Санкт-Петербургский Государственный Университет
Прикладной математики - Процессов Управления

САЛИХОВ Марат Ахатович

Выпускная квалификационная работа

Использование искусственных нейронных сетей в задачах генерации музыки

Уровень образования: бакалавриат
Направление: 02.03.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2018 «Программирование и информационные технологии»

Научный руководитель:
доцент кафедры
теории систем управления
электрофизической аппаратурой,
к. ф.-м. н.
Козынченко Владимир Александрович

Рецензент:
доцент кафедры
технологии программирования,
к.т.н.
Блеканов Иван Станиславович

Санкт-Петербург
2022

Содержание

Введение 4
Постановка задачи 8
Обзор литературы 9
Глава 1. Нейронные сети 11
1.1 LSTM 11
1.2 Генеративно-состязательные сети 13
Глава 2. Данные и их предобработка 16
2.1 Виды музыкальных данных 16
2.2 Выбранная база данных 17
2.3 Музыкальная теория 18
2.4 Предобработка данных 20
Глава 3. Разработка моделей 22
3.1 Генерация ноты 22
3.2 LSTM сеть 24
3.2.1 Данные и параметры 25
3.2.2 Архитектура сети 25
3.2.3 Описание алгоритма генерации 27
3.2.4 Обучение и анализ модели 28
3.3 Генеративно-состязательная сеть 32
3.3.1 Данные и параметры 32
3.3.2 Архитектура сети 33
3.3.3 Описание алгоритма обучения GAN 37
3.3.4 Обучение и анализ модели 38
3.4 Результаты 40

3.5 Выводы 42
Заключение 43
Список использованной литературы 44
Приложение А. Архитектура LSTM сети 46
Приложение Б. Архитектура генеративно-состязательной сети 47

Введение

Генерация музыки является одним из самых абстрактных видов творческой деятельности. В отличии от других типов произведений, музыку способен понять каждый человек, независимо от познаний об окружающем мире. Естественно восприятие мира отдельного человека несомненно влияет на то, как он понимает то или иное произведение. Кроме того, музыка может быть представлена в виде простого формата данных без большой потери основного содержания. Так издревле были придуманы ноты, с помощью которых удобно создавать и воспроизводить музыку. Несмотря на то, что музыку можно представить в виде некоторой модели, есть множество вопросов, на которые не были получены окончательные ответы. Например, “Почему некоторые последовательности нот воспринимаются как музыка, а другие нет”. В музыке можно выделить некоторые закономерности, интуитивно понятные людям, но тяжело формулируемые на математическом языке.

Выявлению и определению основных концепций написания музыки посвящены целые дисциплины такие, например, как теория музыки. Но что же нужно уметь машине, чтобы уметь сочинять музыку, похожую на ту, что пишет человек?

Чтобы сочинить произведение, воспринимаемое человеческим слухом как музыка, компьютер должен уметь выявлять и воссоздавать последовательную музыкальную структуру. Модель должна уметь улавливать ритм музыки, следить за гармонией, а также различать стили, в которых написаны мелодии. Помимо этого, современная музыка часто полифонична, то есть в музыке участвуют сразу несколько потоков нот, написанных для разных инструментов, которые могут объединяться и создавать диссонантные или же консонантные гармонии, объединяясь в некоторый оркестр.

Данная задача является комплексной, а правильно оценить полученные результаты ввиду специфики предметной области проблематично.

Тем не менее, задача автоматической генерации музыки привлекает множество исследователей. Так, с появлением методов машинного обучения, основанных на нейронных сетях, данная тема стала еще актуальней.

Методы глубокого обучения начали применяться к музыке только недавно, и качество результатов все еще сильно отстает от результатов в других областях применения нейронных сетей.

Такое отставание в создании музыки с помощью глубокого обучения может быть связано с многими факторами. Например, одним из таких факторов является то, что музыка в своем понимании – очень широкое понятие, граница между музыкальным творчеством и простым набором звуков очень тонка. Но все же она существует и поэтому для решения этой задачи активно применяются методы машинного обучения, известные своей способностью улавливать зависимости, которые не заметны большинству людей.

Основным отличием данной работы является сведение задачи генерации музыки к прогнозированию временных рядов. Также в данной работе будет представлена реализация архитектуры GAN, ставшей state-of-the-art решением в задачах генеративного обучения [1].

В первой главе представлены общие теоретические сведения по нейронным сетям, рассмотрены основные архитектуры нейронных сетей, которые будут использованы в последующих моделях.

Вторая глава посвящена подготовке данных: выбору базы данных, предобработке данных, а также существующим способам представления музыки. Также в этой главе приведены небольшие понятия касающиеся теории музыки.

В третьей главе представлены разработанные модели, принципы их выбора, ход исследования, а также сравнение полученных архитектур. Помимо этого, приводится анализ полученных решений.

В заключении подведены итоги проведенной работы.

Актуальность работы

Генерация музыки довольно интересная, полезная и актуальная задача. Так как музыка используется повсеместно, создание модели, способной генерировать музыку, будет востребовано как в коммерческих целях, так и в исследовательских.

Так, например, в игровой индустрии многим небольшим компаниям часто не хватает музыки, необходимой на этапах разработки проекта. Им приходится искать либо существующие мелодии в свободном доступе, либо обучаться созданию музыки с нуля. Так как для таких компаний нанять талантливого композитора будет дороже, чем купить лицензию программного обеспечения по генерации музыки, то они скорее всего, прибегнут ко второму варианту.

Помимо этого, разработка систем автоматической генерации музыки может стать интересной перспективой для изучения самого процесса сочинения мелодии человеком, также такие генераторы послужат некоторым способом оценки музыкальных произведений. При использовании генеративно-состязательных сетей помимо обученного генератора, разработчик получает и хорошо обученный дискриминатор, который можно использовать в различных целях, таких, например, как классификация жанра музыки, что является актуальной проблемой ввиду возрастающего числа жанров музыки и размывания границ между ними.

Более широкой перспективой является то что система генерации музыки способна помочь и людям-музыкантам на различных этапах создания музыки: сочинении, аранжировки, оркестровки, постановки и т. д. При сочинении

мелодии, музыкант редко создает новую музыку с нуля. Обычно процесс написания музыки включает в себя процесс повторного воспроизведения или адаптации, сознательно или бессознательно, черт из различной музыки, которую автор уже знает или слышал, следуя при этом некоторым принципам и рекомендациям, таким как теория гармонии. Система генерации музыки может помогать музыканту на разных этапах композиции, инициировать, предлагать, дополнять вдохновение композитора.

Помимо этого, системы генерации музыки, основанные на искусственных нейронных сетях, обладают способностью запоминать и обобщать данные на которых они учатся, благодаря чему, можно создавать произведения в классическом стиле от знаменитых авторов, например, Баха.

Постановка задачи

Целью выпускной квалификационной работы является разработка модели автоматической генерации монофонической музыки с использованием искусственных нейронных сетей. Для достижения поставленной цели, необходимо решить следующие задачи:

1. Выбрать данные для обучения и выполнить их предобработку
2. Реализовать модель, основанную на LSTM
3. Реализовать модель, основанную на генеративно-состязательных сетях
4. Обучить модели и проанализировать полученные результаты

Обзор литературы

В последнее время для создания музыки было предложено большое количество моделей глубоких нейронных сетей. В большинстве работ в области генерации музыки используются MIDI-файлы. Но также используются и WAV файлы, хранящие оцифрованный аудиопоток.

Одной из наиболее известных работ в области генерации музыки является сеть DeepBach, специально разработанная для составления полифонической четырехчастотной хоральной музыки в стиле Иоганна С. Баха [2]. Эта модель проектировалась на основе RNN, с возможностью пользователю накладывать ограничения на такие параметры как, например, ритм и аккорды.

Помимо решения задач по проектированию систем способных генерировать музыку определенного типа и стиля существуют модели по генерации мелодии, имеющей в себе некоторую новизну [3]. Для достижения этого использовались две раздельно обучаемые рекуррентные нейронные сети и некоторый параметр, отвечающий за степень уникальности сгенерированной музыки. Первая из сетей предсказывала длительность ноты, а вторая по предсказанной длительности и по тональностям предыдущих нот высчитывала распределение вероятностей звучания следующего символа.

Существуют и решения, основанные на принципах работы нейронных сверточных сетей. Так, задача генерации музыки была разделена на набор связанных параллельных нейронных сетей [4]. Каждая такая сеть отвечает за одну ноту и имеет связанные веса с каждой другой сетью.

Применяется и архитектура генеративно-состязательных сетей (GAN). Работы основанные на этой модели используют в качестве генераторов RNN, а также сверточные нейронные сети в качестве классификаторов [5; 6]. Иногда подобные решения модифицируют использованием таких методов как Seq2Seq [5; 7].

Интересное и многообещающие решение было получено при рассмотрении ноты, ее высоты, длительности и расположении как некоторое уникальное слово [8]. Для этого были использованы Скрытые Марковские модели (HMM), а также LSTM [9; 10].

В книге [11] представлена исчерпывающая информация о генеративно-состязательных сетях, модификации применимые к данному типу нейронных сетей. Рассмотрены области применения GAN, а также описаны недостатки архитектуры и проблемы, которые могут возникнуть при обучении сети.

Глава 1. Нейронные сети

1.1 LSTM

LSTM (англ. *Long short-term memory*) – сеть с долгой краткосрочной памятью. Представляет собой модификацию рекуррентных нейронных сетей (RNN) и является мощным и популярным инструментом в задачах прогнозирования временных рядов. Также LSTM отлично себя показала в задачах, связанных с обработкой последовательности символов, таких как машинный перевод или предсказание следующего слова, на основе предыдущих.

Основным преимуществом LSTM сети является то, что она за счет своей структуры, способна сохранять значимую информацию в течение долгих промежутков времени. Так, в задачах машинного перевода, при последовательном переводе слов, важно помнить значительную часть предложения, с чем и справляется эта сеть.

В слое LTSM всего одна ячейка, содержащая множество узлов, число которых задается гиперпараметром.

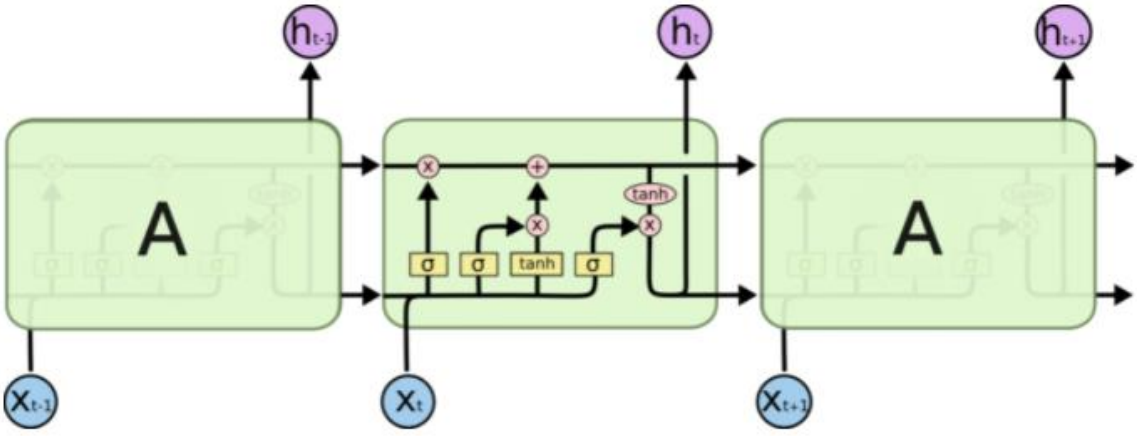

Рис.1: Схема LSTM ячейки [12]

LSTM обрабатывает последовательные входные данные $[x_1, x_2, \dots, x_n]$. На каждом временном шаге t ячейка обновляет свое состояние c_t и скрытое состояние h_t, основываясь на входных данных текущего временного шага x_t, и предыдущих состояний c_{t-1} и h_{t-1}. Этот процесс продолжается до конца входной последовательности. В качестве результата работы LSTM сети, ячейка возвращает свое скрытое состояние h_t размерности равной количеству узлов.

На Рис.1 изображена схема LSTM ячейки. Рассмотрим процесс обновления скрытого состояния ячейки. В LSTM 4 полносвязных слоя нейронов, 3 с сигмоидальной функцией активации и 1 с функций гиперболического тангенса.

Первый этап работы ячейки это обработка объединённого вектора, состоящего из вектора входных данных x_i и вектора скрытого состояния предыдущего временного шага h_{t-1} с помощью фильтра забывания. В ходе первого этапа, вычисляется та доля информации, которая должна сохранится в состоянии c_t.

Следующий этап характерен добавлением новой релевантной информации к вектору состояния ячейки c_t. Объединённый вектор $[h_{t-1}, x_i]$, участвующий в предыдущем этапе, проходит через два полносвязных слоя, их результаты перемножаются, после чего полученный вектор прибавляется к c_t.

Когда преобразования вектора состояния ячейки c_t завершены, обновляется вектор скрытого состояния h_t. Это происходит за счет умножения некоторой доли c_t, полученной с помощью гиперболического тангенса, и результата прохождения вектора $[h_{t-1}, x_i]$ через полносвязный слой с сигмоидальной функций активации.

Обновленный вектор скрытого состояния подается как краткосрочное состояние сети в следующий момент времени. Также LSTM ячейка способна возвращать свое скрытое состояние в каждый момент времени. Эта возможность регулируется специальным гиперпараметром.

Таким образом LSTM сеть запоминает важную информацию и забывает устаревшую, контекстно неверную информацию.

1.2 Генеративно-состязательные сети

Генеративно-состязательная сеть (GAN) – методология построения нейронных сетей, в основе которой стоит идея реализации архитектуры, состоящей из комбинации двух “соперничающих” нейронных сетей. Впервые такие сети были предложены Яном Гудфеллоу и его коллегами в 2014 году [1].

Архитектура генеративно-состязательных сетей строится из Генератора (G) и Дискриминатора (D). Дискриминатор получает на вход два типа данных: заранее подготовленные данные, полученные из предметной области, формирующие обучающую выборку и результат работы генератора. Задача дискриминатора D определить, является ли поступивший к нему на вход образец созданной генератором подделкой или же частью изначального набора данных основанного на реальных данных. Генератор G же старается произвести такие образцы, которые дискриминатор не смог бы отличить от реальных. Схема генеративно-состязательной сети представлена на Рис.2.

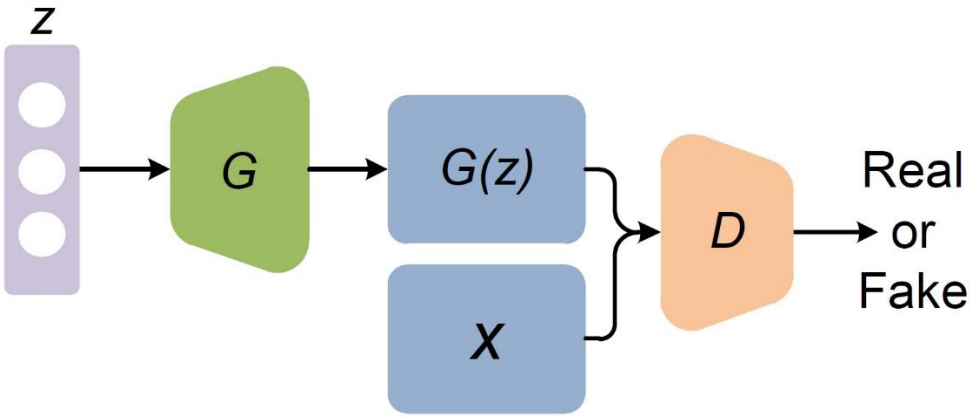

Рис.2: Генеративно-состязательная сеть

Тем самым можно выделить уже несколько преимуществ такого алгоритма машинного обучения:

- На каждой итерации обучения генеративно-состязательных сетей последовательно обновляются веса генератора и дискриминатора, что способствует их параллельному обучению. В зависимости от поставленной задачи, можно выбрать архитектуры сетей так, чтобы получить более сильный дискриминатор или же генератор
- Генератор должен создавать реалистичные данные из случайного шума. Что позволяет после обучения сети, генерировать множество самых различных образцов

Пусть p_{data} – есть некоторая выборка из множества реальных данных, $p_g(x)$- распределение данных, создаваемых генератором, $D(x, \theta_d) \in [0,1]$ – дискриминатор, значение функции которого отражает вероятность того, что x не подделка, θ_d – параметры дискриминатора. Тогда, во время обучения генеративно-состязательной сети дискриминатор максимизирует следующую функцию:

$$f_1 = max_D \, E_{x \sim p_{data}} [\log \mathrm{D(x)}] + E_{x \sim p_g} [\log(1 - D(x))]$$

В то же время, задача генератора, чтобы дискриминатор как можно хуже отличал ложные данные от настоящих. Пусть $p_z(z)$ – распределение шума, θ_g – параметры генератора, z – случайный шум, тогда функция $G(z, \theta_g)$ отображает шум в пространство образцов $p_g(x)$. Тогда генератор минимизирует следующую функцию:

$$f_2 = min_G \, E_{z \sim p_z(z)} \log\left(1 - D\left(G(z)\right)\right)$$

Вышеописанные задачи можно переписать в виде минимакс игры:

$$min_G max_D V(D, G) = E_{x \sim p_{data}(x)} [\log \mathrm{D(x)}] + E_{z \sim p_z(z)} [\log\left(1 - D\left(G(z)\right)\right)]$$

Несмотря на преимущества генеративно-состязательных сетей, у них есть и свои недостатки:

- Неинформативные потери. Ввиду того, что потери вычисляются по постоянно совершенствующемуся дискриминатору, тяжело сравнить потери полученные в разные моменты времени
- Генератор и дискриминатор должны обучаться в некоторой “гармонии”. Если дискриминатор окажется слишком слабым, то генератор будет генерировать один и тот же, немного отличающийся образец, обманывающий генератор. Такая ситуация называется коллапсом модели. Наоборот, слишком сильный дискриминатор не позволит генератору получать полезную информацию от него, что по итогу ведет к бесконечному росту потерь дискриминатора.
- Количество гиперпараметров в генеративно-состязательных сетях значительно превышает количество параметров в других архитектурах.

Глава 2. Данные и их предобработка

Для того, чтобы искусственные нейронные сети смогли генерировать мелодии, сначала необходимо определиться с тем на каких данных будут обучаться эти сети и с каким типом данных они будут работать.

В этой главе будут рассмотрены способы представления музыки, выбран набор данных, на котором будут обучаться модели, а также приведен алгоритм предобработки данных.

2.1 Виды музыкальных данных

Выбор того или иного представления данных может сильно повлиять как на сам процесс генерации музыки, так и на то, какие архитектуры сетей следует использовать.

На данный момент в задачах, связанных с машинным обучением существуют два основных способа представления музыкальных данных: **символьный** и **аудио формат**.

Основным различием между этими видами представления мелодии можно выделить то, что аудио формат, или другими словами аудио сигнал, является непрерывной величиной, обычно записанной с помощью специальных устройств. Символьный же формат является дискретным набором условных знаков, содержащих в себе информацию о длительности и высоте звучания того или иного звука. Аудио формат представлен на Рис. 3.

Рис.3: Пример аудио формата данных представленного в виде формы волны [13]

У аудио формата данных есть и другие представления, помимо формы волны, получаемые с помощью быстрого преобразования Фурье, но в данной работе эти представления рассматриваться не будут.

Несмотря на то, что при использовании аудио формата данных открываются большие возможности в виде воспроизведения любого типа звука, а вследствие и способности генерировать как полифоническую, так и монофоническую музыку, в данной работе выбран символьный формат данных, что является хоть и простым, но более распространенным типом данных в задачах автоматической генерации музыки.

У символьного формата данных существует множество различных представлений, в качестве основы взят устоявшийся способ записи музыки – **музыкальная нотация.** Пример музыкальной нотации представлен на Рис. 4

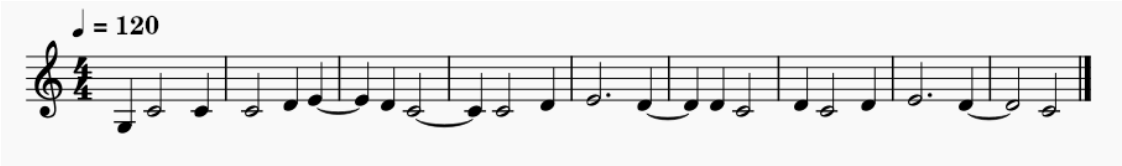

Рис.4: Пример музыкальной нотации

2.2 Выбранная база данных

При выборе набора данных для решения задачи генерации музыки, очень важно, чтобы полученные данные хранили как можно больше нужной информации, необходимой для эффективного обучения нейронных сетей. Для достижения такого результата, необходимо большое количество данных, а также важно, чтобы эти данные сильно не отличались друг от друга.

В качестве набора данных, на котором будут обучаться нейронные сети, была выбрана известная база данных KernScores. В ней хранится большое количество фольклорных мелодий различных стран мира. Большая часть из них принадлежит народной музыке Германоязычных народов. Также музыка,

представленная в этой базе данных, делится на различные классы, обозначающие жанр, что способствует выполнению условия о том, чтобы данные имели общую природу.

KernScores хранит музыкальные файлы закодированные в специальный формат **kern. Этот тип кодирования сохраняет важную информацию о нотах, используемых в музыкальной нотации, их длительности. Данную кодировку легко расшифровать, для последующей работы в любом современном интерфейсе, например, в MIDI- интерфейсе.

В качестве данных, на которых будут обучаться нейронные сети выбран подкласс фольклорных мелодий Германоязычных народов “ballad”, насчитывающий 687 произведений. Количество получаемых образцов данных, извлекаемых из этих мелодий сильно варьируется от используемых параметров обучения сети. Параметры, используемые во время обучения нейронных сетей, будут представлены в Главе 3.

2.3 Музыкальная теория

Для лучшего понимания того, как правильно приступить к работе с данными, в частности к их предобработке, важно изучить то как устроена музыка.

Будем рассматривать мелодию как последовательность *нот* и *пауз*.

Так в основе музыкальной системы лежит ряд, состоящий из 88 нот или же звуков, расположенных по возрастанию *высоты* звука. В данной работе, будут использоваться англоязычные названия нот, то есть C, D, E и так далее.

В музыке расстояние между соседними нотами с одинаковыми названиями называется *октавой*. Основным различим между такими нотами, как описывалось выше, является высота звука.

Помимо высоты звука, нота несет еще одну важную информацию, такую как длительность звучания или же просто *длительность звука*. На Рис.5 представлена иерархия длительности звука, выраженная деревом.

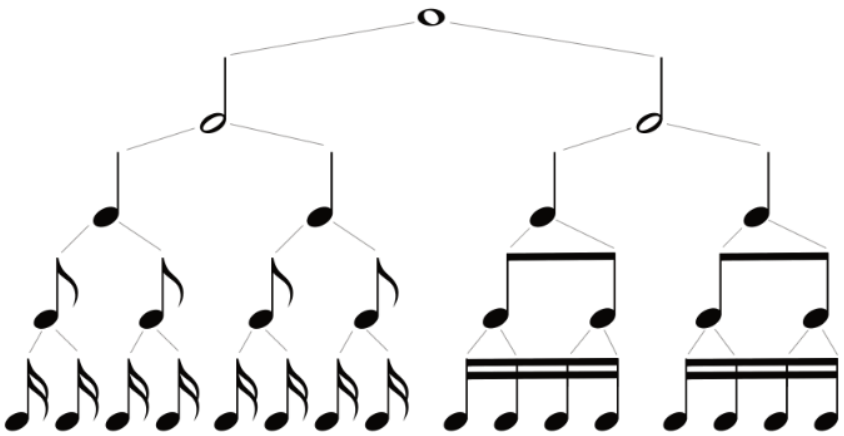

Рис.5: Иерархия длительности нот [15]

Корнем данного дерева является максимальная по длительности звучания нота – *целая нота*. В листах же находятся шестнадцатые ноты, то есть ноты чья длительность в 16 раз меньше чем у целой ноты.

Под паузой будем воспринимать ноту определенной длительности, не имеющий какого-либо звукового сопровождения ей соответствующего.

Еще одним важным понятием, необходимым для дальнейшей работы является *такт*, а также *размер произведения*. Чтобы не углубляться в тонкости определений, рассмотрим пример представленный на Рис.6

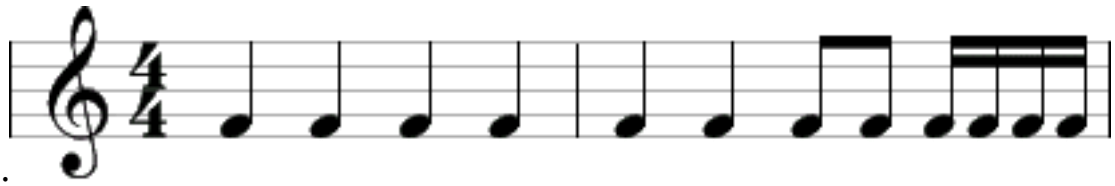

Рис.6: Пример длительности нот [15]

На нем изображена музыкальная нотация с размером четыре четверти. Это означает то, что в каждом такте, то есть небольшом интервале разделенным вертикальной прямой, суммарная длительность всех нот и пауз должна равняться длительности четырех четвертей.

Ключ – это элемент нотного письма, который указывает размещение одной из нот, относительно которой размещаются все остальные ноты.

Под записью C мажор, будем воспринимать некоторую последовательность звуков в определенном интервале относительно *тоники* C, где *тоника* C, является некоторым центром гравитации, к которому стремятся все ноты. Также тоника создает чувство завершенности той или иной части музыкальной нотации.

Помимо этого, получены представления о базовых правилах составления музыки. Но учитывать всех их не всегда приходится, или же в некоторых моментах они вовсе игнорируются. Именно поэтому для генерации музыки, был выбран набор данных KernScores. Следуя эвристике того, что искусственные нейронные сети выделяют основные структурные правила и запоминают их, можно утверждать, что при обучении такой сети на наборе данных из фольклорных песен различных стран мира, определенного жанра, можно добиться того, что сеть сможет выделить и запомнить определенные правила написания мелодий.

2.4 Предобработка данных

После выбора базы данных и исследования предметной области, необходимо преобразовать музыкальные данные, чтобы нейронные сети смогли обучаться на них.

Используя данные о предметной области были приняты следующие допущения к исходным данным. Используя только C мажор и A минор, мы уменьшаем потребность модели в огромном наборе данных, а также упрощаем саму модель, не ставя задачу разбираться в различных видах ключа (например, D, E).

Если же понадобится обучать для всех 24 видов ключей, то придется огромный набор данных изменять и модифицировать под все 24 возможных ключа.

Все это приведет к неоправданно высокому мультипликативному росту исходных данных, а вследствие и длительности обучения, ограниченным вычислительными способностями компьютера на котором запускается программа.

Помимо этого, в результате личного исследования и опроса экспертов предметной области было заключено что порядка 70-80% текстов народного искусства и не только написано на A миноре, либо же на C мажора, что обуславливает возможность ограничиться только ими для упрощения работы модели с возможность взамен усложнить ее там, где это необходимо.

Для того чтобы обработать данные и применить к ним вышеописанные преобразования использовалась библиотека Music21 языка Python. Эта библиотека позволяет конвертировать файлы MIDI, kern в специальный тип Score пакета Music21. После чего работа происходит уже с классом Score.

После применения преобразований, необходимо определиться с тем как данные будут храниться в системе. Для этого было предпринято решение представить музыкальные данные в виде временного ряда. Из теории музыки известно, что минимальная длительность ноты равна шестнадцатой ноте. Следуя этому суждению, можно предположить, что монофоническая музыка есть последовательность значений тональностей, полученных на каждом временном шаге равном шестнадцатой ноте.

Такой подход имеет под собой обоснование, в музыке мы часто видим некоторые тенденции роста высоты звуков или повторения некоторых куплетов, что сильно коррелирует с понятиями теории временных рядов. Тогда задачу генерации музыки можно ставить как задачу прогнозирования временных рядов, с которыми достаточно хорошо справляется LSTM. Кроме этого такое представление музыкальных данных решает одну из проблем автоматических систем генерации музыки, когда сеть может прогнозировать лишь известные ей ноты. Подробнее этот вопрос рассмотрен в Главе 3.

Глава 3. Разработка моделей

В данной главе будут представлены архитектуры моделей автоматического генерации музыки, процесс их обучения, анализ архитектуры сетей и полученных результатов.

Описанные в этой главе сети реализованы и обучены с помощью библиотеки Keras, надстройки над библиотекой искусственных нейронных сетей Tensorflow языка Python [16].

Обучение происходило на личном компьютере с использование NVIDIA CUDA toolkit, позволяющим выполнять вычисления и обучения нейронных сетей на графическом процессоре GPU. Архитектура графического ускорителя: NVIDIA GTX 1050.

3.1 Генерация ноты

В данной работе используется представление музыки в виде временного ряда. После всех преобразований, описанных в Главе 2, каждой встреченной ноте уникальной высоты(тональности) сопоставляется новое значение в словаре. Тем самым, обработав каждую мелодию в выбранном наборе данных, мы получим словарь, содержащий все уникальные ноты. После этого, все данные перекодируются в последовательность целых чисел в соответствие с этим словарем. Чтобы кодировать длительность той или иной ноты, вводится специальный знак, обозначающий то, что нота, предшествующая этому знаку, все еще проигрывается. То есть запись вида:

$$60 - - 61 - -$$

Обозначает то, что нота, соответствующая целочисленному значению 60 имеет длительность в 3 шестнадцатых ноты. В условиях задачи данной работы минимальная продолжительность ноты взяла как шестнадцатая нота. Такой подход к кодированию данных имеет множество преимуществ, таких как:

- Возможность учитывать не только тональность ноты, но и её продолжительность
- Способность генерировать ноты произвольной длины, что способствует созданию нот, отсутствующих в обучающем наборе
- Уменьшение размерности словаря за счет возможности описать любую ноту с помощью комбинации специального символа и целочисленного значения

Тогда задачу генерации музыки можно свести к вычислению закона распределения вероятностей дискретной случайной величины, обозначающей звучание определенной ноты. Пусть задана совокупность $a_i \in A$, где A – есть значения соответствующие уникальным нотам в словаре. Тогда задачей нейронной сети становится поиск вероятностей: $p(a_i) = p_i$, таких что $\sum_{i=1}^{N} p_i = 1$, где N – размер словаря.

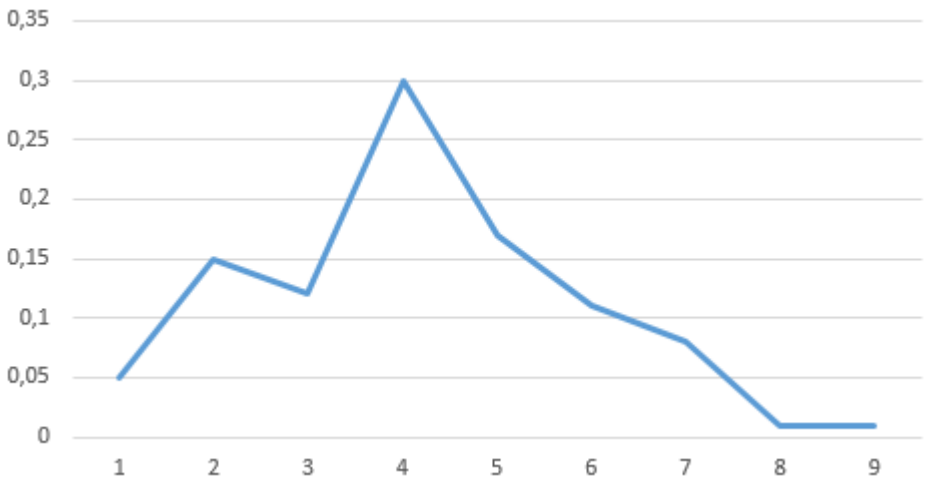

Рис.7 Многоугольник распределения

На Рис.7 представлен многоугольник распределения, являющийся графическим способом представления закона распределения дискретной случайной величины. После вычислений, необходимо выбрать ноту, которая пойдет в результат работы сети.

Помимо обычного выбора, основываясь на полученных вероятностях, существует подход, вносящий еще больше стохастики в процесс генерации

мелодии. Выбор ноты осуществляется с помощью дополнительного гиперпараметра temperature. Temperature отвечает за преобразование вероятностей и по своим свойствам напоминает работу активационной функции softmax. Результирующая формула выглядит следующим образом:

$$pred = \log(probab) \, / temp$$

$$probab = \frac{e^{pred}}{\sum_{i=1}^{N} e^{pred_i}}$$

, где probab – есть изначальный ряд распределения дискретной случайной величины, возвращаемый нейронной сетью.

Такая модификация процесса генерации мелодии позволяет регулировать то, насколько полученный результат будет случайным. Если параметр $temperature = 1$, то на распределение вероятностей не будет оказано никакого влияния. В случае, когда гиперпараметр принимает значения близкие к нулю, то распределение вероятностей изменяется так, что вероятность с максимальным значением становится равной 1, а все другие стремятся к 0, тем самым делая процесс генерации музыки более детерминированным. При увеличении параметра temperature, достигается обратный результат, энтропия распределения вероятностей становится больше, и шанс того, что будет выбрана нота, не соответствующая изначальному предсказанию сети, становится выше.

3.2 LSTM сеть

Данный раздел посвящен реализации архитектуры автоматической генерации музыки, основанной на LSTM нейронной сети.

Эта нейронная сеть хорошо зарекомендовала себя во многих сферах применения глубокого обучения, например, в контекстном анализе. LSTM сеть, в отличие от базовой RNN сети, имеет особенность запоминать важные и забывать ненужные знания в ходе своего обучения, что крайне важно в задачах генерации музыки. Также эта нейронная сеть способна учитывать

длительные тенденции в изменениях значений в обучающей выборке, что также способствует применению LSTM в задачах генерации музыки.

3.2.1 Данные и параметры

Для обучения модели основанной на LSTM использовался сборник “ballad” базы данных KernScores. Каждая мелодия была преобразована в последовательность целых чисел, а также специального знака, обозначающего длительность звучания ноты. Между каждой последовательностью, обозначающей музыку были расставлены символы “/”, которые сигнализируют о начале или о конце мелодии. После чего был образован единый файл, который после уже разбивался на обучающие выборки.

Количество уникальных символов в этом наборе данных равно 34. Параметр **sequence_length**, отвечающий за размер одного образца был выбран равным 64, что имеет под собой обоснование из теории музыки, так как именно такую длительность имеет музыкальная нотация с размером четыре четверти. Такой размер нотации является самым популярным и распространенным. Размер обучающей выборки равен 109263. После чего к данным было применено преобразование one-hot-encoding, необходимое для того, чтобы нейронные сети решали именно задачу *классификации*, а не *регрессии*.

Подробнее о параметрах обучения модели будет описано в параграфе 3.2.4. Обучение и анализ модели.

3.2.2 Архитектура сети

В рамках первой модели используется лишь небольшое количество слоев, представленных ниже:

- LSTM слой — модификация обычного рекуррентного слоя. Принимает на вход последовательность тензоров. Иначе этот слой можно воспринимать как, слой, получающий на каждом временном интервале

некоторое значение или тензор. Выход слоя состоит из последовательности векторов, соответствующих скрытому состоянию каждой ячейки LSTM, либо скрытому состоянию последней ячейки. В Keras это варьируется с помощью параметра return_sequences. В данной модели используются две последовательно размещенные LSTM сети. Такое размещение, позволяет второй LSTM сети выделять сложные высокоуровневые признаки, такие, например, как ритм. Вторым параметром сети является количество узлов, или другими словами размерность скрытого пространства.

- Dropout слой — это метод регуляризации, который исключат влияние некоторых нейронов во время прямого распространения. Во время обучения сети на каждой эпохе, по заранее заданной вероятности, существует шанс, что некоторые нейроны в сети, расположенной до dropout будут выключены. Несмотря на то, что слой прореживания (dropout) используется для борьбы с переобучением, он все еще необходим в задачах генерации музыки. Наличие dropout дает нам гарантию того, что между нейронами образуется равномерное распределение нужной для генерации музыки информации, а также уверенность в том, что сеть не будет повторять примеры из обучающего множества.
- Dense слой — полносвязный слой нейронной сети, где каждый вход соединен с каждым выходом предыдущего слоя. В данном случае, он используется только для подсчета распределения вероятностей звучания определенной ноты на выходе модели
- Activation слой — определяет тип функции активации, используемой в слое нейронной сети. В данной модели, используется softmax для подсчета распределения вероятностей.

В Таблице 1 [Приложение А. Архитектура LSTM сети] представлены параметры архитектуры нейронной сети. Количество узлов LSTM равно 256.

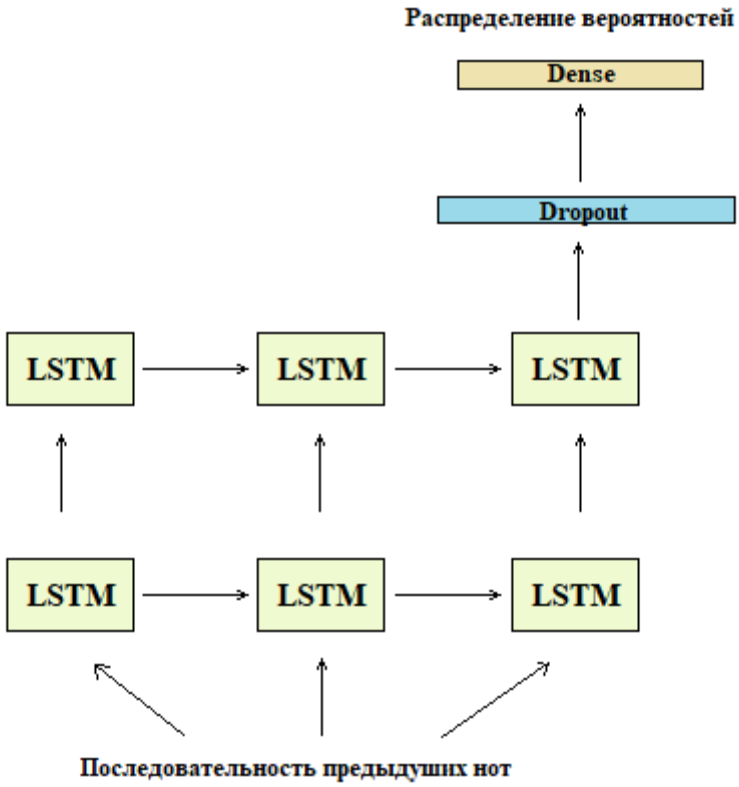

Рис.8 Представление архитектуры сети

На Рис.8 изображена архитектура нейронной сети.

3.2.3 Описание алгоритма генерации

Данная сеть работает по принципу языковых моделей. То есть прогнозирование следующего символа или ноты в последовательности, пользуясь информацией о предыдущих символах. На Рис. 9 изображена схема последовательной генерации.

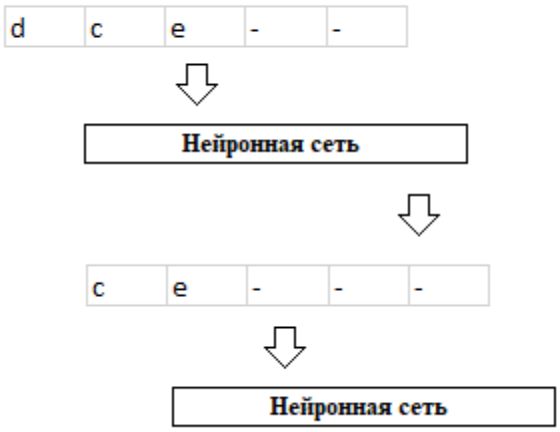

Рис.9 Принцип генерации языковых моделей

На вход модели подается последовательность нот размера **sequence_length** закодированных с помощью one-hot-encoding. Другими словами, в каждый момент времени сеть получает на вход ноту, представленную в виде вектора.

После чего данные проходят через LSTM ячейки. Когда вся последовательность символов будет передана, последний слой LSTM сети передаст свое скрытое состояние в полносвязный слой Dense, чьим выходом является вектор размерности map_size, где map_size это количество уникальных нот в словаре. Выход сети Dense передается в активационную функцию softmax, которая преобразует вектор в распределение вероятностей.

3.2.4 Обучение и анализ модели

Для обучения LSTM модели использовались следующие параметры:

- Функция потерь – категориальная кросс-энтропия
- Оптимизатор Adam со скоростью обучения 0,001
- Количество эпох обучения – 50

- Batch size – 128
- Dropout 30%
- Использовалась метрика Accuracy

На Рис.10 представлены графики изменения метрики Accuracy и функции потерь Loss в ходе обучения модели.

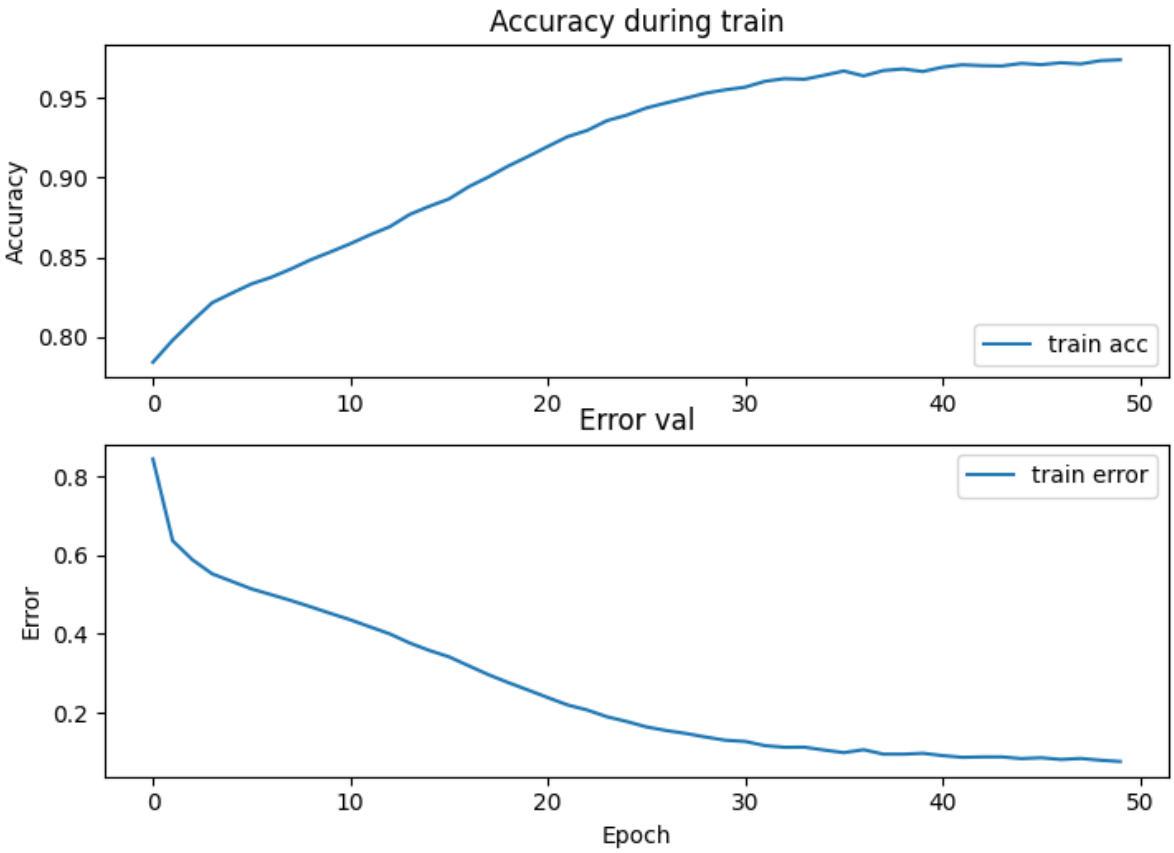

Рис.10 Графики метрики accuracy и функции потерь loss во время обучения

На Рис.11 и Рис. 12 представлены результаты работы системы генерации музыки, основанной на LSTM сети.

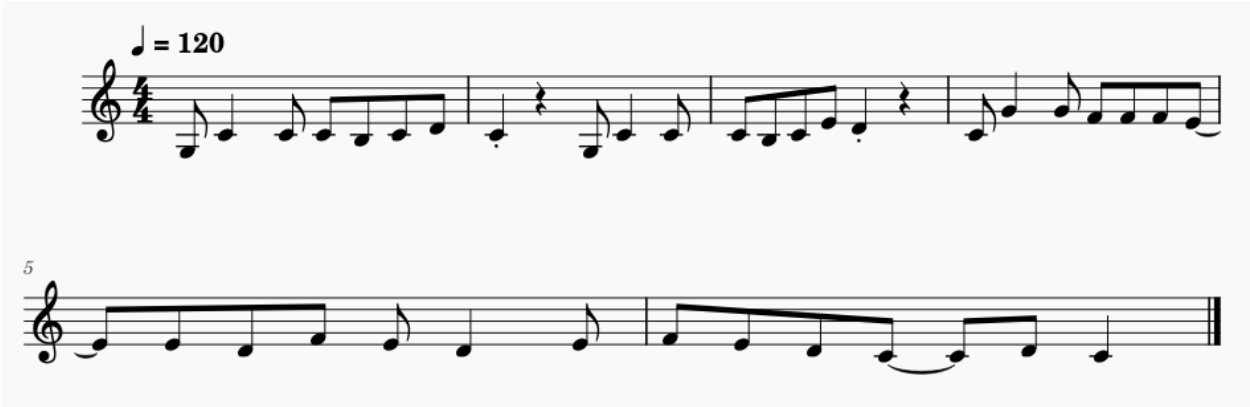

Рис.11 Музыкальная нотация сгенерированная
сетью с параметром temperature = 0

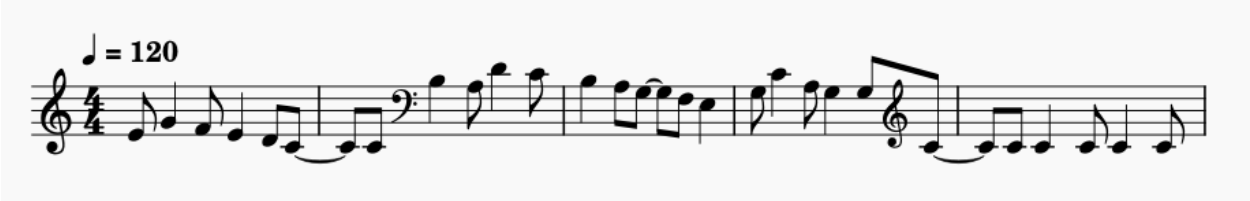

Рис.12 Музыкальная нотация сгенерированная
сетью с параметром temperature = 1

Как видно из результатов генерации, сеть выучила некоторые комбинации нот, или же аккорды, встречаемые в наборе данных. Также сеть смогла уловить смысл тоники, на Рис. 11 мелодия начинается с одной ноты и в конце постепенно стремится к похожей ноте, хоть и на тональность выше. Помимо этого, отчетливо видны повторяющиеся компоненты, а также отсутствие разного рода артефактов в виде быстрой смены нот. Модель смогла близко приблизиться к понятию такт и размер произведения, видно, что сумма длительности нот в каждом такте стремится к четверти, но не всегда ей равняется.

Таким образом LSTM сеть смогла выделить многие важные принципы составления музыки и успешно интерпретировать их.

Помимо этого, так как сеть обучалась на данных в которых есть разделительные символы “/”, обозначающие конец мелодии, то система

способна сама решать, когда ей следует заканчивать мелодию. То есть как только сеть прогнозирует символ “/”, то генерация мелодии заканчивается.

Но модель не идеальна, она все еще не может воспринимать сложные музыкальные концепции, повторяет заученные комбинации нот, а также требует начальной последовательности нот, для более качественной генерации.

Последняя проблема частично решается, если на вход сети подать 64 символа “/”, тогда “модель поймет”, что она уже встречала подобную конструкцию в обучающем наборе, и спрогнозирует наиболее часто встречаемую ноту с которой начинались мелодии.

Данная модель способна генерировать качественные мелодии, но она делает это соблюдая определенные правила и зависимости, встречаемые в обучающем наборе. Предпочтительно, чтобы генератор был способен выделять более общие, абстрактные принципы составления музыки. То есть, например, использовать комбинации нот, которые отсутствуют именно в данном обучающем наборе, но используются повсеместно в других мелодиях.

С подобными задачами хорошо справляются генеративно-состязательные сети. Поэтому, была разработана вторая модель, основанная на генеративно-состязательных сетях.

3.3 Генеративно-состязательная сеть

Данный раздел посвящен реализации системы автоматической генерации музыки, основанной на GAN архитектуре. Генеративно-состязательные сети отлично показали себя во всех задачах генеративного обучения за счет своей уникальной структуры.

3.3.1 Данные и параметры

Для обучения модели использовался тот же сборник “ballad” базы данных KernScores, что и для обучения LSTM сети.

Но в данной архитектуре используется другой подход к генерации мелодии. Если сеть, основанная на LSTM, создавала музыку последовательно предсказывая ноты на основе существующего контекста, состоящего из нот, то для генеративно-состязательной сети нужен уже другой подход:

- Генерация должна происходить не на основе существующей последовательности нот, а с помощью вектора шума заданной размерности
- Сгенерированный образец должен структурно совпадать с образцами из набора данных, на которых учится дискриминатор

Для этого необходимо, чтобы генератор возвращал в качестве результата целую последовательность нот, закодированных one-hot кодировкой. Размер этой последовательности регулируется параметром **sequence_length**. Сгенерированный сетью образец преобразуется с помощью $temperature$ в мелодию.

В ходе исследования различных архитектур сетей генератора и дискриминатора была выявлена следующая проблема: после длительного обучения генератор начинал прогнозировать некорректную последовательность символов:

– – – – – – …

То есть в качестве результата сеть возвращает последовательность служебных символов, обозначающих длительность ноты. Как только дискриминатор определял, что это подделка, генератор немного менял вероятности звучания других нот, чтобы подстроиться под изменения дискриминатора. Такого рода результаты называются формой. Они начинают возникать, когда генеративно-состязательные сети сталкиваются с коллапсом модели, при котором генератор способен создавать лишь небольшой набор образцов, обманывающих дискриминатор.

Для борьбы с коллапсом модели существует модификация – WGAN, основанная на функции потерь Вассерштейна [17]. Но, конкретно в данной работе, применение этой модификации не дало значительных результатов.

Изменение кодировки данных может помочь справится с этой проблемой. Вместо одного общего служебного символа "– " обозначающего длительность ноты, теперь используется множество подобного рода символов, каждый из которых будет обозначать длительность ноты определенной тональности.

Это привело к более стабильной работе генеративно-состязательных сетей, но увеличило размер словаря приблизительно в два раза, что значительно усложнило модели. Количество уникальных символов в обучающем наборе равно 64. Всего было подготовлено 109263 обучающих данных.

3.3.2 Архитектура сети

В ходе разработки системы были исследованы различные модели, выводы по которым представлены ниже.

Архитектура, в которой и генератор, и дискриминатор состоят только из полносвязных слоев оказалась неэффективна для данной задачи. Даже после 1000 эпох обучения дискриминатор не достиг $accuracy = 1$. Также

дискриминатор не смог уменьшить свои достаточно высокие потери, вследствие чего генератор не способен произвести хороший результат, так как он не может получить достаточного количества важной информации. После всевозможного перебора гиперпараметров, результат оказался прежним.

Тогда, было принято решение использовать вместо дискриминатора, основанного только на Dense слоях, дискриминатор, комбинирующий Dense и LSTM слои. Результат обучения оказался противоположным. Дискриминатор слишком быстро научился правильно определять настоящие данные от ложных. Сеть генератора, ввиду сильного дискриминатора, обладала слишком высокими потерями. На каждой итерации обновления генератора, ему удавалось уменьшить значение функции потерь только на небольшое число. Дискриминатор же все так же быстро обучался, адаптируясь к изменениям в генераторе. Вследствие чего результатом работы генератора был некоторый набор несвязанных между собой нот.

После того как в качестве генератора начала использоваться модель с LSTM слоями, ситуация изменилась. Обучение генератора и дискриминатора стало более стабильным, но, несмотря на изменение предобработки данных, происходил коллапс модели. Сеть в качестве результата возвращала всё ту же последовательность служебных символов, только теперь вероятности были распределены между всеми возможными длительностями.

Были сделаны следующие выводы, способствующие решению поставленной задачи:

- Дискриминатор и генератор должны быть примерно на одном уровне по сложности своей архитектуры, по количеству обучаемых параметров
- Дискриминатор должен обучаться быстрее генератора, так в качестве оптимизаторов для дискриминатора используется Adam с $learning\ rate = 0{,}001$, для генератора Adam с $learning\ rate = 0{,}0001$

В области генеративного обучения часто используются сети DCGAN (*Deep Convolutional Generative Adversarial Network*). Генеративно-состязательные сети с использованием сверточных слоев хорошо решают задачи, связанные с генерацией реалистичных изображений. Если рассмотреть сочинение мелодии как генерацию некоторой картинки, то возможно применить DCGAN и в данной задаче.

В качестве дискриминатора, используемого в модели, выбрана архитектура сверточной сети, представленная на Рис.13. Количество обучаемых параметров указано в Таблице 2 [Приложение Б. Архитектура генеративно-состязательной сети].

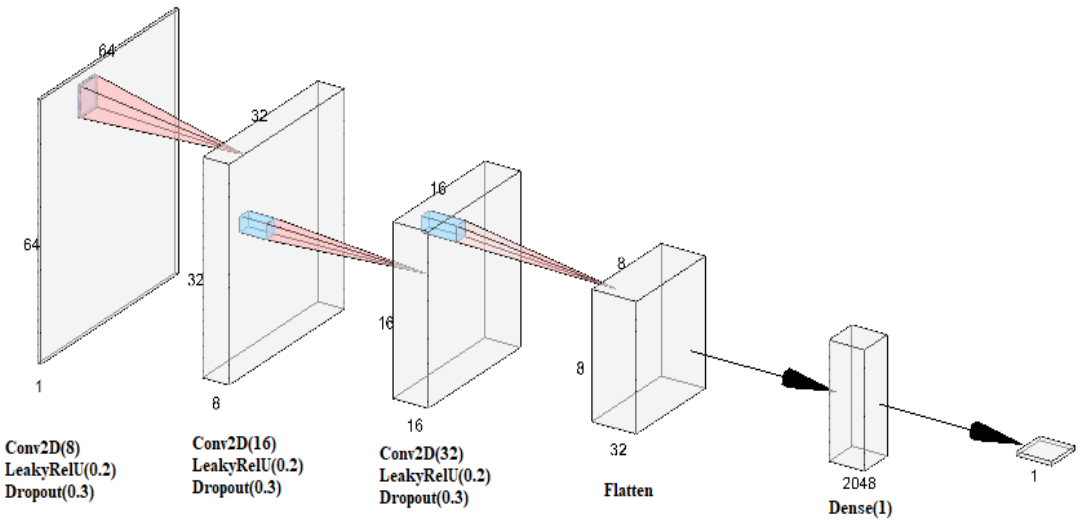

Рис.13 Архитектура дискриминатора

Архитектура генератора изображена на Рис.14. Количество обучаемых параметров указано в Таблице 3.

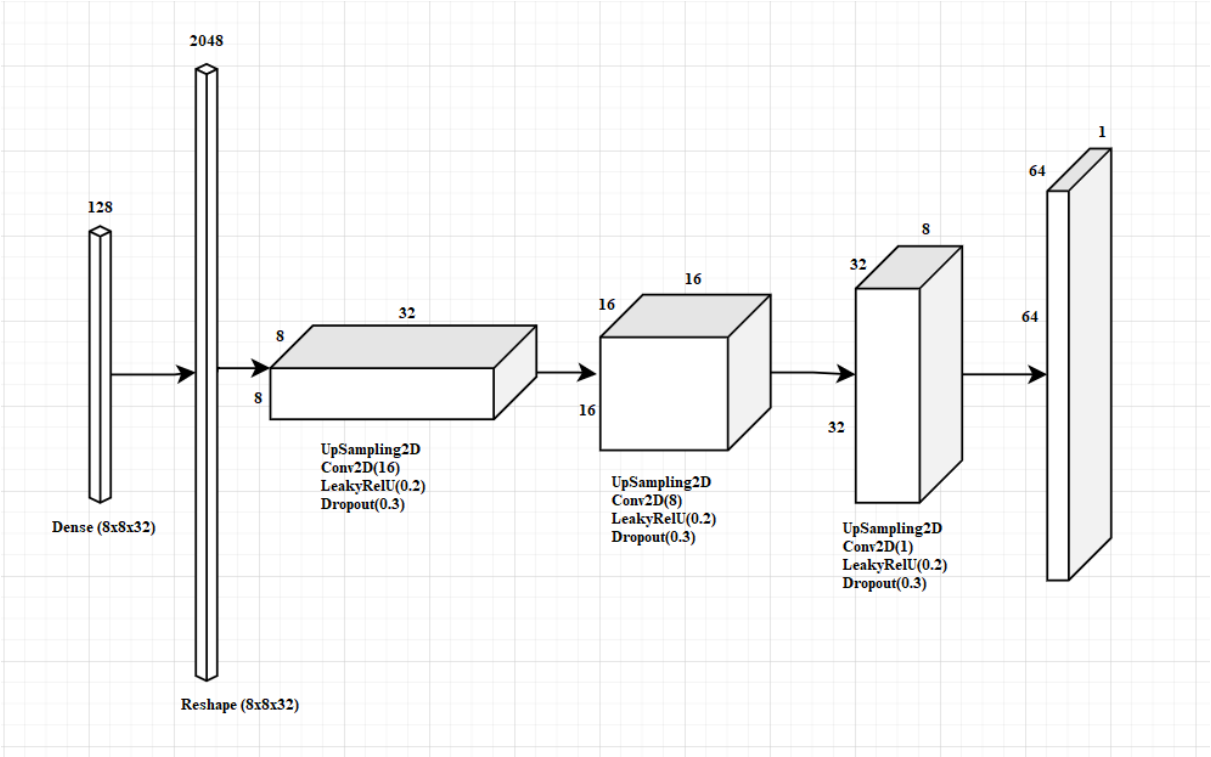

Рис.14 Архитектура генератора

После того, как дискриминатор скомпилирован, его веса фиксируются, после чего генератор добавляется перед входом дискриминатора, тес самым создавая комбинированной модель. Уже с помощью этой модели генератор обучается, пользуясь ответами, выдаваемыми дискриминатором. Комбинированная модель представлена в Таблице 4.

3.3.3 Описание алгоритма обучения GAN

Генеративно-состязательные сети уникальны тем, что их архитектура подразумевает то, что дискриминатор и генератор должны параллельно и равномерно обучаться. Алгоритм обучения сети, используемый в данной модели:

- На каждой итерации обучения, последовательно обучается дискриминатор, затем генератор
- Для того чтобы дискриминатор был сильнее генератора, его обучение происходит в цикле. В данной модели дискриминатор обучается дважды на одно обучение генератора.
- Дискриминатор обучается сначала на выборке размера batch_size реальных данных, после чего на данных сгенерированных еще не обновленным генератором. В первом случае высчитывается потеря дискриминатора относительно правильного ответа “1”, во втором относительно “0”.
- После того как дискриминатор обучился, происходит обновление генератора. На основе вектора шума размерности 128 сеть генерирует новый образец, после чего данный образец проходит через фиксированный дискриминатор, который выдает степень своей уверенности является ли этот фрагмент данных подделкой. Так как дискриминатор фиксирован, его обучения не происходит. Вычисляется потеря дискриминатора относительно правильного ответа “1”, после чего ошибка распространяется вглубь сети, меняя веса генератора. Таким образом генератор обучается генерировать данные так, чтобы дискриминатор на основании этих данных выдавал ответ “1”.

3.3.4 Обучение и анализ модели

Для обучения GAN модели использовались следующие гиперпараметры:

- Функция потерь – бинарная кросс-энтропия
- Оптимизатор Adam для генератора со скоростью обучения 0,0001
- Оптимизатор Adam для дискриминатора со скоростью обучения 0,001
- Количество эпох обучения – 100
- Batch size – 64
- Дополнительная метрика генератора – accuracy
- Дополнительная метрика дискриминатора – accuracy
- Размерность вектора шума генератора – 128
- Размер генерируемого образца – 64
- Количество обучений генератора за итерацию – 1
- Количество обучений дискриминатора за итерацию – 2

Модель обучалась на наборе данных, состоящем из 109263 образцов длины 64. На Рис.15-16 представлены графики изменения метрики Accuracy и функции потерь Loss в ходе обучения модели.

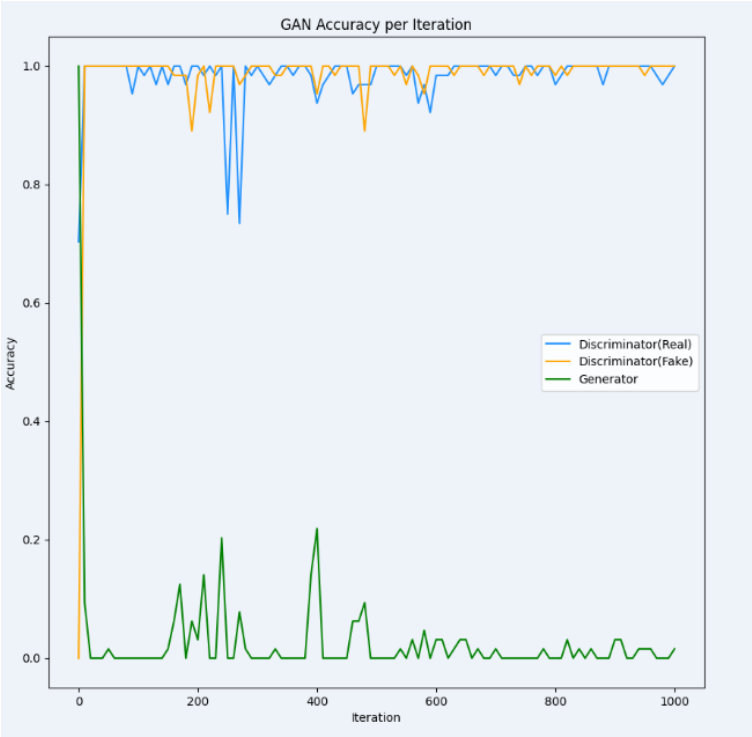

Рис.15 График метрики accuracy во время обучения

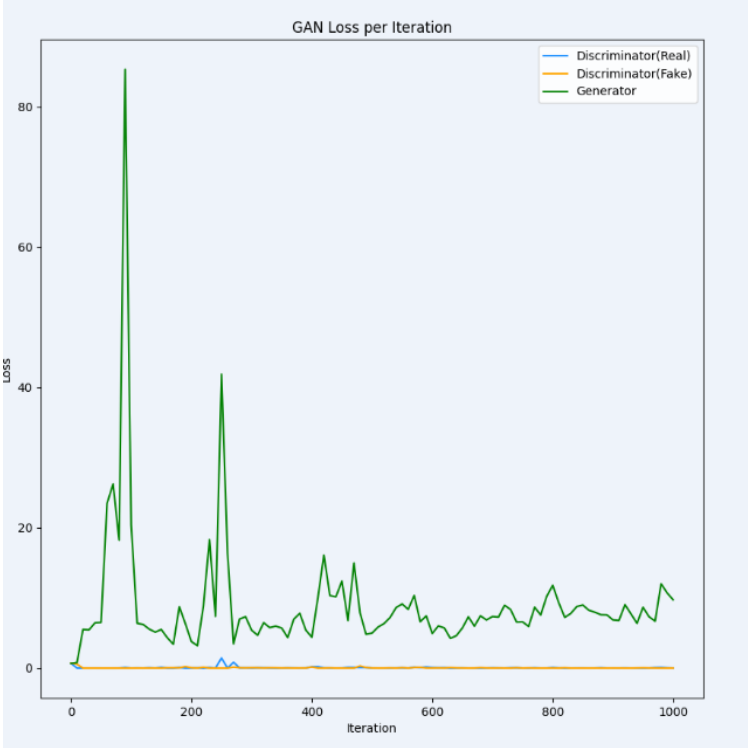

Рис.16 График изменения функции потерь во время обучения

На Рис. 17 представлен результат работы системы генерации музыки, основанной на GAN.

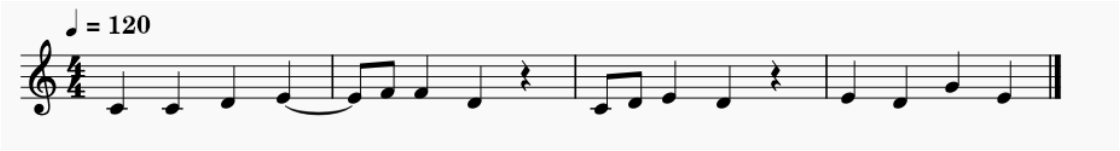

Рис.17 Музыкальная нотация сгенерированная сетью

Как видно из результатов генерации, сеть выучила некоторые базовые правила генерации музыки, также модель смогла сгенерировать мелодию размера четыре четверти.

3.4 Результаты

Модель, основанная на GAN, способна создавать мелодии, выбирая достаточно необычные и уникальные комбинации нот, однако она имеет множество недостатков:

- Модель способна генерировать только мелодии определенного, заранее заданного размера
- Ввиду того, что мелодии генерируется целиком, а не последовательно, ноты, следующие друг за другом, могут сильно отличаться в тональности
- Сеть сильно зависит от начального этапа обучения, например, при коллапсе модели, случившимся на первых итерациях, дальнейшее обучение будет безрезультативным.
- Результаты обучения и работы сети сильно зависят от случайно сгенерированных начальных весов.
- Для того чтобы получить хорошую модель, необходимо длительное, контролируемое извне обучение

Модель, основанная на LSTM, не обладает данными недостатками, она стабильно и не долго обучается, способна генерировать мелодию с хорошей локальной структурой нот, а также соблюдает глобальную структуру музыки

за счет возможности закончить генерацию мелодии в нужный момент, на определенной ноте.

Тем самым, в данной работе, реализованная LSTM модель имеет меньше недостатков, чем GAN модель. Вследствие чего, возможно, мелодии, сгенерированные LSTM сетью, будут более “качественными”. Для исследования выдвинутого предположения был проведен опрос, в котором приняло участие 16 человек. В нем было представлены 3 аудиофайла: мелодия из обучающего набора и мелодии, сгенерированные LSTM и GAN моделями. Результаты опроса представлены на Рис 18. Перед испытуемыми ставилась следующая задача: оценить каждую из мелодий по десятибалльной шкале, 0 соответствует музыке, созданной компьютером, 10 – созданной человеком.

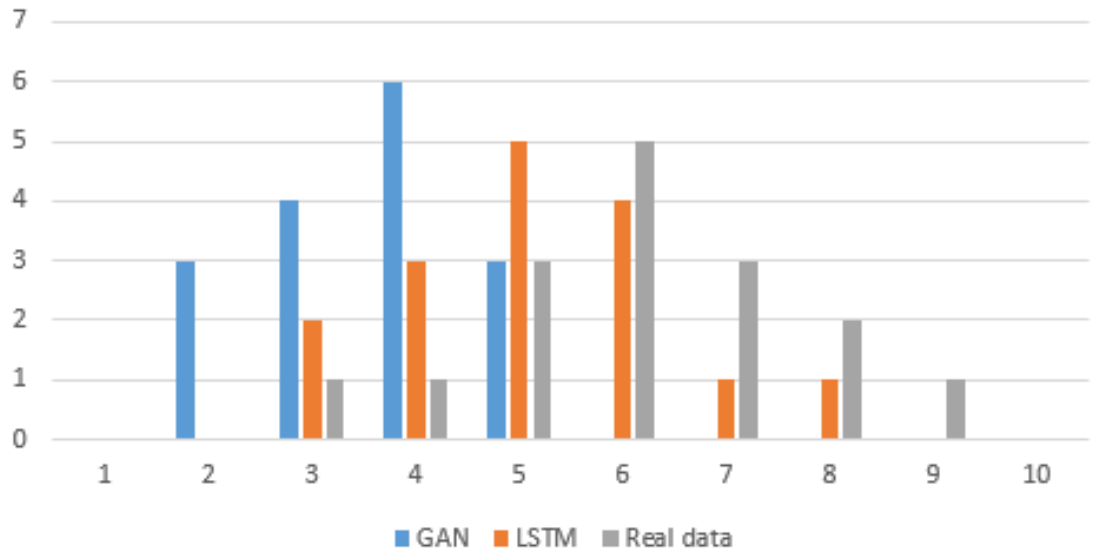

Рис.18 Результаты опроса

Из результатов опроса можно заключить, что LSTM модель показала неплохие результаты, мелодии, сгенерированные GAN было легко отличить от музыки, написанной человеком.

3.5 Выводы

На основании вычислительных экспериментов сделаны следующие выводы.

При использовании музыкальных данных в виде временного ряда сеть LSTM показала стабильно хорошие результаты. Мелодии, сгенерированные этой сетью, по качеству были достаточно близки к музыке, написанной человеком.

При применении представления музыки в виде временного ряда в GAN были получены средние результаты. Сделаны предположения о причинах данного явления. При этом с помощью описанного в данной работе способа представления музыкальных данных генеративно-состязательные сети способны генерировать полифоническую музыку.

Помимо этого, такая предобработка данных имеет множество преимуществ, в том числе ускорение обучения сетей, за счет уменьшения размера словарей.

Заключение

В выпускной квалификационной работе:

- Произведен выбор набора данных для обучения и выполнена его предобработка
- Реализована модель, основанная на LSTM
- Реализована модель, основанная на генеративно-состязательных сетях
- Обучены модели и проанализированы полученные результаты

Список использованной литературы

[1] Goodfellow, Ian J.; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua Generative Adversarial Networks, June 2014

[2] Gaëtan Hadjeres, François Pachet, Frank Nielsen DeepBach: A Steerable Model for Bach Chorales Generation, 3 Dec 2016

[3] F. Colombo, A.Seeholzer and W. Gerstner, "Deep artificial composer: A creative neural network model for automated melody generation": *Proc. Int. Conf. Evol. Biol. Inspired Music Art*, pp. 81-96, 2017.

[4] D. D. Johnson, "Generating polyphonic music using tied parallel networks": Proc. Int. Conf. Evolutionary and Biologically Inspired Music and Art, pp. 128-143, 2017.

[5] Nipun Agarwala, Yuki Inoue, and Axel Sly Music Composition using Recurrent Neural Networks, 2017

[6] R. K. H. Toh and A. Sourin, "Generation of Music with Dynamics Using Deep Convolutional Generative Adversarial Network»: 2021 International Conference on Cyberworlds (CW), 2021, pp. 137-140

[7] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks”, 2014.

[8] Andrew Shin, Leopold Crestel, Hiroharu Kato, Kuniaki Saito, Katsunori Ohnishi, Masataka Yamaguchi, Masahiro Nakawaki, Yoshitaka Ushiku, Tatsuya Harada Melody Generation for Pop Music via Word Representation of Musical Properties, 31 Oct 2017

[9] Bosch A. Hidden Markov Models. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer: Boston, 2011

[10] S. Hochreiter, J. Schmidhuber, “Long short-term memory” Neural Comput. Nov. 1997.

[11] Дэвид Фостер, Генеративное глубокое обучение. Творческий потенциал нейронных сетей - пер. с англ.: ДМК Пресс, 2020

[12] Christopher Olah. Understanding LSTM Networks, 2015

[13] Jean-Pierre Briot, Gaëtan Hadjeres, François-David Pachet, Deep Learning Techniques for Music Generation -- A Survey, 2017

[14] I.V. Sposobin “Elementary theory of music”, 2017

[15] Music-theory [Электронный ресурс] URL: https://www.musictheory.ru/index.php?option=com_content&view=article&id=5&Itemid=164&lang=ru

[16] Антонио Джулли, Суджит Пал. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow / / пер. с англ. – М.: ДМК Пресс, 2018. – 294 с.

[17] Martin Arjovsky, Soumith Chintala, Léon Bottou Wasserstein GAN, 26 Jan 2017

Приложение А. Архитектура LSTM сети

Полная архитектура LSTM сети представлена в Таблице 1.

Таблица 1 – Архитектура LSTM сети

№	Слой	Output shape	Количество параметров
1	Input_1 (Input Layer)	64x34	-
2	LSTM	64x256	297984
3	LSTM	256	525312
4	Dropout	-	-
5	Output dense	34	8738

Приложение Б. Архитектура генеративно-состязательной сети

Архитектура генератора сети представлена в Таблице 2, архитектура дискриминатора в Таблице 3, комбинированная модель описана в Таблице 4

Таблица 2 – Архитектура генератора

№	Слой	Output shape	Количество параметров
1	Input_1 (Input Layer)	64	-
2	Dense	2048	67584
3	Reshape	8x8x32	-
5	UpSampling2D	16x16x32	-
6	Conv2D 1 layer	16x16x16	32784
7	LeakyRelU	16x16x16	-
8	Dropout	-	-
9	UpSampling2D	32x32x16	-
10	Conv2D 2 layer	32x32x8	8200
11	LeakyRelU	-	-
12	Dropout	-	-
13	UpSampling2D	64x64x8	-
14	Conv2D 4 layer	64x64x1	513
15	Reshape	64x64	-
16	Softmax activation	64x64	-

Таблица 3 – Архитектура дискриминатора

№	Слой	Output shape	Количество параметров
1	Input_1 (Input Layer)	64x64x1	-
2	Conv2D 1 layer	32x32x8	520
3	LeakyRelU	32x32x8	-
4	Dropout	-	-
5	Conv2D 2 layer	16x16x16	8208
6	LeakyRelU	16x16x16	-
7	Dropout	-	-
8	Conv2D 3 layer	8x8x32	32800
9	LeakyRelU		-
10	Dropout		-
11	Flatten	2048	-
12	Dense	1	2049

Таблица 4 – Архитектура комбинированной сети

№	Слой	Output shape	Количество параметров
1	Input_1 (Input Layer)	64	-
2	Generator	64x64	109081
3	Discriminator	1	43577