучения. В силу этого, в этом разделе будет дана обобщённая информация о нейронных сетях, которая потребуется в дальнейшей работе, не сфокусированная на детальном объяснении устройства каждой структуры и алгоритма.

Нейронная сеть – одна из самых известных моделей машинного обучения, имитирующая работу человеческого мозга. Современный вид простейшей нейросети, называемой полносвязной (fully-connected, FC) сетью или многослойным перцептроном (multi-layer perceptron, MLP) был предложен Филипом Розенблаттом в 1962 в книге «Принципы нейродинамики перцептроны и теория механизмов мозга» [20].

Простейший элемент нейросети – нейрон – имеет вход, выход и функцию активации внутри. Поступающая на вход информация – вектор из рациональных (в силу особенностей работы компьютера) чисел, умножается скалярно на вектор весов той же размерности, что и входной, затем к ней прибавляется число, называемое сдвигом (bias). Выход нейрона – результат умножения, к которому применена функция активации (обычно это нелинейная дифференцируемая функция) [21, с.23].

В полносвязной сети нейроны объединены в слои. Каждый нейрон в слое связан со всеми нейронами предыдущего слоя. Последний, выходной слой, обычно имеет специальный вид – например, в случае классификации данных на десять классов он будет состоять из десяти нейронов. Порядковый номер нейрона с максимальным выходным значением может считаться меткой класса, определённого сетью.

Формально, выход нейрона y представим в виде

$$y_i = f\left(\left(w^i, x\right) + b_i\right), \quad (1)$$

где i – номер слоя, x – входящая информация, w^i – вектор весов, b_i – сдвиг, f – функция активации. Векторы x и w имеют одинаковую размерность.

Выход слоя можно представить в виде

$$y = f(Wx + b), \quad (2)$$

где x – входящая информация, W – матрица весов (j-я строка матрицы – вектор весов для нейрона с номером j), b – вектор сдвига, f – функция активации.

Многослойный перцептрон может быть представлен в виде линейной комбинации слоёв. Если при этом выбрать линейную функцию активации внутри нейронов, каждый слой можно будет считать линейной функцией, зависящей от входной информации, весов и сдвигов нейронов. Тогда вся сеть, будучи линейной комбинацией линейных функций, будет являться линейной функцией, то есть, её можно будет заменить всего одним слоем. Таким образом, чтобы каждый слой внутри нейронной сети имел значение, функция активации должна быть нелинейной. Кроме того, дальше будет показано, что она должна быть дифференцируемой (или хотя бы кусочно-дифференцируемой) для вычисления функции ошибки при обучении.

Множество весов и сдвигов для всех нейронов в сети, называется множеством параметров модели, и часто обозначается как $\Theta = \{W_1, \dots W_n, b_1, \dots b_n\}$, где W_i – матрица весов, b_i – вектор значений сдвига для слоя i.

Тогда многослойный перцептрон может быть представлен в виде функции

$$N(x, \Theta): X \to Y, \quad (3)$$

где X и Y – множества всех входных и выходных данных соответственно.

Нейронная сеть, состоящая из более, чем одного слоя, называется глубокой. Раздел машинного обучения, работающий с такими сетями, получил

название «глубокое обучение». На сегодняшний день разработано множество архитектур нейронных сетей для выполнения различных задач. Кроме того, некоторые слои в нейросетях также отличаются от классического набора нейронов, предложенного Ф. Розенблаттом.

Изменяя параметры Θ, можно влиять на результат работы сети. Подбор оптимальных параметров называется обучением модели. Для обучения необходим инструмент вычисления качества работы сети, и таким инструментом является функция ошибки (loss function, лосс).

Пример простой функции ошибки – среднеквадратичная (mean squared error, MSE):

$$MSE(x, y, \Theta) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - N(x_i, \Theta)\right)^2, \quad (4)$$

где $x_i \in X$ и $y_i \in Y$ – данные из обучающего набора данных (датасета). Обычно датасет содержит пары данных вида «входные данные – ожидаемые выходные данные». Здесь x_i – элемент, подающийся на вход, y_i – ожидаемый выход, N – размер датасета.

Поскольку ранее было упомянуто требование о дифференцируемости функции ошибки в нейронной сети, вся модель N является комбинацией дифференцируемых функций. Значит, производные функции ошибки по параметрам Θ могут быть вычислены по правилу дифференцирования сложной функции – а зная производные лосс-функции, мы можем изменить параметры так, чтобы уменьшить ошибку. Формально современный метод обучения нейронных сетей называется стохастическим градиентным спуском [22]. Он отличается от обычного градиентного спуска тем, что градиенты функции ошибки вычисляется не для всех данных в датасете, а для небольших случайных выборках из него, называемых батчами. Это помогает существенно сократить использование вычислительных ресурсов.

1.2. Свёрточная нейронная сеть

Один из главных компонентов любой современной нейронной сети, обрабатывающей изображения или видео – свёрточные слои. Формально свёртка (convolution) – это операция над двумя матрицами $A^{n\times m} = \left(a_{i,j}\right)$ и $B^{l\times k} = \left(b_{i,j}\right)$, результатом которой является матрица $C^{n-m+1\times m-k+1}$. Каждый элемент результата вычисляется как скалярное произведение матриц B и $\bar{A}$, где $\bar{A}$ – подматрица A размера $(l \times k)$. Вид матрицы $\bar{A}$ определяется положением элемента $\mathrm{c}_{i,j}$:

$$\mathrm{c}_{i,j} = \sum_{u=0}^{l-1}\sum_{v=0}^{k-1} a_{i+u,j+v}b_{u,v}\,, \tag{5}$$

Матрица B обычно называется ядром или фильтром свёртки. Визуально всю операцию можно представить как перемещение фильтра по матрице A и умножение наложенных друг на друга элементов матрицы и ядра. Область, которую охватывает ядро, называется поле восприятия (receptive field). Пример полного применения операции свёртки представлен на рис.1

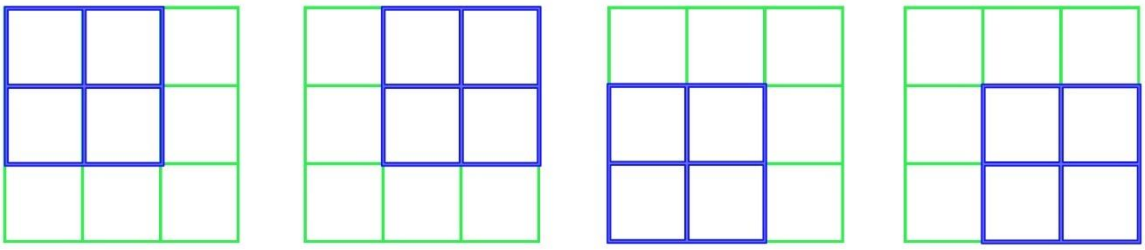


Рисунок 1 – Последовательность применения свёртки с ядром 2x2 (выделено тёмным синим цветом) к матрице размера 3x3 (выделена светлым зелёным цветом).

Результат применения ядра свёртки к матрице можно рассматривать как число, реагирующее на определённую структуру элементов матрицы (или пикселей конкретного канала изображения) внутри поля восприятия. Характер

определяемой структуры зависит от параметров фильтра, которые являются обучаемыми параметрами. Результирующее изображение часто называют картой признаков (feature map). На рисунке 2 можно увидеть примеры применения свёрток с фильтрами размером 3x3 к изображению.

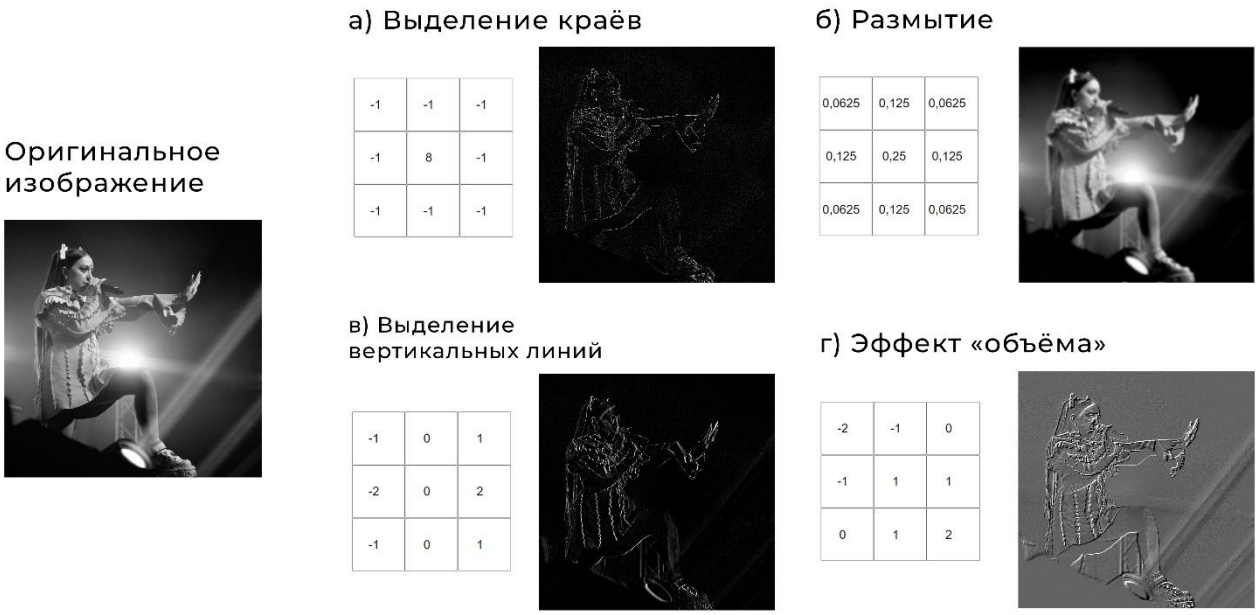


Рисунок 2 – Результат применения различных ядер свёрток к изображению. Для упрощения, оригинальное изображение имеет всего один канал. Для каждого примера под буквами от а) до г) приведены соответствующие фильтры и результат, полученный применением каждого ядра.

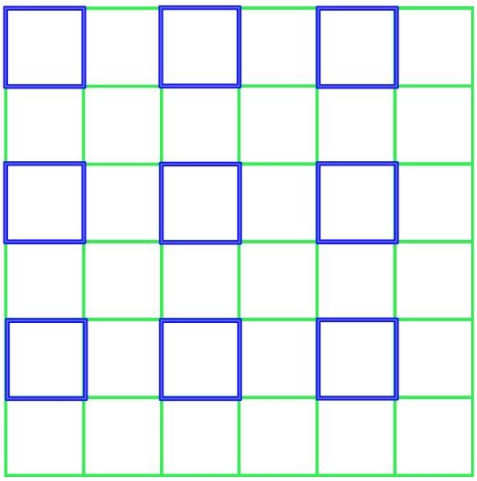
Каждый пиксель результирующего изображения под буквой б) является результатом простого сложения значений всех пикселей вокруг него на оригинальном фото – значения пикселей результата вычисляются путём «смешения» пикселей вокруг, и итоговое изображение получается размытым. Под буквой в) происходит похожее действие, только пиксели результирующего изображения реагируют на изменение значений оригинальных пикселей с тёмных на яркие в диагональном направлении, и результат выглядит как

объёмное изображение, «освещённое» источником света в правом нижнем углу.

Под буквами а) и в) пиксели результирующего изображения отражают наличие на исходном линий – фильтр а) реагирует на любые линии, в) – на вертикальные.

Применение свёрток внутри сети объединяется в свёрточные слои. Свёрточный слой обычно имеет свои параметры. В их числе количество входных каналов (цветное изображение будет иметь 3 канала, чёрно-белое – 1), количество выходных – зачастую внутри свёрточного слоя располагается несколько параллельных свёрток, каждая из которых обучается отдельно. Каждая свёртка применяется к изображению независимо от других, на выходе все результирующие изображения конкатенируются в один массив.

Важными параметрами свёрточного слоя также являются размер ядра, шаг (stride) – количество пикселей или элементов входной матрицы, на которое каждый раз сдвигается ядро. Например, на рисунке 1 шаг равен единице – между применениями фильтр сдвигается на одну ячейку по горизонтали и вертикали. Одним из самых используемых параметров является также дополнение нулями или паддинг (padding) – благодаря нему вокруг исходной матрицы формируется рамка из нулей заданной толщины, что позволяет влиять на размер выходного изображения. Последнее значение, связанное со свёртками – расширение (dilation), или расстояние между ячейками ядра свёртки. На рисунке 1 dilation = 1, на рисунке 3 приведён пример, в котором dilation = 2.



0	0	0	0	0	0
0	7	0	-6	9	0
0	2	8	-3	-5	0
0	1	3	4	4	0
0	5	-8	-4	2	0
0	0	0	0	0	0

Рисунок 3 – Параметры свёрточных слоёв. Слева: применение фильтра 3x3 с dilation=2 (тёмно-синие клетки) к матрице 6x6 (светло-зелёные клетки). Справа: применение padding=1 (тёмно-синие клетки) к матрице 4x4 (светло-зелёные клетки).

Каскад из нескольких свёрточных слоёв – основа свёрточной нейронной сети (convolutional neural network, CNN) [23]. Свёртки на первых слоях обучаются реагировать на простые признаки вроде прямых линий и контуров, как на рисунке 2, более глубокие слои получают на вход карты простых признаков и обнаруживают похожие признаки на них – в итоге более глубокие слои реагируют на простые элементы «карт простых элементов», то есть, на сложные композиции простых признаков.

Однако свёрточные слои не так хороши в обработке данных, как слои полносвязной нейронной сети, поэтому карты признаков, полученных серией свёрточных слоёв, обычно передаются в несколько полносвязных для финальной обработки. Перед этим карты признаков представляются в виде вектора – все ряды матриц в картах конкатенируются, и матрицы переводятся в уплощённое представление (flatten). На основании информации в таком виде свёрточные слои могут произвести, например, классификацию или детектирование объектов на изображении.

Впервые архитектура свёрточной нейронной сети была описана Яном Лекуном в 1988 году [24], и с тех пор появилось множество вариаций подобных сетей. В контексте данной работы наиболее интересна модель ResNet (2015) [25]. Основная её особенность – наличие соединений быстрого доступа (shortcut connections) между блоками свёрточных слоёв. Схема работы подобного соединения представлена на рисунке 4.

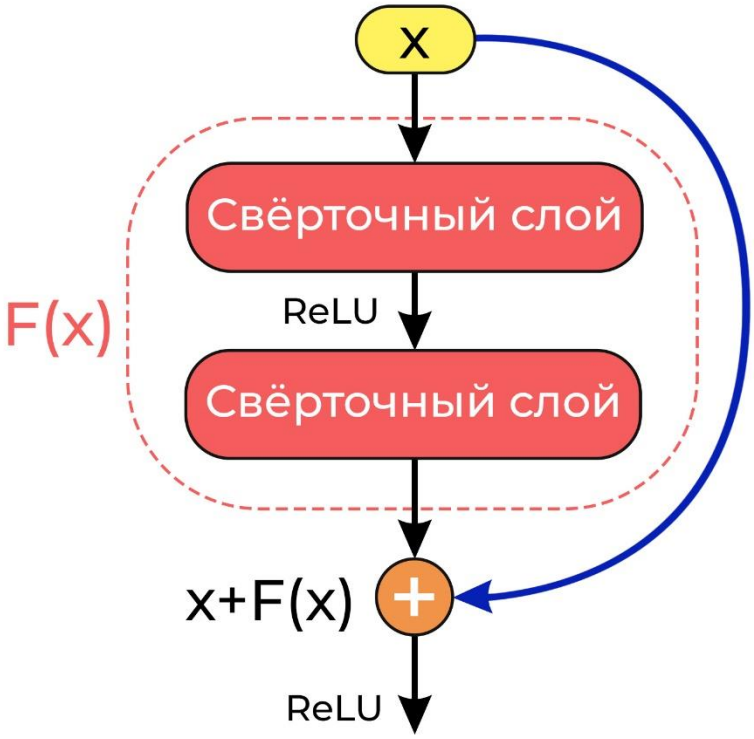


Рисунок 4 – Устройство блока с соединением быстрого доступа.

Информация проходит через блок из нескольких свёрточных слоёв (в данном случае – двух), обозначенный как F(x). После этого результат применения свёрточного блока складывается с оригинальной информацией. Соединение быстрого доступа обозначено синей стрелкой. Стрелки, подписанные “ReLU”, подразумевают применение функции активации при переходе между слоями – в данном случае, это функция ReLU.

Смысл применения блоков с соединением быстрого доступа в том, чтобы передавать в сеть не только данные карт признаков, но и часть оригинальных

данных. Блоки, подобные изображённым на рисунке 4, называются остаточными (residual) блоками, а нейронные сети, использующие такие блоки, называются остаточными сетями (residual networks).

Кроме классических свёрток, существуют ещё и так называемые обратные или транспонированные свёртки (transposed convolutions) [26, с.20]. Главная область их применения – увеличение разрешения входных данных. Транспонированную свёртку можно представить как обычную, параметры которой подбираются таким образом, чтобы входное изображение увеличивалось в размере. Пример приведён на рисунке 5 .

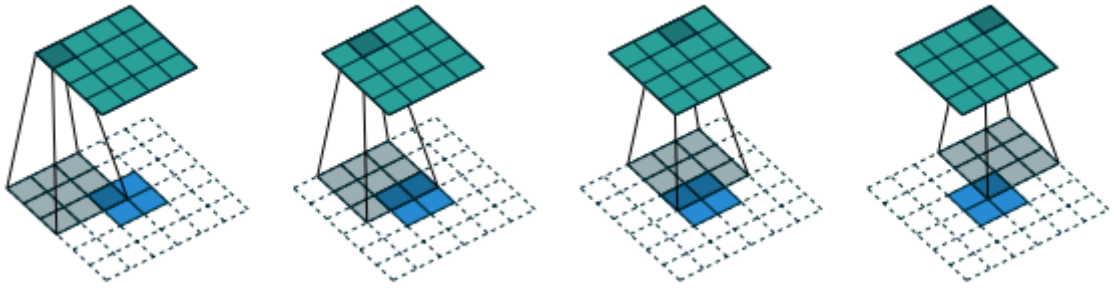


Рисунок 5 – Транспонированная свёртка преобразует нижнее изображение размера 2x2 пикселя (выделено синим) в верхнее изображение размером 4x4 (выделено зелёным). Эту операцию можно рассматривать как применение обычной свёртки с размером ядра 3x3 и значением padding=2. Иллюстрация взята из статьи [26].

1.3 Автокодировщики

Автокодировщики (Autoencoders, автоэнкодеры) – это группа моделей глубокого обучения, цель которых – сжатие данных в небольшое представление, сохраняющие основные их черты и декодирование обратно в оригинальный вид с минимальными потерями полезной информации [27]. За кодирование и декодирование отвечают части, которые обозначаются как

Encoder (энкодер, кодировщик) и Decoder (декодер, декодировщик). Сжатое представление данных часто называют латентным представлением, латентным кодом или вектором, потому что его можно рассматривать как вектор или набор векторов из некоторого многомерного пространства, размерность которого обычно меньше, чем размерность оригинальных данных. Схема автокодировщика представлена на рисунке 6.

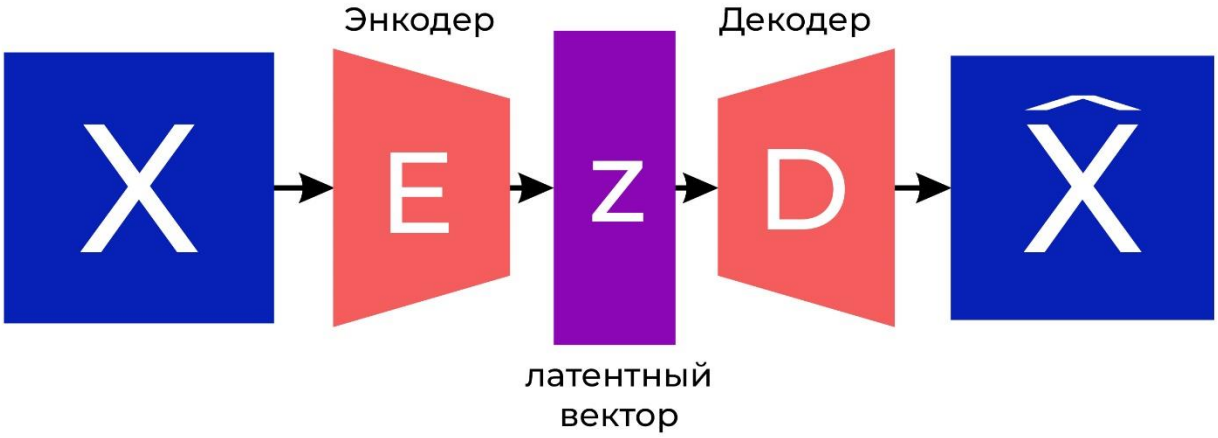


Рисунок 6 – Схема автоэнкодера. Входные данные X кодируются энкодером E в латентный вектор z. Затем декодер D восстанавливает из z данные $\widehat{X}$, стараясь минимизировать разницу между X и $\widehat{X}$.

Формально, если имеются энкодер $E: \mathbb{Q}^n \to \mathbb{Q}^m$, декодер $D: \mathbb{Q}^m \to \mathbb{Q}^n$, и входная информация $x \in X \subset \mathbb{Q}^n$ то задача обучения модели может быть описана как

$$L\big(x, D \circ E(x)\big) \to \min_{\Theta}, \tag{6}$$

где Θ – множество параметров энкодера и декодера, L – функция ошибки, часто обозначаемая как ошибка восстановления (reconstruction loss). При обучении автокодировщиков на изображениях, чаще всего эту роль выполняет упомянутая выше функция MSE.

Подобные алгоритмы могут быть использованы для решения множества задач. Во-первых, с их помощью можно сжимать информацию, например, для

систем распознавания лиц, которые должны хранить и обрабатывать большие объёмы данных. Это не просто помогает экономить место в хранилище, но и позволяет сравнивать между собой не фотографии лиц, а их меньшие латентные представления, содержащие основную информацию [28]. Во-вторых, при выполнении генеративных задач часто применяется подход, при котором генерируется не весь объём нужной информации, а её латентное представление, которое после можно перевести в заданный формат с помощью автокодировщика. Именно это применение подобных моделей имеет наибольшее значение в контексте данной выпускной работы.

Автокодировщики не получают никаких задач, кроме минимизации потерь при сжатии и реконструкции входной информации. Несмотря на это, латентные коды данных с похожими признаками – если мы рассматриваем изображения лиц, это могут быть, например, лица людей одной расы – обычно кодируются похожим образом – если рассматривать такие коды в виде векторов, они с большей вероятностью будут сонаправленными.

Эта особенность открывает возможности для интересного использования. Можно, например, вычислить векторы для всех лиц, на которых есть улыбка, вычислить среднее значение каждой кооординаты, и вычесть из получившегося усреднённого вектора такой же вектор для всех изображений с грустным выражением лица. Результатом будет «средний вектор улыбки»:

$$v_{smile} = \frac{1}{n}\sum_{i=1}^{n} v_i - \frac{1}{m}\sum_{j=1}^{m} v_j\,, \tag{7}$$

где n – число всех лиц с улыбкой m – число хмурых лиц. Если прибавить v_{smile} к любому изображению лица и декодировать результат, на оригинальном изображении появится улыбка. Выполнение подобной операции с автокодировщиком, обученном на датасете Labeled faces in wild [29] представлено на рисунке 7. При внимательном рассмотрении лиц на рисунке

можно заметить, что проводимые операции влияют не только на наличие улыбки, но и на освещение лица, форму носа, наличие макияжа и контрастность кадра. Это означает, что фотографии из датасета, содержащие улыбку, в среднем более «смещены» в каждом из перечисленных выше параметров.

Рисунок 7 – Добавление улыбки с помощью автоэнкодера. Первое изображение слева – изображение лица, сжатое и восстановленное автокодировщиком. К латентному представлению каждого следующего изображения представляется «вектор улыбки», умноженный на 0,2. Первое изображение справа декодировано из суммы латентного представления первого фото слева и полного вектора улыбки.

Оригинальные автоэнкодеры сжимают входные данные в один вектор, и распределение всех латентных векторов обучающего датасета обычно рассматривается как равномерное. Однако такой подход уступает в качестве кодирования информации другому, – в котором распределение латентных кодов задаётся в качестве одного из параметров сети [30]. Чаще всего латентные векторы предполагаются распределёнными в соответствии с гауссовским многомерным распределением, в самом простом случае выбирается стандартное нормальное. Автоэнкодеры, использующие латентные пространства такого типа, называются вариационными (Variational autoencoder, VAE).

VAE кодирует не сами латентные векторы, а параметры распределения для них (в случае распределения Гаусса это векторы математического

ожидания и стандартного отклонения), при восстановлении информации она сначала сэмплируется из заданного распределения с вычисленными ранее параметрами в точку латентного пространства, и затем декодируется, как обычно.

С точки зрения обучения, всё ещё остаётся задача уменьшения ошибки восстановления, описанная для классических автоэнкодеров. Кроме того, необходимо обучать сеть так, чтобы учитывалось желаемое латентное распределение. Для этого в функцию ошибки добавляется новое слагаемое – дивергенция Кульбака — Лейблера, часто обозначаемая как «КЛ-дивергенция» или «KL loss» и являющаяся мерой схожести для двух вероятностных распределений [30, с.4].

Общая формула дивергенции Кульбака-Лейблера для распределений p и q векторов x из множества X:

$$D_{KL}(p||q) = -\sum_{x \in X} p(x)\, log\, \frac{q(x)}{p(x)} \tag{8}$$

Рассмотрим пример: КЛ-дивергенция между некоторым распределением латентных векторов z, зависящих от оригинальных данных x и стандартным нормальным распределением будет иметь вид

$$D_{KL}(p(z|x) \,||\, \mathrm{N}(0, I)) = -\frac{1}{2}\sum_{i=1}^{Z}\left(1 + log\left(\sigma_i^2\right) - \mu_i^2 - \sigma_i^2\right), \tag{9}$$

где Z – общее количество данных в датасете, σ_i и μ_i – координаты векторов стандартного отклонения и математического ожидания для каждого вектора z.

Лосс-функция для обучения VAE принимает общий вид

$$L_{VAE} = -D_{KL}(p\,(z|x)||q(z)) + L_{reconstruction}, \tag{10}$$

где $\mathrm{q(z)}$ – желаемое распределение латентных векторов, $p(\mathrm{z|x})$ – распределение, полученное после кодирования входных данных x, $\mathrm{L}_{reconstruction}$ – ошибка восстановления.

Вариационные автоэнкодеры кодируют данные достаточно хорошо, чтобы с помощью обученной модели можно было генерировать новую

информацию путём сэмплирования (выбора случайного вектора) из латентного распределения. На рисунке 8 приведены примеры генерации с использованием вариационного автокодировщика.

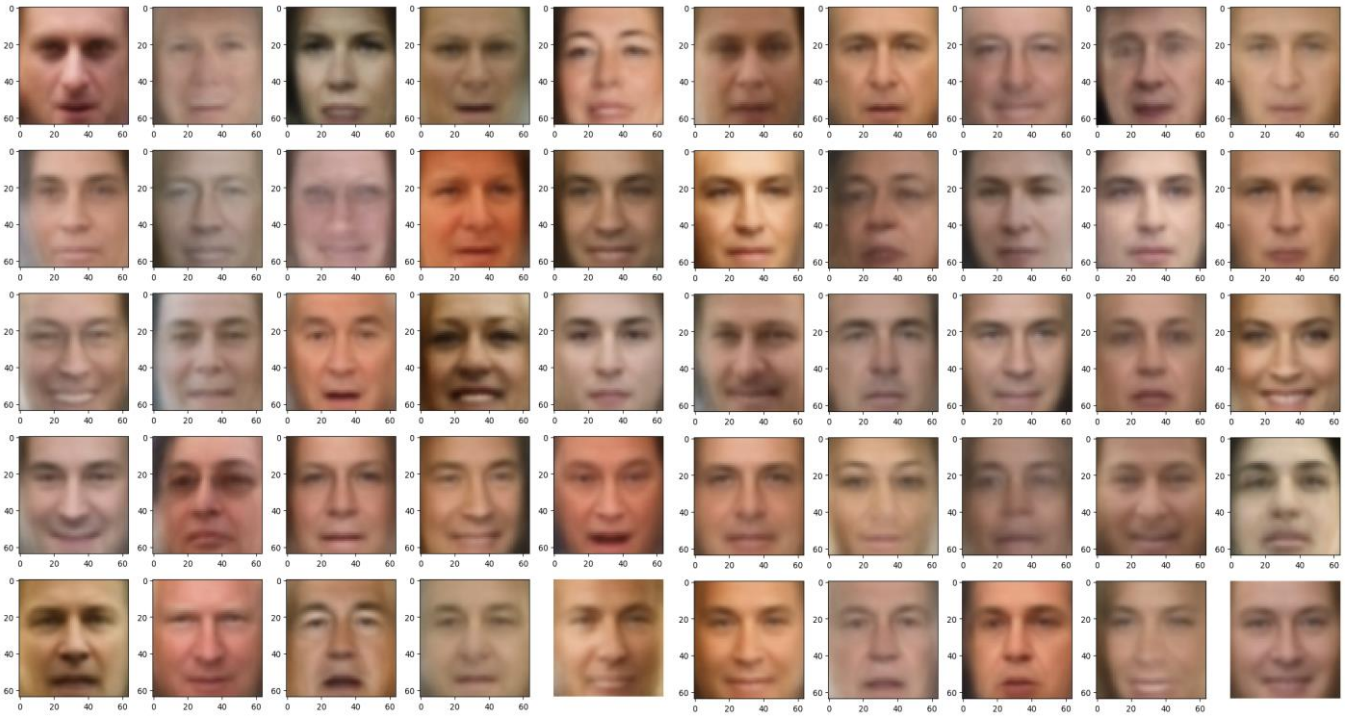


Рисунок 8 – Примеры лиц, сгенерированные вариационным автокодировщиком. Модель получала на вход случайный вектор из многомерного распределения Гаусса, и обрабатывала его с помощью своего декодера.

Развитие автокодировщиков не остановилось на вариационной версии архитектуры. В 2018 году был предложен следующий шаг в развитии этого семейства моделей – вариационные автоэнкодеры с векторной квантизацией (Vector-quantized VAEs, VQ-VAE) [11].

Главное отличие VQ-VAE от всех предыдущих версий – использование дискретных латентных кодов вместо непрерывных. Выходная информация из декодера заменяется набором векторов, называемых эмбеддингами (embedding) из словаря (codebook) фиксированного объёма. Координаты векторов в словаре являются обучаемым параметром сети – это позволяет

модели выучивать оптимальное распределение латентных векторов, чего не могли делать предыдущие версии. Схема устройства VQ-VAE представлена на рисунке 9.

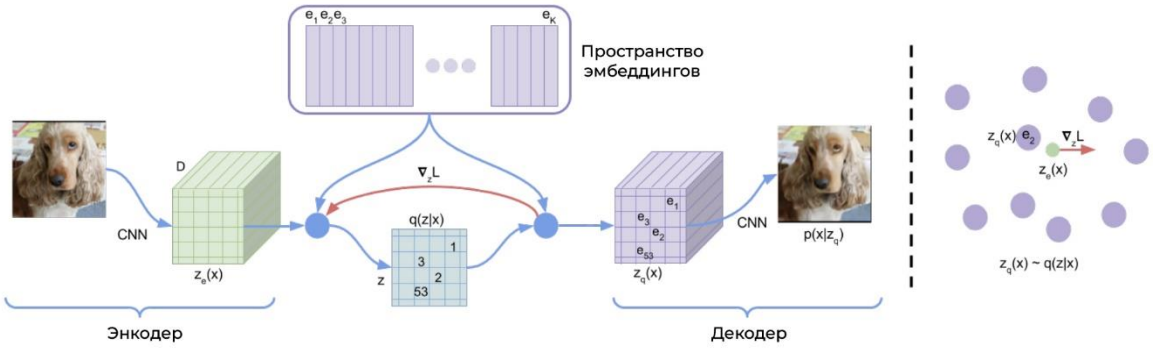


Рисунок 9 – VQ-VAE. Слева – схема работы вариационного автоэнкодера с векторной квантизацией. Справа – визуализация пространства эмбеддингов. D карт признаков, полученных из энкодера, представляются в виде набора D-мерных векторов, каждый из которых заменяется ближайшим к нему эмбеддингом из словаря. Иллюстрация взята из оригинальной статьи, в которой был предложен VQ-VAE [11, с.4].

Для обучения полной сети нужно обучить энкодер и декодер – для этого, как и в обычных автоэнкодерах, вычисляется reconstruction loss. Помимо этого, обучается квантователь – часть модели, отвечающая за переход от выхода энкодера к набору эмбеддингов (этот процесс называется квантованием). Для этого к функции ошибки добавляются ошибка квантования (vector quantization loss) и ошибка приверженности (commitment loss) [11, с.4]:

$$L_{vq} = \|sg[z_e(x)] - e\|_2^2, \qquad L_{comminment} = \|z_e(x) - sg[e]\|_2^2, \qquad (11)$$

где $z_e(x)$ – выход энкодера, e – эмбединги, $sg[\ .\ ]$ – оператор остановки градиентов. Его применение означает, что градиенты для функции ошибки не будут вычисляться для аргумента оператора – то есть, в первом случае,

градиенты будут вычислены только для квантователя, который отвечает за координаты эмбеддингов, а во втором – только для энкодера. $\| \|_2^2$ – L_2-норма.

Благодаря ошибке квантования, координаты эмбеддингов двигаются квантователем в сторону координат выходных данных энкодера на каждом шаге обучения. Ошибка приверженности – наоборот, обучает энкодер так, чтобы координаты выходного набора векторов двигались ближе к координатам эмбеддингов. Это позволяет ограничить «разброс» выходных данных кодировщика.

Полная функция ошибки для VQ-VAE принимает вид:

$$L_{VQ-VAE} = L_{reconstruction} + L_{vq} + \beta L_{comminment}. \quad (12)$$

Здесь β – стоимость приверженности (commitment cost) – отвечает за соотношение между ошибками приверженности и квантования. Как правило, выбирается значение $\beta = 0{,}25$, то есть, больший приоритет при обучении отдаётся квантованию.

VQ-VAE – это основа наиболее совершенной версии автокодировщика, которая сегодня используется в большинстве генеративных моделях, связанных с автокодировкой, – иерархически организованного VQ-VAE, или VQ-VAE-2 [31]. Суть этой архитектуры в кодировании нескольких наборов кодов вместо одного, с использованием отдельных словарей для каждого. Схема иерархического автокодировщика с векторным квантованием приведена на рисунке 10.

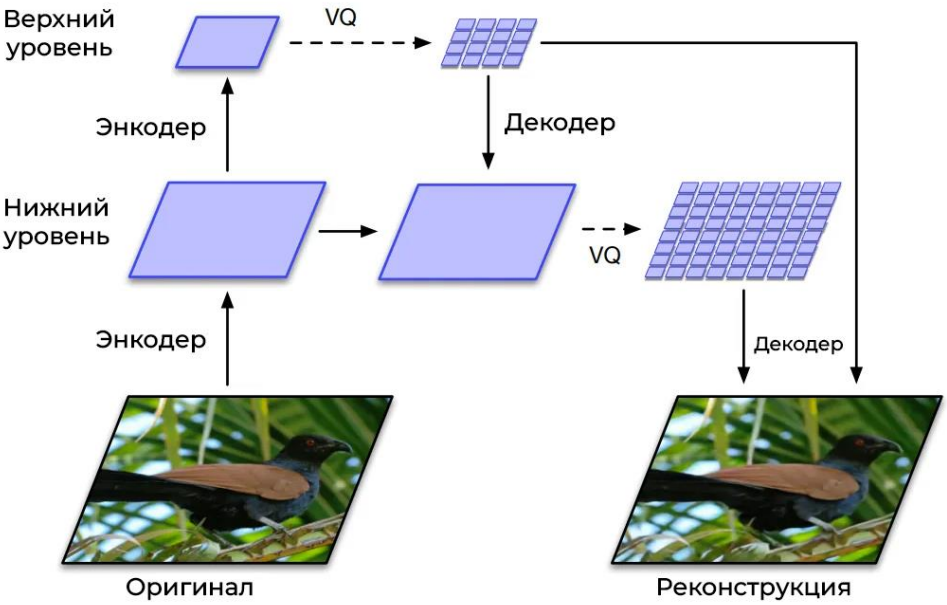


Рисунок 10 – Иерархически организованный VQ-VAE. Входное изображение сначала кодируется в представление нижнего уровня, которое затем кодируется и квантуется в представление и коды верхнего уровня. После этого верхние коды обрабатываются отдельным декодером, восстановленный результат конкатенируется с нижним представлением и квантуется в нижние коды. Затем декодер нижнего уровня восстанавливает изображение из нижних и верхних кодов.

Переведённая на русский схема из статьи о VQ-VAE-2 [31, с.4].

Смысл использования разных уровней кодов – в том, что они могут хранить разные данные. В коды верхнего уровня, как правило, записывается глобальная информация – такая, как цвет и форма, поскольку верхние коды обрабатываются двумя энкодерами и содержат информацию о более сложных и обобщённых признаках. Коды нижнего уровня же содержат информацию о локальных признаках и деталях. Обычно используются два уровня представления информации, но в случаях, когда качество восстановления признаётся недостаточным, может быть применён трёхуровневый VQ-VAE-2 [31, с.4].

Для каждого из уровней обучается своя пара «кодировщик – декодировщик» и свой квантователь. Для обучения используется та же функция ошибки, что и для оригинального VQ-VAE, которая вычисляется отдельно на каждом уровне.

1.4 Трансформерная архитектура

Задачу генерации аудио можно рассматривать как задачу генерации последовательности звуковых сигналов (звуковой волны). Как уже говорилось во введении к этой работе, в 2017 году в статье «Attention is All You Need» была предложена архитектура, которая сегодня является одним из наиболее частых решения для задач генерации последовательностей – трансформер.

Такая нейросеть в общем случае похожа на автокодировщики тем, что также состоит из энкодера и декодера. Существенные отличия наблюдаются в устройстве этих частей и связей между ними. Кодировщик и декодировщик трансформера состоят из одинаковых слоёв специального вида для каждой части. Схема трансформера представлена на рисунке 11.

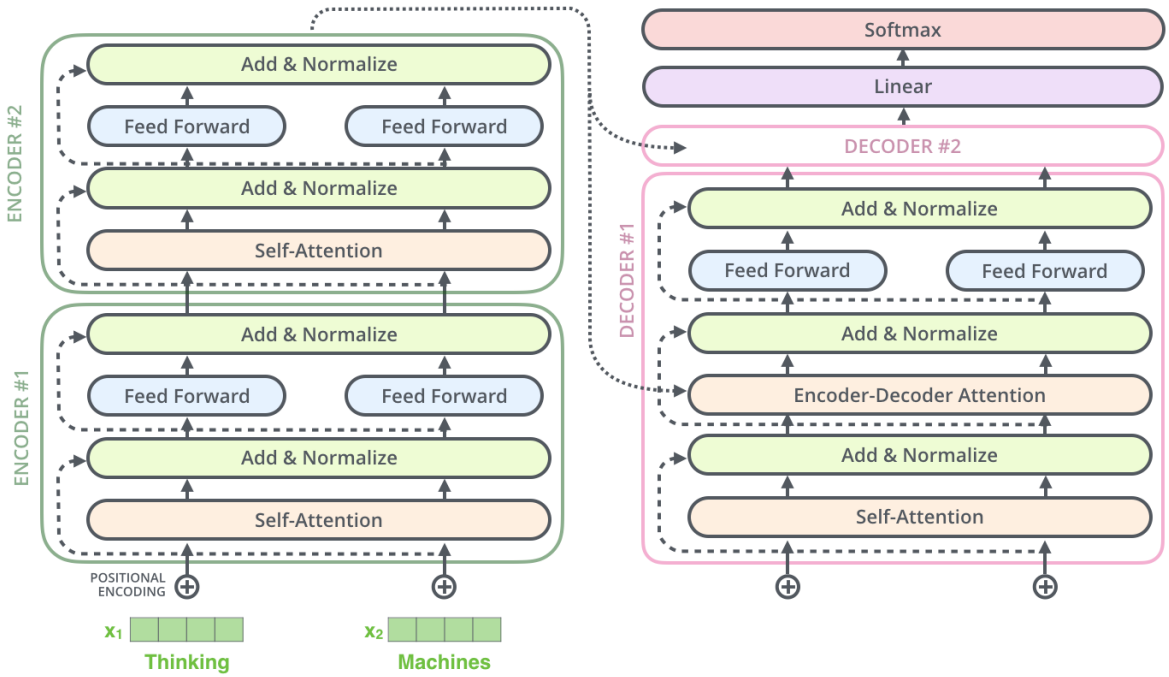


Рисунок 11 – Архитектура трансформера. Слева изображены два слоя энкодера, справа – декодер и выходные слои сети. Иллюстрация взята из [32].

Энкодер в трансформере состоит из двух частей: механизма самовнимания (Self-Attention) и нескольких полносвязных слоёв, обозначенных на рисунке 11 как «feed-forward». Выходы из этих слоёв складываются с входами по описанному в параграфе 1 принципу остаточных соединений, после чего нормализуются и передаются в следующие слои.

Подробнее стоить остановиться на механизме самовнимания. Он играет в трансформерах ключевую роль – помогает сети лучше «запоминать» входные последовательности и взвешивать значимость каждой части последовательности для всех других частей – получается, что слова или сигналы «обращают внимание» друг на друга. Части, на которые разбиваются исходные последовательности, называются токенами. Это могут быть как слова, так и отдельные символы – зависит от выбранного способа разбиения на токены (токенизации). Токены затем преобразуются трансформером в эмбеддинги – векторы, с которыми будет работать сеть. Способ кодирования эмбеддингов также обучается сетью, чтобы получать максимально «удобные» для работы входные данные.

Начало работы механизма внимания происходит, когда входные эмбеддинги умножаются на три обучаемые трансформером матрицы W_q, W_k, W_v, называемых матрицами запроса (query), ключа (key) и значения (value). Результат этой операции – три вектора q, k, v для каждого эмбеддинга являющиеся его векторами запроса, ключа и значения.

После этого происходит оценка связей между словами – вычисление внимания. Внимание может быть любой функцией двух векторов, принимающей положительные значения и отражающей их совместное расположение в пространстве – самый простой вариант – это нормированное скалярное произведение (Scaled Dot-Product Attention). Вектор запроса q для каждого эмбеддинга скалярно умножается на векторы ключей k для всех остальных эмбеддингов. Результат этого действия – набор значений

«важности» каждого элемента последовательности для одного конкретного элемента – затем делится на корень квадратный из длины векторов k и пропускается через функцию softmax:

$$softmax(\mathrm{z_i}) = \frac{\mathrm{e}^{\mathrm{z_i}}}{\sum_{\mathrm{j}=1}^{\mathrm{K}} \mathrm{e}^{\mathrm{z_j}}}\ ,\ \mathrm{i} = 1,2,\dots,\mathrm{K}. \qquad (13)$$

Выход функции softmax – вектор, сумма значений которого равна единице. Можно рассматривать его как нормированные веса значимости слов в последовательности для каждого отдельного слова. Затем каждый вектор значения v умножается на полученный softmax-вектор, и векторы значений всех эмбеддингов складываются. В результирующем векторе больший «вес» имеют самые «важные» для конкретного эмбеддинга части последовательности. Затем каждый такой вектор независимо передаётся в полносвязную нейросеть, а из неё – в механизм внимания следующего слоя энкодера. Все части исходной последовательности обрабатываются энкодером параллельно.

Зачастую вместо обычного внимания используется многоголовое (multi-head attention). В его основе – несколько слоёв внимания, «голов», которые обрабатывают эмбеддинги параллельно. Результаты обработки конкатенируются между собой и так же, как в обычном случае, передаются в полносвязные слои. За счёт того, что матрицы W_q, W_k, W_v внутри каждой головы внимания обучаются независимо, каждая голова внимания может воспринимать свой контекст и обращать внимание на связь между эмбеддингами в уникальной форме.

Декодер в трансформере также состоит из нескольких одинаковых слоёв, но имеет свои особенности. Самовнимание в декодере является маскированным (Masked Self-Attention): при обработке декодером элемента последовательности на шаге i, все её элементы на шаге i+1 скрываются маской, не позволяя декодеру использовать их эмбеддинги. Это важно для обучения

пошаговой генерации последовательностей, принципы которой объяснены ниже. Кроме того, некоторые attention-слои используют не самовнимание, а внимание между энкодером и декодером (Encoder-Decoder Attention). В нём векторы значений и ключей берутся для выходов энкодера, а векторы запросов – для входа текущего слоя декодера. Это позволяет эффективно обращать внимание на входящую в декодер последовательность. Как и в энкодере, после каждого слоя декодера используются слои многослойного перцептрона.

При решении задачи генерации, выход декодера обычно поступает в один слой полносвязной сети, количество нейронов в котором равно количеству всех возможных элементов последовательности (размеру словаря). Этот слой часто называется линейным слоем, поскольку он не применяет функцию активации. Над выходом этого слоя затем выполняется softmax-преобразование, чтобы определить наиболее вероятное следующее слово в последовательности.

Для работы трансформера также важно позиционное кодирование входной последовательности: без него при обработке определённого слова или токена, модель не будет иметь представления о его положении в последовательности. Чтобы этого избежать, ко всей последовательности прибавляются значения специальной функции (часто её роль играет синус или косинус с достаточно большим периодом) от позиций слов.

Итак, схема работы трансформерной модели имеет вид:

1. Входные данные преобразуются в эмбеддинги, к каждому эмбеддингу прибавляется код позиции.
2. Векторные представления элементов обрабатываются энкодером и передаются в декодер
3. После обработки эмбеддингов декодером, выходная информация передаётся в линейный слой и функцию softmax, которая возвращает вероятность появления в последовательности для всех элементов

словаря. Один из этих элементов выбирается следующим в последовательность.

4. Выбранный элемент добавляется в исходную последовательность. Полученная последовательность снова подаётся в трансформер, чтобы определить следующий её элемент, пока не сгенерируется последовательность нужной длины или не будет получен специальный токен конца последовательности.

Выбор следующего элемента последовательности на основании выхода softmax-функции называется сэмплированием. Существуют разные его виды.

Наивное сэмплирование – способ, носящий лишь теоретическое назначение. Его суть заключается в выборе случайного элемента из словаря. Очевидно, вывод модели при его использовании будет слишком случайным и хаотичным – в случае с генерацией звука, он будет напоминать белый шум.

Противоположность наивного – жадное сэмплирование (greedy sampling) подразумевает выбор самого вероятного элемента на каждом шаге. Этот способ может хорошо работать для правильно обученных моделей, но у него есть минус – наиболее вероятные исходы часто бывают слишком общими и неинформативными (именно поэтому модель выбирает их следующими вариантами – такие элементы могут стоять после многих других, не неся в себе полезной информации), в результате вывод модели может получиться не таким богатым и разнообразным, как для других случаев сэмплирования.

Top-k сэмплирование – способ, при котором выбирается случайный вариант из k самых вероятных. Это позволяет разнообразить вывод нейросети, ограничив при этом его хаотичность и случайность.

Top-p сэмплирование похоже на предыдущее: оно подразумевает выбор случайного элемента из нескольких наиболее вероятных, суммарная вероятность (значение функции softmax) выбора которых меньше заданного числа p ($0 < p < 1$).

Для обучения трансформерных моделей удобно использовать функцию ошибки, называемую кросс-энтропией (cross entropy loss). Это мера различия между множествами, с помощью которой можно определить, насколько генерация модели отличается от нужных данных. Формула для кросс-энтропии имеет вид

$$CrossEntropy = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\, log(p_{ic}), \tag{14}$$

где N – количество элементов, C – количество классов (в случае с трансформером – объём словаря), y_{ic} – c-я координата one-hot вектора для y_i. Она будет равна единице, если y_i принадлежит классу c, и нулю в остальных случаях. p_{ic} – предсказанная моделью вероятность принадлежности y_i классу c, получаемая в результате softmax-преобразования.

Глава 2. Построение алгоритма

Построенный алгоритм для обусловленной генерации звука совмещает в себе использование описанных выше подходов. В процессе проектирования была выбрана архитектура, похожая на архитектуру Im2Wav [7], но адаптированная для использования в условиях ограниченных вычислительных ресурсов. Схема алгоритма представлена на рисунке 12.

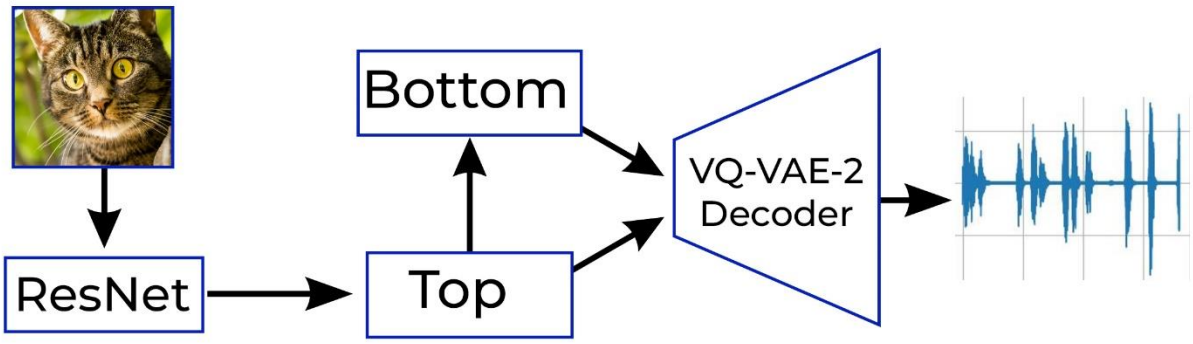


Рисунок 12 – Схема построенного алгоритма. Входное изображение, поступая в алгоритм, обрабатывается предобученной сетью ResNet, которая определяет принадлежность объекта на фото к некоторому классу. Метка класса затем поступает в трансформерную модель Top, которая генерирует коды верхнего уровня для VQ-VAE-2. На основании этих кодов трансформерная модель Bottom генерирует коды нижнего уровня. Из набора верхних и нижних кодов восстанавливается звук с помощью декодера VQ-VAE-2

Все программы, необходимые для построения алгоритма, были написаны на языке программирования Python. Для программирования и обучения нейронных сетей использовался фреймворк Pytorch, для работы с аудиофайлами – библиотека torchaudio, входящая в Pytorch. Для оптимизации математических вычислений использовалась библиотека Numpy, для работы с

данными – библиотека Pandas. Для визуализации данных при обучении и анализе работы нейросетей применялась библиотека Matplotlib.

Большая часть вычислений при обучении сетей и обработке данных с их помощью проводилась на графическом ускорителе Nvidia Tesla P100, бесплатный доступ к которому предоставляет платформа Kaggle.

2.1 Данные

Проводить обучение моделей изначально планировалось на датасете VGG-Sound – это крупное собрание видеофайлов, содержащее более 200 тыс. пар «видео – аудио», разбитых на 300 классов по источникам звука [33]. В целях экономии ресурсов из оригинального датасета были взяты только данные двух классов: кошек и собак, но величина набора данных осталась внушительной – около 6 тысяч пар.

Однако у VGG-Sound есть ряд проблем. Главная из них – факт, что датасет состоит из различных видео, находящихся в открытом доступе на YouTube, многие из которых содержат некачественные или несогласованные аудио и видео. В целом, данные являются достаточно «шумными»: и среди видео, и среди соответствующих им звуков, достаточно брака, который влияет на обучение сетей. При наличии больших вычислительных мощностей, можно было не бояться обрабатывать такой звук: например, потратить больше времени на обучение моделей больших масштабов, но для демонстрации решения задачи генерации такие шаги будут избыточны.

В дальнейшем для обучения нейронных сетей был выбрана выбрана часть датасета «Acoustic Event Dataset» [34], также содержащая только звуки собак и кошек из оригинала, но состоящая из гораздо более чистых записей. Он гораздо меньше масштабом – всего 275 файлов, но позволяет лучше обучать модели на небольших вычислительных ресурсах, что будет продемонстрировано далее. Что же касается видео и изображений – для

решения задачи обусловленной генерации обучение на согласованных парах видео и аудио, взятых из одного файла, теоретически позволяет нейронным сетям лучше понять зависимость звука от входных изображений – например, выучить тембр собак определённой породы или заметить, что мяуканье котят звучит выше, чем у взрослых кошек. Но при оперировании на небольших вычислительных мощностях уменьшается вероятность построить модели, которые смогут выучить такие взаимосвязи.

Кроме того, обучение компонентов алгоритма не подразумевает обязательное наличие общего источника у обучающих данных для каждой отдельной модели, главное требование – их корректное функционирование в рамках отдельных задач. Всё это позволяет отказаться от использования строго согласованных пар данных.

2.2 Способы обработки звука и автокодировщики

Первой реализованной нейросетью из алгоритма является звуковой автокодировщик. Трансформерные модели по плану должны генерировать коды для него, поэтому именно автоэнкодер должен быть обучен первым.

Для обработки звука нейронными сетями используются различные способы его представления. «Сырые» аудиоданные, представляющие собой последовательность сигналов с заданной частотой дискретизации (количество изменений сигнала в секунду), используются, например, в работе алгоритма Im2Wav, значение частоты дискретизации обрабатываемого им аудио – 16 кГц [7, с.3].

Библиотека torchaudio предлагает разные способы предобработки звука, наиболее интересным и распространённым среди которых является мел-спектрограмма – способ изобразить зависимость частоты сигнала от времени. Пример алгоритма, использующего мел-спектрограммы в обучении – AudioGen [9].

Работа с сырым аудио при высокой частоте дискретизации требует использования больших вычислительных мощностей, превышающих максимальную мощность ускорителя P100. Из-за этого необходимо было выбрать способ обработки звука – либо уменьшать частоту дискретизации, теряя в качестве, либо преобразовывать звук в мел-спектрограммы и работать с ними.

Стоит заметить, что преобразования звука в спектрограмму и наоборот также предполагают некоторые потери в качестве звука. На этапе разработки алгоритма было принято обучить два автокодировщика – первый для работы с сырым аудио при небольшой частоте дискретизации и второй для обработки мел-спектрограмм. После этого можно сравнить показатели их качества и выбрать лучший для дальнейшей работы.

В начале работы была предпринята попытка обучить оригинальный VQ-VAE без иерархической структуры для работы со спектрограммами. Однако качество восстанавливаемых такой моделью спектрограмм было слишком низким для использования, поэтому в дальнейшем обучались только модели, подобные VQ-VAE-2. Пример реконструкций, полученных разными актокодировщиками, представлен на рисунке 13 (слева).

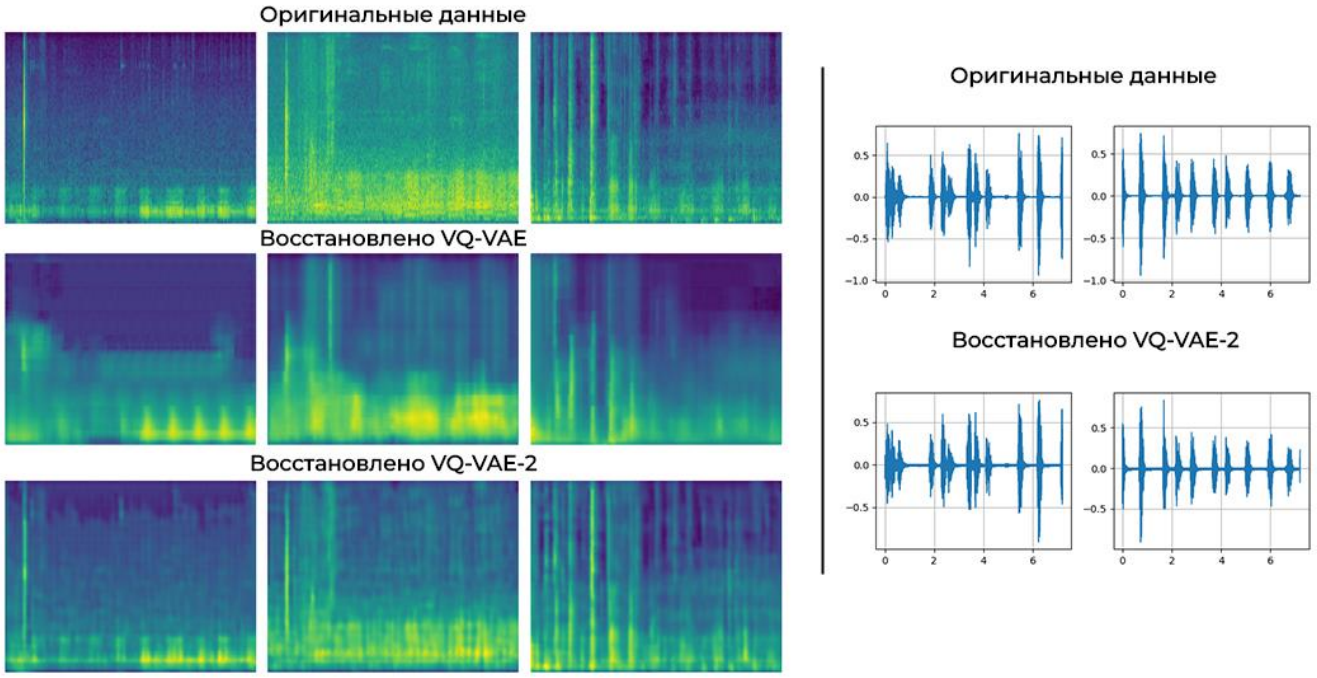


Рисунок 13 – Звуковые данные и их реконструкции. *Слева*: Верхний ряд: примеры спектрограмм из датасета. Средний ряд: восстановленные версии этих спектрограмм, полученные автокодировщиком без иерархической структуры. Нижний ряд: спектрограммы, восстановленные иерархическим VQ-VAE-2. *Справа*: Верхний ряд – оригинальные данные. Нижний – реконструкции, полученные VQ-VAE-2, обученном на сыром аудио.

Для автокодировщика, работающего с сырыми аудиоданными, частота дискретизации сжатых данных подбиралась экспериментально на основании ощущений от звука при разной частоте дискретизации. Минимальной частотой, сохраняющей приемлемое качество звука и уменьшающей объём обрабатывающей информации, была выбрана частота в 4 кГц. Вычислительных мощностей, предоставляемых Kaggle, оказалось достаточно для обработки файлов с такой частотой дискретизации. Примеры восстановленных данных приведены на рисунке 13 (справа).

Субъективное качество генерации автоэнкодеров для спектрограмм и сырого звука находится примерно на одном уровне. Однако кодировщик, работающий со спектрограммами, хуже кодирует звуки в латентные

представления, не делая почти никаких отличий между звуками кошек и собак. Сначала была выдвинута гипотеза о том, что причина плохого кодирования – зашумлённый датасет VGG-Sound, но после замены датасета на меньший проблема сохранилась. Автокодировщик, обрабатывающий сырые сигналы, кодирует данные в представление, позволяющее лучше разделить двумерные проекции данных на два класса, поэтому для итогового алгоритма была выбрана эта версия. Сравнение латентных представлений моделей на одном датасете представлено на рисунке 14.

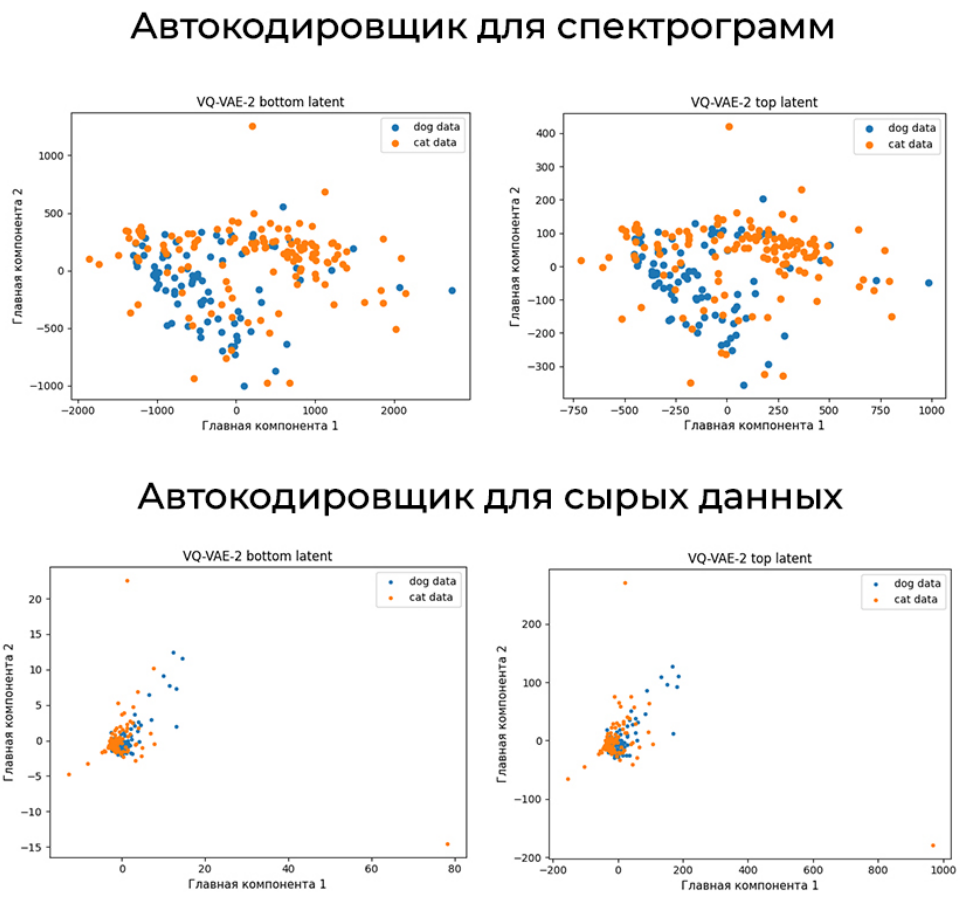


Рисунок 14 – Сравнение латентных представлений одинаковых данных разными автоэнкодерами. Верхний ряд: VQ-VAE-2, обученный на спектрограммах. Нижний ряд: VQ-VAE-2, обученный на сыром звуке. Иллюстрации получены с помощью метода главных компонент. Синими точками обозначены звуки собак, оранжевыми – кошек. Заметно, что классы на нижнем изображении разделимы лучше.

Поскольку сырые аудиоданные представимы в виде последовательности сигналов – вектора из чисел, для их обработки часто – в том числе, в данной работе, – используются одномерные свёртки. Такие свёртки работают также, как и двумерные, но вместо матрицы, которая обрабатывается с помощью фильтра-матрицы, в данном случае вектор данных обрабатывается фильтром-вектором, который последовательно движется слева направо с заданным шагом.

Для квантования данных в автокодировщике используются два словаря, каждый из которых содержит 2048 эмбеддингов размерности 32. Основной элемент кодирующей части – свёрточные слои со значением stride=2, каждый из которых уменьшает входные данные в два раза. Первое квантование применяется после трёх таких слоёв, второе – после дополнительных двух. Результатом этого является сжатие информации в 8 и 32 раз в кодах нижнего и верхнего уровня соответственно.

Между основными свёртками информация проходит через одинаковые блоки, состоящие из семи свёрточных слоёв с остаточными подключениями, которые не меняют размерность информации, но позволяют сети лучше распознать признаки в обрабатываемых данных. Для достижения этого в каждом слое блока задаётся значение stride=1 и выбираются одинаковые значения padding и dilation. Последовательность этих значений для семи слоёв увеличивается: 1, 2, 3, 6, 9, 18, 27, что означает увеличение поля восприятия каждого следующего слоя в блоке – это позволяет выделять не только локальные, но и глобальные признаки информации.

Декодеровщик повторяет устройство кодировщика, но вместо свёрток с шагом 2 использует транспонированные свёртки, увеличивающие данные в два раза. Структура блоков с остаточными подключениями при этом не изменяется.

2.3 Трансформерные модели

Для алгоритма было обучено два трансформера: модели Top и Bottom. Каждая модель состоит из 6 слоёв энкодера и 12 слоёв декодера. Модель Top обучена генерировать коды верхнего уровня для VQ-VAE-2 на основании метки класса, модель Bottom создаёт коды нижнего уровня на основании верхних, сгенерированных моделью Top. Подробности устройства моделей приведены в таблице 1. Использование такой сравнительно небольшой архитектуры продиктовано необходимостью разместить обе модели на графическом ускорителе с памятью 16 Гб, и итоговое количество слоёв было найдено эмпирически в ходе анализа работы графического ускорителя с моделями разных размеров.

Модель	Размерность входных данных	Размерность выходных данных	Размерность эмбеддингов	Количество голов внимания	Нейронов в полносвязных слоях
Top	1	901	100	4	1024
Bottom	901	3601	100	4	1024

Таблица 1 – Параметры трансформерных моделей Top и Bottom. Модели устроены идентично, за исключением объёма данных, с которыми они должны работать.

2.4 Обработка входных данных

Алгоритм Im2Wav для генерации аудио из видео пропускает входную запись через трансформерную модель CLIP, чтобы получить богатое описание происходящего на видео.

В контексте данной работы применение такой модели было бы избыточно, поэтому в финальном алгоритме для обработки данных используется предобученная свёрточная нейронная сеть ResNet-50, которая

была дообучена, чтобы различать классы входных данных. Нейросеть обрабатывает изображения. В случае работы с видео, оно предварительно представляется в виде последовательности кадров, каждый из которых обрабатывается сетью и получает метку класса. Медианная метка со всех кадров передаётся в трансформерную модель Top.

Модель присваивает изображению метку «0», если на нём находится кот, и метку «1», если на нём изображена собака.

Глава 3. Результаты

Для оценки работы моделей глубокого обучения используются общепринятые метрики, применение которых к конкретным моделям зависит от специфики их работы. Поскольку данная работа фокусируется на построении генеративного алгоритма с использованием ограниченных ресурсов, а не на качестве, высоких значений метрик не ожидается. Но оценка моделей – важная часть их разработки, и в данном случае она проводится в демонстративных целях. При подсчётах модели оперировали на валидационных данных, которых не было в обучающей выборке, то есть, полученные значения качества справедливы для новой для модели информации.

Accuracy («точность») – одна из простейших метрик машинного обучения, которая может быть использована для определения качества классификатора. Она определяется как отношение количества верных предсказаний модели к общему количеству совершённых предсказаний, то есть, является долей правильных ответов:

$$Accuracy = \frac{1}{N}\sum_{i}^{N} 1(y_i = \hat{y_i}), \tag{15}$$

где N – размер валидационной выборки, y_i – корректная метка класса, $\hat{y_i}$ – результат работы нейронной сети. В этой работе оценивается точность работы модели ResNet.

Оценка работы автокодировщика будет проводиться с помощью звукового расстояния Фреше (Frechet Audio Distance) [35]. Это метрика схожести между распределениями звуковых файлов из датасета и их реконструкций, полученных моделью. Формула для её подсчёта:

$$F(X,Y) = \|\mu_X - \mu_Y\|_2^2 + \mathrm{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X\Sigma_Y}\right), \tag{16}$$

где μ_X и μ_Y – математические ожидания для двух распределений, Σ_X и Σ_Y – ковариационные матрицы, Tr – след матрицы.

Самые сложные модели для оценки – трансформерные. Сложно математически формализовать качество генерации, так что многие популярные метрики – сравнительные. Я буду использовать перплексию (Perplexity), которая вычисляется как экспонента от кросс-энтропии и отображает степень хаотичности генерации – чем больше перплексия, тем меньше нейронная сеть «уверена» в результате генерации, а значит, тем хаотичнее генерация. Подсчёты проводились для результатов, сгенерированных top-k сэмплированием с k=6 для обоих моделей. Судя по вычисленной перплексии, модель Top обучилась лучше модели Bottom, её генерации более связны и осмысленны.

Модель	Количество параметров	Шагов обучения	Время обучения	Accuracy	FAD	Perplexity
VQ-VAE-2	580 641	10 920	3,4 часа	–	32883	–
Top	5 341 084	10 080	10,3 часов	–	–	6.217
Bottom	5 341 084	10 080		–	–	13.3276
ResNet-50 (пред-обученная)	23 512 130	До 60×10^4	Не приведено	0.51	–	–
ResNet-50 (дообучение)	23 512 130	160	2.06 минут	0.98	–	–

Таблица 2 – Метрики машинного обучения для обученных моделей. Модели Top и Bottom тренировались одновременно, приведено общее время обучения двух нейросетей. Информация о количестве шагов обучения ResNet-50 взята из оригинальной статьи [25], о времени обучения авторы не сообщают. Без дообучения ResNet-50 имеет Accuracy на уровне случайного угадывания, но спустя две минуты обучения полносвязных слоёв работает почти безупречно.

В отличие от дискриминативных моделей, которые не создают новой информации, и главная метрика которых вполне может быть точностью, как у ResNet, для генеративных очень важна субъективная оценка качества генерации. Например, оценивая «на слух» результаты работы VQ-VAE-2, можно заметить, что она лучше восстанавливает резкие и короткие звуки – такие, как лай собак, чем длинные и протяжные – например, мяуканье. Такой результат может быть связан с разными факторами – например, необходимостью сжимать входные данные или недостаточной размерностью пространства эмбеддингов модели, причина появления которых – ограниченность ресурсов. Однако восстанавливаемые звуки всё равно обладают хорошо различимыми признаками оригиналов – это было продемонстрировано на рисунке 13, и в их звучании легко опознать как лай, так и мяуканье.

Самая сложная часть алгоритма для обучения – трансформерные модели, которые требуют гораздо больших ресурсов: как показано в таблице 2, при меньшем количестве признаков у моделей в сравнении с автоэнкодером, их обучение заняло в три раза больше времени. При этом, на данном этапе работы трансформеры плохо справляются с генерацией: на слух её результаты чаще похожи на случайный шум, чем осмысленные звуки. Однако по косвенным признакам – уменьшению лосс-функции при тренировке и сравнительно небольшим значениям перплексии, можно с уверенностью сказать, что данные модели смогли в какой-то степени выучить распределения кодов VQ-VAE-2. Достижение устойчивой генерации потребует большего времени обучения и в идеальном варианте – масштабирования моделей, но теоретическая возможность их встраивания в алгоритм была как объяснена практически, так и доказана эмпирически.

Статья об алгоритме Im2Wav [7] – пример, демонстрирующий возможность обучения схожих автокодировщиков и трансформерных моделей

для генерации качественных звуков. В моих трансформерных моделях 18 слоёв, в моделях Im2Wav – 48 [7, с.3], однако в обоих случаях модели обучились до качества определённого уровня. Масштабирование и обучение моих нейронных сетей на более качественных данных может стать темой для будущих исследований.

Заключение

Задача данной работы состояла в исследовании возможности создания алгоритма для обусловленной генерации аудио на основании входного изображения или видео в условиях ограниченных вычислительных ресурсов и последующем построении алгоритма, адаптированного для работы в описанных условиях. На основании проделанной работы, сделаны следующие выводы:

1. Исследование, проведённое выше, доказало практическую возможность создания описанного алгоритма, все изученные модели глубокого обучения теоретически возможно обучить и запустить, используя небольшие вычислительные мощности.
2. Для работы в условиях ограниченных мощностей входные данные часто необходимо преобразовывать, уменьшая их качество – следовательно, и качество выходных данных полного алгоритма, что не противоречит задаче данной работы, но является важной практической деталью.
3. Описанный выше алгоритм был разработан. Для него были спроектированы, обучены или дообучены четыре независимые модели глубокого обучения, которые затем были оценены как субъективно, так и с использованием релевантных метрик.

Таким образом, я считаю задачу данной работы, выполненной в полном объёме. В области обусловленной генерации аудио остаётся большое пространство для исследований – будущие работы могут фокусироваться на увеличении качества представленного мной алгоритма, например, за счёт масштабирования трансформерных моделей, ответственных за генерацию, и обучения на больших датасетах, таких, как VGG-Sound.

Список использованных источников

1. Goodfellow I. et al. Generative adversarial nets //Advances in neural information processing systems. – 2014. – Т. 27.
2. Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. – 2017. – Т. 30.
3. Ramesh A. et al. Zero-shot text-to-image generation //International conference on machine learning. – Pmlr, 2021. – С. 8821-8831.
4. Esser P., Rombach R., Ommer B. Taming transformers for high-resolution image synthesis //Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. – 2021. – С. 12873-12883.
5. Ghorpade J. et al. GPGPU processing in CUDA architecture //arXiv preprint arXiv:1202.4347. – 2012.
6. Schuhmann C. et al. Laion-5b: An open large-scale dataset for training next generation image-text models //Advances in Neural Information Processing Systems. – 2022. – Т. 35. – С. 25278-25294.
7. Sheffer R., Adi Y. I hear your true colors: Image guided audio generation //ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – IEEE, 2023. – С. 1-5.
8. Betker J. et al. Improving image generation with better captions //Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf. – 2023. – Т. 2. – №. 3. – С.
9. Kreuk F. et al. Audiogen: Textually guided audio generation //arXiv preprint arXiv:2209.15352. – 2022.
10. Borsos Z. et al. Audiolm: a language modeling approach to audio generation //IEEE/ACM Transactions on Audio, Speech, and Language Processing. – 2023.
11. Van Den Oord A. et al. Neural discrete representation learning //Advances in neural information processing systems. – 2017. – Т. 30.

12. Liu H. et al. Audioldm 2: Learning holistic audio generation with self-supervised pretraining //IEEE/ACM Transactions on Audio, Speech, and Language Processing. – 2024.
13. Radford A. et al. Language models are unsupervised multitask learners //OpenAI blog. – 2019. – Т. 1. – №. 8. – С. 9.
14. Evans Z. et al. Fast Timing-Conditioned Latent Audio Diffusion //arXiv preprint arXiv:2402.04825. – 2024.
15. Rombach R. et al. High-resolution image synthesis with latent diffusion models //Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. – 2022. – С. 10684-10695.
16. Dalal M., Li A. C., Taori R. Autoregressive models: What are they good for? //arXiv preprint arXiv:1910.07737. – 2019.
17. Ziv A. et al. Masked Audio Generation using a Single Non-Autoregressive Transformer //arXiv preprint arXiv:2401.04577. – 2024.
18. McGrady R. et al. Dialing for Videos: A Random Sample of YouTube //Journal of Quantitative Description: Digital Media. – 2023. – Т. 3.
19. Razavi A., Van den Oord A., Vinyals O. Generating diverse high-fidelity images with vq-vae-2 //Advances in neural information processing systems. – 2019. – Т. 32.
20. Розенблатт Ф., Алтаев В. Я., Власюк Б. А. Принципы нейродинамики: Перцептроны и теория механизмов мозга: Пер. с англ. – Мир, 1965.
21. Krose B., Smagt P. An introduction to neural networks. – The University of Amsterdam, 1996.
22. Amari S. Backpropagation and stochastic gradient descent method //Neurocomputing. – 1993. – Т. 5. – №. 4-5. – С. 185-196.
23. O'shea K., Nash R. An introduction to convolutional neural networks //arXiv preprint arXiv:1511.08458. – 2015.
24. LeCun Y. et al. Gradient-based learning applied to document recognition //Proceedings of the IEEE. – 1998. – Т. 86. – №. 11. – С. 2278-2324.

25. He K. et al. Deep residual learning for image recognition //Proceedings of the IEEE conference on computer vision and pattern recognition. – 2016. – C. 770-778.
26. Dumoulin V., Visin F. A guide to convolution arithmetic for deep learning //arXiv preprint arXiv:1603.07285. – 2016.
27. Bank D., Koenigstein N., Giryes R. Autoencoders //Machine learning for data science handbook: data mining and knowledge discovery handbook. – 2023. – C. 353-374.
28. Kantarcı A., Ekenel H. K. Thermal to visible face recognition using deep autoencoders //2019 International conference of the biometrics special interest group (BIOSIG). – IEEE, 2019. – C. 1-5.
29. Huang G. B. et al. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments //Workshop on faces in'Real-Life'Images: detection, alignment, and recognition. – 2008.
30. Kingma D. P., Welling M. Auto-encoding variational bayes //arXiv preprint arXiv:1312.6114. – 2013.
31. Razavi A., Van den Oord A., Vinyals O. Generating diverse high-fidelity images with vq-vae-2 //Advances in neural information processing systems. – 2019. – T. 32.
32. Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/
33. Chen H. et al. Vggsound: A large-scale audio-visual dataset //ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – IEEE, 2020. – C. 721-725.
34. Takahashi N. et al. Deep convolutional neural networks and data augmentation for acoustic event detection //arXiv preprint arXiv:1604.07160. – 2016.
35. Kilgour K. et al. Fr\'echet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms //arXiv preprint arXiv:1812.08466. – 2018.