Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики

К защите допустить:

Заведующая кафедрой
информатики

_____ Н. А. Волорова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

СЕРВИС ДЛЯ ИЗВЛЕЧЕНИЯ ХАРАКТЕРИСТИК МУЗЫКАЛЬНОЙ КОМПОЗИЦИИ С ПРИМЕНЕНИЕМ МАШИННОГО ОБУЧЕНИЯ

БГУИР ДП 1-40 04 01 00 097 ПЗ

Студент	Д. В. Шматков
Руководитель	П. С. Саттарова
Консультанты:	
от кафедры Информатики	П. С. Саттарова
по экономической части	К. Р. Литвинович
Нормоконтролёр	Н. Н. Бабенко

Рецензент

Минск 2017

РЕФЕРАТ

Пояснительная записка 72 с., 13 рис., 6 табл., 25 формулы, 24 источника.
ПРОГРАММНОЕ СРЕДСТВО, ВЕБ-СЕРВИС, УНИВЕРСИТЕТ, АНАЛИЗ МУЗЫКАЛЬНЫХ КОМПОЗИЦИЙ, МАШИННОЕ ОБУЧЕНИЕ, ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Цель настоящего дипломного проекта состоит в разработке программной системы, предназначенной для автоматизации извлечения характеристик музыкальных произведений.

В процессе анализа предметной области были выделены основные характеристики музыкальных композиций и способы их извлечения, которые в настоящее время практически не охвачены автоматизацией. Было проведено их исследование и моделирование. Кроме того, рассмотрены существующие средства, разрозненно разрабатываемые и применяемые сотрудниками университетов и отдельными компаниями (так называемые частичные аналоги). Выработаны функциональные и нефункциональные требования.

Была разработана архитектура программной системы, для каждой ее составной части было проведено разграничение реализуемых задач, проектирование, уточнение используемых технологий и собственно разработка. Были выбраны наиболее современные средства разработки, широко применяемые в индустрии.

Полученные в ходе технико-экономического обоснования результаты о прибыли для разработчика, пользователя, уровень рентабельности, а также экономический эффект доказывают целесообразность разработки проекта.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

API – Application Programming Interface (программный интерфейс приложения).

t-SNE – алгоритм машинного обучения, который относят к методам множественного обучения признаков [1].

music2vec – английское сокращение от music to vector (композиция в вектор).

MP3 – кодек третьего уровня, разработанный командой MPEG, формат файла для хранения аудиоинформации.

WAV – формат файла-контейнера для хранения записи оцифрованного аудиопотока.

FLAC – популярный свободный кодек, предназначенный для сжатия аудиоданных без потерь.

M4A – проприетарный (патентованный) формат аудиофайла с потерями.

JSON – JavaScript Object Notation (текстовый формат обмена данными, основанный на JavaScript).

XML – eXtensible Markup Language (расширяемый язык разметки).

JVM – Java Virtual Machine (основная часть исполняющей системы Java).

MSIL – Microsoft Intermediate Language (внутренний системный язык программирования Microsoft).

LLVM – Low Level Virtual Machine (универсальная система анализа, трансформации и оптимизации программ).

JIT – динамическая компиляция, технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы.

PyPy – интерпретатор языка программирования Python.

ANSI – американский национальный институт стандартов.

ISO – международная организация по стандартизации.

TPU – специализированные интегральные схемы, разработанные специально для машинного обучения.

ASIC – интегральная схема, специализированная для решения конкретной задачи.

ИИ – искусственный интеллект.

cgroups – механизм ядра Linux, который ограничивает и изолирует вычислительные ресурсы (процессорные, сетевые, ресурсы памяти, ресурсы ввода-вывода) для групп процессов.

LXC – система виртуализации на уровне операционной системы для запуска нескольких изолированных экземпляров операционной системы Linux на одном узле.

HTTP – HyperText Transfer Protocol (протокол прикладного уровня передачи данных).

WSGI – Web Server Gateway Interface (стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером).

UNIX – семейство переносимых, многозадачных и многопользовательских операционных систем.

fork – системный вызов, создающий новый процесс (потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.

exec – системный вызов, загружающий в пространство процесса новую программу.

Affero GPLv3 – свободная лицензия, созданная специально для таких программ, как вебприложения, так что пользователи, использующие изменённую программу через сеть, могут получить её исходный код.

Фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и
объединение разных компонентов большого программного проекта.

RNN – рекуррентная нейронная сеть.

Linux – семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты.

Debian – операционная система, основанная на Linux, состоящая из свободного ПО с открытым исходным кодом.

ВВЕДЕНИЕ

Подъем популярности сервисов электронного распространения медиаинформации дал беспрецендентный доступ пользователям к записям музыкальных композиций. Сейчас такие сервисы, как iTunes, Google Music, Яндекс Музыка, Spotify и другие, предоставляют мгновенный доступ к миллионам записей. Перед пользователями возникает проблема выбора следующей композиции для прослушивания. Для решения этой проблемы, а так же для того, чтобы облегчить ориентацию в этом большом количестве информации, сервисы электронного распространения записей предоставляют пользователям системы рекомендации контента.

Современные системы рекомендации контента обычно используют статистику прослушивания композиций пользователями, чтобы сделать рекомендации более точными. Однако на такие системы часто оказывают влияние смещения популярности в больших масштабах, из-за чего системы не могут рекомендовать композиции, которые менее популярны. Так же это затрудняет создание индивидуальных рекомендаций для пользователей, исходя из их предпочтений. Поэтому возникает необходимость извлекать информацию из композиций, а так же классифицировать сами копозиции. На сегодняшний день извлечение характеристик и классификация музыкальных композиций является интересной и соревновательной задачей, которой занимаются различные компании и исследователи. На данный момент сильное развитие получают способы, основанные на машинном обучении.

Целью данного дипломного проекта является создание сервис для извлечения музыкальных характеристик, который будет пригоден для дальнейшей интеграцией с другими системами. В ходе работы предстоит выполнить следующие задачи:

а) выбрать способ извлечения базовых характеристик музыкальных композиций (таких как спектр, ритм, продолжительность);

б) построить модель машинного обучения для излечения высокоуровневых характеристик;

в) собрать и упорядочить данные для обучения модели;

г) построить эффективную архитектуру сервиса;

д) построить API для интеграции с другими системами.

Реализация сервиса для извлечения характеристик музыкальных композиций позволит упростить задачу классификации контента для сервисов электронного распространения записей, а так же создать систему для создания более точных и индивидуальных рекомендаций для пользователей.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Извлечение музыкальной информации - небольшая, но интересная и быстрорастущая область исследований, которая охватывает большой диапазон продуктов, используемых по всему миру. Эта область охватывает несколько дисциплин научных знаний: музыковедение, психологию, обработку сигналов, машинное обучение и комбинации этих дисциплин. Задача, которую будет решать разрабатываемый сервис, относится к задачам извлечения музыкальной информации.

Несмотря на то, что извлечение музыкальной информации является еще небольшой областью исследований, для извлечения характеристик музыкальных приложений применяется достаточное количество методов. Эти методы различаются как на основаниии применяемых подходов, так и на основании уровней извлекаемых характеристик. На основании характеристик выделяют методы, которые извлекают низкоуровневые характеристики (спектр, ритм), и методы которые извлекают высокоуровневые характеристики (жанр, настроение). В основном, нас будут интересовать методы, которые извлекают высокоуровневые характеристики.

Для извлечения высокоуровневых характеристик выделяют два класса методов: основанные на математических функциях, в основе которых лежит машинное обучение. В музыке появляются новые направления, а старые могут изменяться, поэтому наиболее перспективными являются методы, которые способны постраиваться под переменчивую природу музыки. Под такие условия в большей степени подходят методы, в основе которых лежит машинное обучение.

Машинное обучение - класс методов, характерной чертой которых является не прямое решение задачи, а решение, которое строится в процессе обучения при применении решения к множеству сходных задач. В последнее время этот класс методов получил активное развитие и поддержку со стороны крупных компаний и университетов.

Цифровой аудиосигнал можно представить в виде изображения звуковой волны (см. рисунок 1.1).

Цифровой аудиосигнал представляет собой множество точек, которые расставлены на одинаковом расстоянии друг от друга (см. рисунок 1.2). Расстояние между точками называется частотой дискретизации. Чем меньше интервал (выше частота дискретизации), тем шире частотный диапазон, который можно закодировать таким образом.

Амплитуда колебаний зависит от времени звука и коррелирует с громкостью звука. А частота колебаний напрямую связана с высотой

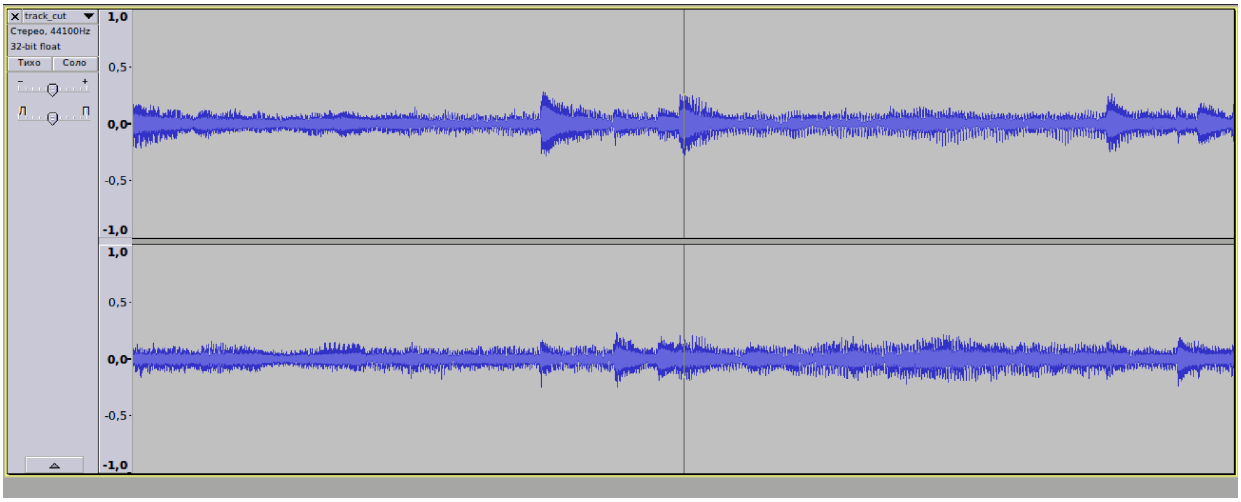

Рисунок 1.1 – Изображение звуковой волны

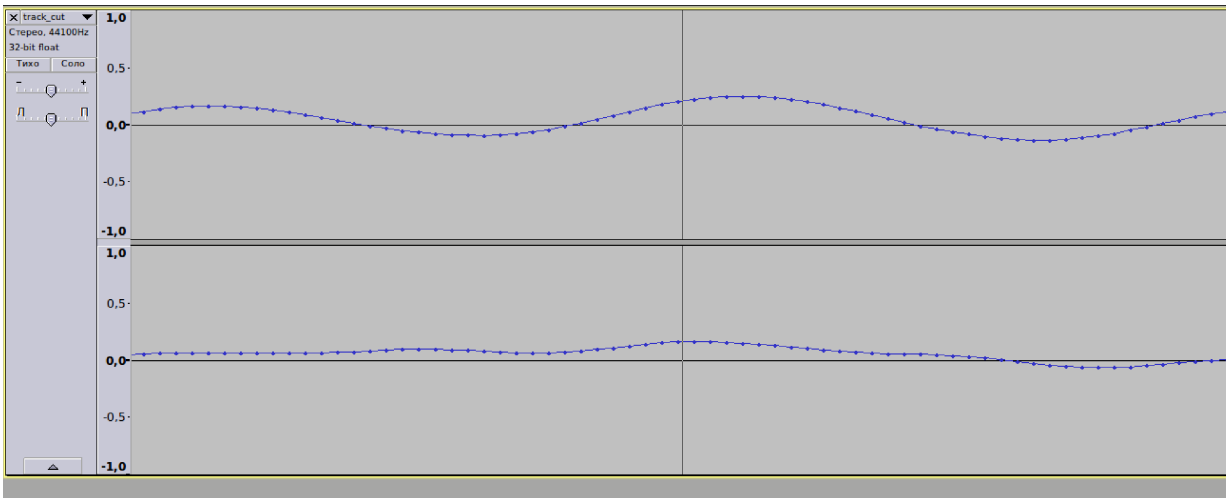

Рисунок 1.2 – Изображение звуковой волны в большом разрешении

звука. Для того, чтобы получить информацию о частоте колебаний, необходимо применить преобразование Фурье. Преобразование Фурье позволяет разложить периодическую функцию в сумму гармонических с разными частотами. Коэффициенты гармонических функций при сложении будут давать нам те частоты, которые мы хотим получить.

При применении преобразования Фурье ко всей звуковой дорожке мы получим "смазанный"во времени спектр. Для того, чтобы получить спектр, не теряя временной составляющей, необходимо применить к сигналу оконное преобразование Фурье. Оконное преобразование Фурье отличается от обычного тем, что мы делим наш сигнал на короткие отрезки (окна) и применяем к каждому преобразование Фурье. По-сути мы получаем набор спектров: отдельно для каждого отрезка. В результате мы получим картинку, которой можно описать звуковую дорожку (см. рисунок 1.3).

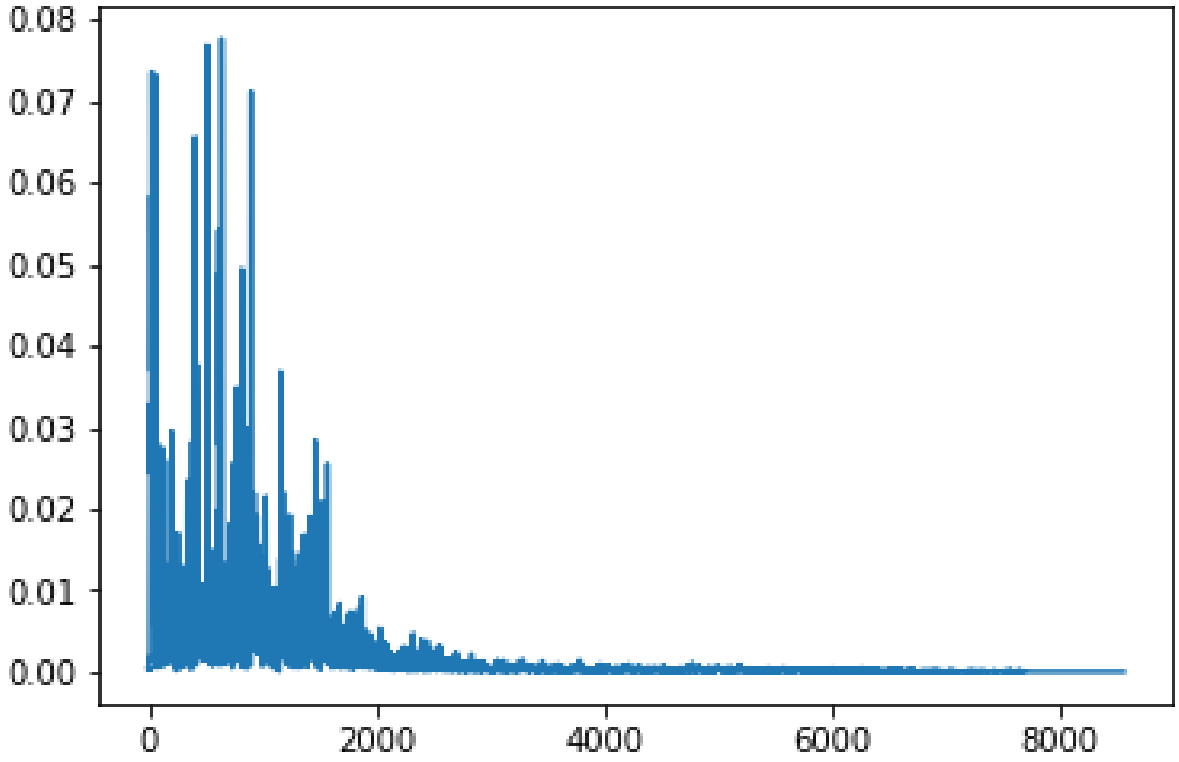

Рисунок 1.3 – Спектральная характеристика

Жанр композиции является высокоуровневой характеристикой. Для определения жанра необходимо разобраться, как человек воспринимает звук.

Ухо человека устроено так: звуковые волны смещают барабанную перепонку при взаимодействии с ней. Вибрации передаются во внутреннее ухо и считываются им. Смещение барабанной перепонки зависит от звукового давления, при том зависимость является не линейной, а логарифмической. Для измерения громкости принято использовать относительную шкалу - уровень звукового давления (измеряется в децибелах). Так же воспринимаемая громкость зависит и от частоты звука. Для оценки громкости звука используется логарифмическая единица измерения - фон. В шкале фонов, в отличие от децибелов, значения громкости связаны с чувствительностью на разных частотах человеческого уха. Частота 1000 Гц является чистым тоном, и уровень фона для неё численно равен уровню в децибелах. Для остальных частот используют поправки, которые представляют собой стандартизированное семейство кривых, называемых изофонами (см. рисунок 1.4).

Аналогично с громкостью, частота так же воспринимается человеческим ухом нелинейно. Для измерения воспринимаемой частоты человеческим ухом используется мел-шкала (см. рисунок 1.5). Шкала основана на статистической обработке субъективного восприятия звука на больших

данных. За 1000 мел взят звук с частотой 1000 Гц при уровне громкости 40 фон. За 0 мел взят звук частотой 20 Гц при уровне громкости 40 фон.

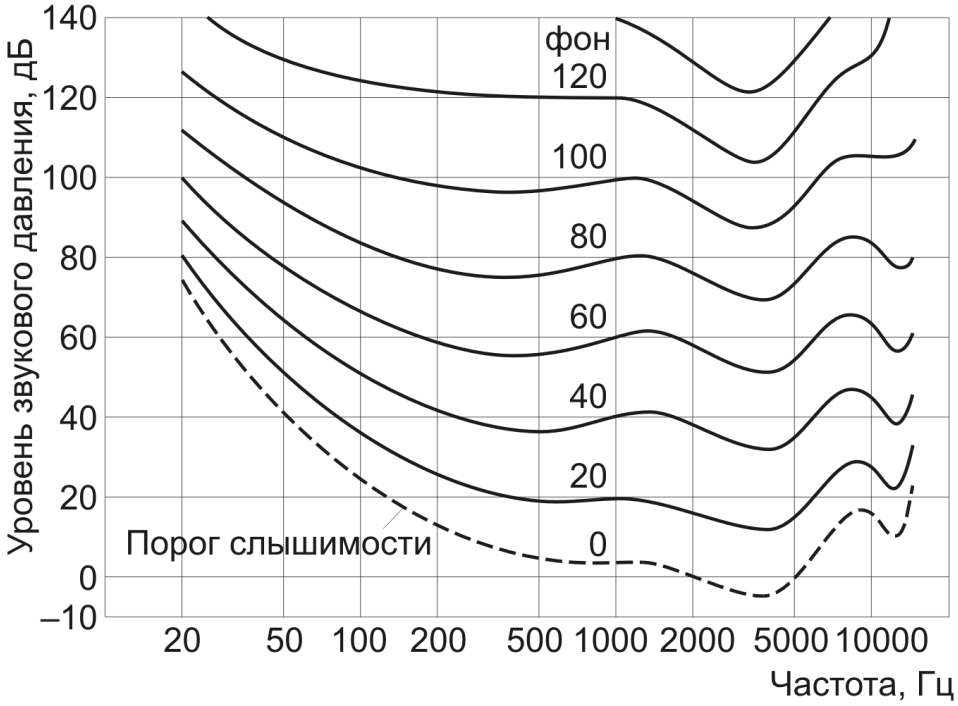

Рисунок 1.4 – Контур равных громкостей

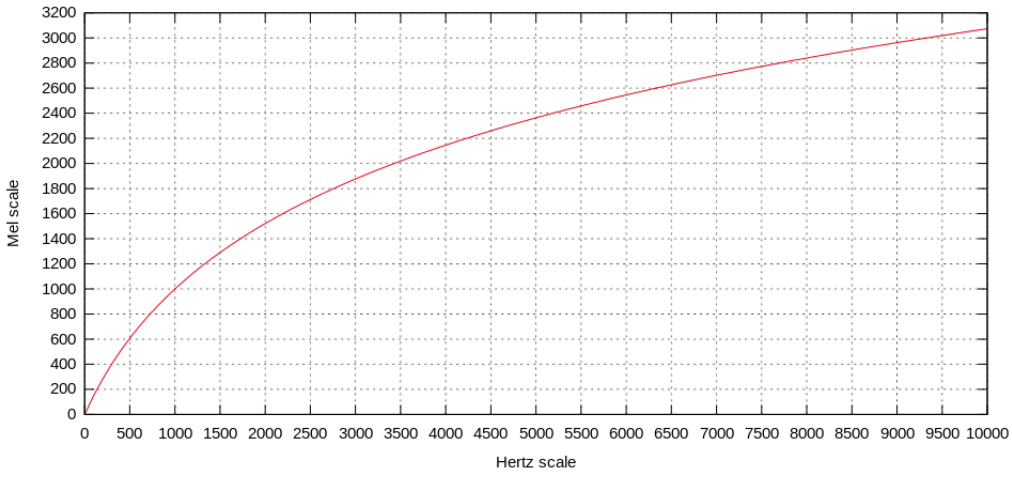

Рисунок 1.5 – Мел шкала

До недавнего времени большинство методов машинного обучения и обработки сигналов использовались неструктурированными архитектурами. Эти архитектуры обычно содержат не более одного или двух слоев нелинейных функций. Примерами неглубоких архитектур являются модели гауссовых смесей, линейные или нелинейные динамические системы, условные случайные поля, модели максимальной энтропии, метод опорного

вектора, логистическая регрессия, регрессия ядра, многослойные персептроны с одним скрытым слоем и экстремальными обучающими машинами. Например, метод опорного вектора используют модель разграничения линейных шаблонов с одним или нулевым уровнем преобразования объектов, когда используется трюк ядра или иным образом. Мелкие архитектуры были эффективны при решении многих простых или хорошо ограниченных проблем, но их ограниченное моделирование и репрезентативная власть могут создавать трудности при работе с более сложными приложениями реального мира, включающими естественные сигналы, такие как человеческая речь, естественный звук и язык, а также естественные изображения и визуальные сцены.

Однако механизмы восприятия информации человека (например, зрение и слух) указывают на необходимость глубоких архитектур для извлечения сложной структуры и создания внутреннего представления. Например, системы производства и восприятия речи человека оснащены четко стратифицированными иерархическими структурами для преобразования информации с уровня сигнала на лингвистический уровень. В аналогичном ключе человеческое зрение также является иерархической по своей природе на уровне восприятия. Естественно полагать, что современное окружение может быть усовершенствовано при обработке этих типов естественных сигналов, если могут быть разработаны эффективные алгоритмы глубокого обучения.

Исторически сложилось так, что понятие глубокого обучения было основано на исследованиях искусственной нейронной сети. Нейронные сети с обратной связью со многими скрытыми слоями, которые часто называют глубокими нейронными сетями, являются хорошими примерами моделей С глубокой архитектурой. Обратное распространение, популяризированное в 1980-х годах, было широко известным алгоритмом для изучения параметров этих сетей. К сожалению, обратное распространение само по себе не очень хорошо работает для для обучения сетей с большим количеством скрытых слоев. Широкое присутствие локальных оптимумов в невыпуклой целевой функции глубоких сетей является основным источником трудностей в обучении. обратное распространение основано на локальном градиентном спуске и обычно начинается в некоторых случайных начальных точках. Обратное распространение, которое не используется в связке с чем-то, обычно попадает в ловушку в локальных оптимумах, и степень ошибки значительно возрастает по мере увеличения глубины сетей. Эта трудность частично отвечает за провалы большинства исследований машинного обучения и обработки сигналов от нейронных сетей до неглубоких моделей с

функциями выпуклых потерь, для которых глобальный оптимум может быть эффективно получен за счет меньшей мощности моделирования.

Проблема оптимизации, связанная с глубокими моделями, была эмпирически облегчена с использованием трех методов: большего количества скрытых единиц, улучшенных алгоритмов обучения и лучших методов инициализации параметров.

Использование скрытых слоев со многими нейронами в глубокой нейронной сети значительно улучшает мощность моделирования глубокой нейронной сети и создает много достаточно близких к оптимальным конфигураций. Даже если параметры обучения захвачены в локальный оптимум, результат, получаемый глубокой нейронной сетью, может все же работать достаточно хорошо, так как вероятность плохого локального оптимума ниже, чем при использовании небольшого числа нейронов в сети. Однако использование глубоких и широких нейронных сетей потребовало бы огромной вычислительной мощности во время процесса обучения, и это одна из причин, почему только в последние годы исследователи начали серьезно изучать как глубокие, так и широкие нейронные сети.

Более эффективные алгоритмы обучения также способствовали успеху глубоких нейронных сетей. Например, стохастические алгоритмы обратного распространения вместо алгоритмов обратного распространения, оценивающих небольшие множества элементов, используемые для обучения глубоких нейронных сетей в настоящее время. Отчасти это связано с тем, что алгоритм стохастического градиента является наиболее эффективным алгоритмом, когда обучение осуществляется на одной машине, а обучающий набор является большим. Но, что более важно, алгоритм стохастичего градиентного спуска часто может выпрыгивать из локального оптимума из-за шумных градиентов, оцененных из одной или нескольких партий выборок. Аналогичные способности продемонстрировали другие алгоритмы обучения, такие как методы подпространства Крылова.

Для проблемы с обчень большим количеством локальных оптимумов для глубокой нейронной сети очевидно, что лучшие методы инициализации параметров приведут к улучшению моделей, поскольку оптимизация начинается с этих исходных параметров. Однако до недавнего времени не было известно как эффективно инициализировать параметры глубокой нейронной сети.

Метод инициализации параметров глубокой нейронной сети, который привлек наибольшее внимание, - это метод без учителя. В этих работах был введен класс глубоких байесовских вероятностных генеративных моделей, называемых сетью глубоких убеждений. Для изучения параметров

в сетей глубоких убеждений был разработан жадный алгоритм обучения. Поэтапно, обрабатывая каждую пару слоев в сети глубоких убеждений как ограниченную машину Больцмана. Это позволяет оптимизировать параметры сети глубоких убеждений с вычислительной сложностью, линейной по глубине сети. Позже выяснилось, что параметры сетей глубоких убеждений могут быть непосредственно использованы в качестве начальных параметров многослойного перцептрона или глубокой нейронной сети и приводят к лучшим результатам, чем те, которые случайным образом инициализируются после контролируемого обучения методом обратного распространения ошибки, когда набор тренировок мал. Таким образом, глубокие нейронные сети изучались с неконтролируемым предварительным обучением сети глубоких убеждений с последующей тонкой настройкой обратной передачи. В последнее время исследователи стали более осторожными в определении отличий между глубокими нейронными сетями и сетями глубоких убеждений.

Процедура предварительной обработки сетями глубоких убеждений не является единственной, которая позволяет эффективно инициализировать глубокие нейронные сети. Альтернативный неконтролируемый подход, который одинаково хорошо работает, заключается в том, чтобы каждый раз перестраивать глубокую нейронную сеть, рассматривая каждую пару уровней как де-шумирующий автокодер, регулируемый путем установки произвольного подмножества входных данных на ноль. Другой альтернативой является использование контрастных автоэнкодеров для одной и той же цели, предпочитая модели, которые менее чувствительны к входным вариациям, то есть наказывают градиент активности скрытых единиц по отношению к входам. Кроме того, была разработана разрешающая симметричная машина, которая имеет очень похожую архитектуру для ограниченной машины Больцмана как структурные блоки сетей глубоких убеждений. В принципе, разрешающая симментричная машина также может использоваться для эффективной инициализации параметров глубокой нейронной сети. Было показано, что помимо неконтролируемой предварительной подготовки, контролируемая предварительная обработка, или иногда называемая дискриминационная предварительная обработка, так же показала себя эффективным методом, а в случаях, когда размеченных данных достаточно, показывает себя лучше, чем неконтролируемые техники предварительной обработки. Идея дискриминирующей предварительной подготовки состоит в том, чтобы начать с модели с одним скрытым слоем, обученным с помощью метода обратного распространения ошибки. Каждый раз, когда мы хотим добавить новый скрытый слой, мы заменяем выходной слой случайным образом

инициализированным новым скрытым и выходным слоями и тренируем весь новый многоуровневый перцептрон или глубокую нейронную сеть с использованием метода обратного распространения ошибки. В отличие от неконтролируемых методов предварительной обработки, для метода дискриминирующей предварительной обработки требуются размеченные данные.

Как описано выше, глубокое обучение относится к довольно широкому классу методов машинного обучения и архитектур, с отличием от использования многих уровней нелинейной обработки информации, которые являются иерархическими по своей природе. В зависимости от того, как архитектуры и методы предназначены для использования, например, синтез и генерация или распознавание и классификация, можно широко классифицировать большую часть работы в этой области на три класса:

а) генерирующие глубокие архитектуры;

б) дискриминационные глубокие архитектуры;

в) гибридная глубокая архитектура.

Генерирующие глубокие архитектуры предназначены для захвата корреляций наблюдаемых или видимых данных высокого порядка для анализа или синтеза шаблоан и характеризуют совместные статистические распределения видимых данных и связанных с ними классов. В последнем случае использование правила Байеса может превратить этот тип архитектуры в дискриминационный.

Дискриминационные глубокие архитектуры, которые призваны непосредственно обеспечивать дискриминационную силу для целей классификации шаблонов, часто характеризуя последующие распределения классов, обусловленные видимыми данными.

Гибридные глубокие архитектуры, где целью является дискриминация, которая помогает с результатами генерирующих архитектур посредством лучшей оптимизации и регуляризации, или где дискриминационные критерии используются для изучения параметров в любой из моделей с глубокой генерацией.

В соответствии с общепринятой традицией машинного обучения, может быть естественным классифицировать методы глубокого обучения на глубокие дискриминационные модели, такие как глубокие нейронные сети, и модели с глубокой вероятностной генерацией, такие как сети глубокого убеждения. Однако в этой схеме классификации отсутствует ключевая информация, полученная в ходе глубоких исследований по вопросу о том, как генеративные модели могут значительно улучшить подготовку глубоких нейронных сетей и других глубоких дискриминационных моделей

посредством лучшей регуляризации. Кроме того, модели с глубокой генерацией необязательно должны быть вероятностными, например, глубоким автоенкодером, многокоординатным автоэнкодером. Тем не менее, традиционная двухсторонняя классификация действительно указывает на несколько ключевых различий между глубокими дискриминационными моделями и глубокими порождающими вероятностными моделями. По сравнению с этими двумя, глубокие дискриминационные модели, такие как глубокие нейронные сети, обычно более эффективны для обучения и тестирования, более гибкие для построения и более подходящие для сквозного обучения сложным системам. С другой стороны, глубокие вероятностные модели легче интерпретировать, проще встраивать знания домена, проще составлять и легче справляться с неопределенностью, но, как правило, трудноразрешимы для вывода и обучения для сложных систем.

Традиционная нейронная сеть использовалась для распознавания речи в течение многих лет. При использовании в одиночку её производительность обычно ниже, чем современные системы скрытых марковских моделей с вероятностями наблюдения, аппроксимированными гауссовскими моделью смеси. С тех пор, как несколько лет назад технология глубокого обучения была успешно применена к анализу звука и крупным задачам распознавания речи путем интеграции мощной дискриминирующей обучающей способности глубокой нейронной сети с возможностями последовательного моделирования скрытых марковских моделей.

В экспериментах раннего фонетического распознавания использовалась стандартная объектная функция на основе кадра в классификации статических характеристик, кросс-энтропия, для оптимизации весов глубокой нейронной сети. Параметры перехода и оценки языковой модели были получены из скрытой модели маркова и прошли обучение независимо от весов глубокой нейронной сети. Тем не менее, в течение долгой истории исследований скрытых моделей Маркова было известно, что критерии классификации последовательности могут быть очень полезными для повышения точности распознавания речи и распознавания. Это связано с тем, что критерии классификации последовательностей более непосредственно коррелируют с показателем эффективности, чем кросс-энтропия.

Преимущество использования таких классификационных критериев последовательности было показано на неглубоких нейронных сетях. Один популярный тип критерия классификации последовательностей, максимальная взаимная информация, был успешно применен для изучения весов глубокой нейронной сети для задачи распознавания голоса. Использование кросс-энтропии для обучения глубокой нейронной сети для распознавания

последовательности абонентов явно не учитывает тот факт, что соседние кадры имеют меньшие расстояния между назначенными распределениями вероятности по меткам класса звонков. Чтобы преодолеть этот недостаток, можно оптимизировать условную вероятность всей последовательности меток, учитывая всю видимую характеристику высказывания или эквивалентную скрытую функциональную последовательность, выделенную глубокой нейронной сетью. Чтобы оптимизировать условную вероятность журнала на данных обучения, мы берем градиент над параметрами активации, параметрами перехода и весами нижнего слоя, а затем продолжаем обратное распространение ошибки, определенной на уровне предложения.

Далее была разработана сверточная структура, которая накладывается на ограниченную машину Больцмана при создании сети глубокого убеждения. Свертка выполняется во времени путем совместного использования весов между скрытыми единицами в попытке обнаружить одну и ту же инвариантную функцию в разные моменты времени. Затем выполняется операция максимального пула, где достигаются максимальные активации по малым временным окрестностям скрытых единиц, что доказывает некоторую локальную временную инвариантность. Полученная сверточная сеть глубокого убеждения применяется к аудио и речевым данным для ряда задач, включая распознавание музыки исполнителя и жанровую классификацию, идентификацию спикера, гендерную классификацию говорящего и классификацию звонка, с обнадеживающими результатами.

Понятие свертки во времени было создано в нейронных сетях временного смещения как мелкая нейронная сеть, разработанная на ранних этапах исследований задачи распознавании речи. Только недавно при использовании глубоких архитектур, было обнаружено, что частотное распределение веса более эффективно для высокопроизводительного распознавания звонков, чем во временной области, как в предыдущей технологии нейронных сетей временного смещения. Эти исследования также показывают, что проектирование пула в глубокой сверточной нейронной сети для правильного компромисса между инвариантностью к длине голосового тракта и дискриминацией между звуками речи приводит к еще лучшему распознаванию. Эти результаты также указывают на направление соотношения между дискриминацией траектории и инвариантностью, выраженной во всей динамической структуре речи, определенной в смешанных временных и частотных областях, с использованием свертки и объединения. Более того, самые последние исследования показывают, что сверточные нейронные сети также полезны для широкого распознавания речи на основе словаря и далее демонстрируют, что несколько сверточных слоев

обеспечивают еще большее улучшение, когда сверточные слои используют большое количество ядер свертки или карт признаков.

В дополнение к ограниченной машине Больцмана, глубоким нейронным сетям, другие глубокие модели также были разработаны и опубликованы в литературе для обработки речи и связанных с ней приложений. Например, глубоко структурированное поле случайных состояний, которое содержит много уровней случайных состояний, успешно использовалось в задаче идентификации языка, распознавании звонков, последовательной разметки на естественном языке и доверительная калибровка в распознавании речи. Кроме того, в то время как рекуррентные нейронные сети дали успешные результаты в ранних исследованиях в распознавании звонков, было непросто повторять результаты из-за сложности в обучении, не говоря уже о расширении для более крупных задач распознавания речи. С тех пор алгоритмы обучения для рекуррентных нейронных сетей значительно улучшились, и в последнее время были получены гораздо лучшие результаты с использованием рекуррентных нейронных сетей. Рекуррентные нейронные сети также недавно были применены к приложениям для обработки музыки, где в RNN исследуется использование выпрямленных линейных скрытых единиц вместо логистической или тангенциальной нелинейности. Исправленные линейные единицы вычисляют максимум из ноля и значения и приводят к уменьшенным градиентам, меньшей диффузии в рекуррентных нейронных сетях и более быстрому обучению.

2 ОБЗОР СУЩЕСТВУЮЩИХ АНАЛОГОВ

Существует ряд компаний и сервисов, которые решают задачу извлечения характеристик музыкальных композиций и задачи, достаточно близкие к данной. Рассмотрим наиболее популярные и известные из них.

2.1 The Echo Nest

The Echo Nest - компания из Соммервиля, которая занимается разработкой сервиса для анализа музыкальных композиций и составлении рекомендаций. Их продукт собирает информацию, используя акустический и текстовый анализ. Текстовый анализ состоит в том, что любое упоминание музыкальной композици, которое найдено в интернете, проходит через системы The Echo Nest. Эти системы настроены на то, чтобы извлекать ключевые слова и термины. Этим данным потом присвается собственный вес, который говорит об их важности. Акустический анализ начинается с того, что композиция разбивается на небольшие кусочки. Затем для каждого сегмента определяются громкость, тембр и другие характеристики. Далее полученные данные объединяются и анализируются. Анализ проводится с помощью методов машинного обучения, что позволяет понять композицию на высоком уровне. Объединение акустического и текстового анализа позволило создать мощную технологию, которая сделала компанию мировым лидером в алгоритмах анализа музыки. В 2014 году компанию купил мировой гигант стримминга музыки - Spotify.

2.2 Niland

Niland - компания из Парижа, которая занимается поиском и реккомендациями музыкальных композиций. Их продукт анализирует композицию, используя акустический анализ. Решаемые задачи разделяются на 2 типа: оценка похожести и классификация. Для оценки похожести используется подход music2vec. То есть преобразование композиции в вектор из значений характеристик. Для классификации композиций используется преобразование музыки в вектор для дальнейшей обработки с помощью алгоритмов машинного обучения. Преобразование музыки в вектор включает в себя:

а) вычисление спектрограммы, т.е. вычисление интенсивности различных частот в различные моменты времени;

б) извлечение кратковременных особенностей, т.е. свойств сигнала, которые имеют высокую частоту обновления;

в) моделирование статического распределения кратковременных особенностей и объединение их в вектор.

В результате получается вектор большой размерности, порядка тысячи элементов, который впоследствие используется для классификации композиций и извлечения характеристик высокого уровня.

2.3 Music Technology Group

Music Technology Group - исследовательская группа с факультета информации и коммуникационных технологий университета Pompeu Fabra, Барселона. Специализируется на вычислительных задачах в области музыки и звука. Свои исследования группа основывает на знаниях из других областей, таких, как обработка сигналов, машинное обучение и взаимодействие человека с компьютером. Темы исследований, которыми группа занимается в данный момент:

а) обработка аудиосигналов, т.е. спектральное моделирование для синтеза и трансформации звука;

б) описание звука и музыки, т.е. семантический анализ и классификация аудио;

в) продвинутое взаимодействие с музыкой, т.е. разработка интерфейсов для создания и изучения музыки;

г) звуковые и музыкальные сообщества, т.е. технологии социальных сетей для музкальных и звуковых приложений.

Группа создала библиотеку для определения степени похожести и классификации музыки, для описания музыки с помощью высокоуровневых характеристик - Gaia. Обрабатываются данные в несколько этапов: подготовка данных, обучение на подготовленных данных, проверка результатов. Подготовка данных включает в себя:

а) удаление метаданных;

б) извлечение тональных характеристик;

в) нормализацию выделенных характеристик;

г) преобразование низкоуровневых характеристик в нормально распределенные величины.

Для обучения используется метод опорных векторов с полиномиальным ядром (однородным). В качестве метрики точности используется n-кратная кроссвалидация.

2.4 DeepMind

DeepMind - компания, которая занимается искусственным интеллектом. Ранее она была известна под названием Google DeepMind. Компания занимается решением проблем, связанных с интеллектом, т.е. занимается разработкой обучающихся алгоритмов общего назначения. На данный момент компания сосредоточила свои усилия на разработке интеллекта, который способен играть в компьютерные игры - от стратегических до аркад.

Сандер Дилеман, ученый из этой компании, стал соавтором статьи [2], в которой утверждается, что глубинное обучение способно гораздо лучше справляться с рекомендациями, чем колаборативная фильтрация. В качестве инструмента использовалась сверточная нейронная сеть с 7-8 слоями. Для визуализации данных использовался алгоритм t-SNE. Алгоритм определял интструменты, аккорды, и даже гармонии и прогрессии.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Требования к проектируемому программному средству

По результатам изучения предметной области, анализа литературных источников и обзора существующих систем-аналогов сформулируем требования к проектируемому программному средству.

Сервис должен предоставлять простой API для извлечения характеристик из аудиозаписей. Для поддержки большего числа пользователей и эффективного процесса интеграции с системами пользователя сервису необходимо поддерживать основные форматы взаимодействия между сервисами:

а) JSON;

б) XML.

Сервис будет являться модулем, который достаточно просто встраивать в существующие системы. Для повышения удобства использования развертывание модуля должно быть простым процессом, который не занимает много времени.

Для подготовки композиций к дальнейшему анализу, а так же для их анализа, в сервисе реализованы сложные математические вычисления. Для того, чтобы повысить комфортность использования, сервису следует анализировать композиции за приемлемое время. Так же необходимо снизить требовательность к характеристикам компонентов вычислительной системы, на которой будет запущен сервис. Поэтому необходимо оптимизировать вычисления, производимые для анализа композиций.

Музыкальные жанры подвержены изменению. Для того, чтобы поддерживать аналитическую составляющую в актуальном состоянии, необходимо иметь возможность обновлять части системы. Поэтому сервис должен быть построен из независимых компонентов, согласованных на уровне интерфейсов.

Аудиозаписи могут храниться в различных форматах, поэтому необходимо реализовать поддержку обработки основных форматов распростанения цифровых аудиозаписей, таких как mp3, wav, flac, m4a.

Для предоставления как можно более полной информации об аудиозаписи извлекаемые характеристики должны быть различных уровней, т.е. высокоуровневые и низкоуровневые. Определим основные характеристики композиции для извлечения:

а) длина;

б) громкость;

в) количество ударов в минуту;

г) жанр.

Так же необходимо учитывать, что человеческий слух воспринимает информацию несколько иначе, чем компьютер. Поэтому нужно делать поправку на особенности человеческого слуха при анализе аудиозаписей.

3.2 Построение архитектуры

Сформулировав требоваяния к программному продукту, приступим к проектированию.

Необходимо определиться с методом преобразования адиозаписей к виду, который будет точно описывать композицию. Существует большая разница между характеристиками композиции, которые влияют на предпочтения пользователей, и характеристиками, которыми обладает сам сигнал. Поэтому работать непосредственно со звуковыми сигналами недостаточно эффективно, а так же достаточно долго. Спектрограммы ускоряют процесс работы, но так же не настолько эффективны, поскольку не учитывают особенности человеческого уха. Человеческое ухо, как говорилось ранее, воспринимает звук несколько иначе, что требует определенных корректировок. Для того, чтобы учесть поправки на восприятие звука ухом человека, можно отобразить спектрограмму на мел-шкалу. В результате мы получим мел-спектрограмму. Работать с мел-спектрограммой достаточно долго ввиду её большого размера. Необходимо найти способ сжать информацию до приемлемых размеров, при этом не потеряв в точности.

Для того, чтобы сжать информацию и учесть поправки на человеческое ухо, было решено использовать мел-частотные кепстральные коэффициенты.

Достоинства этого метода:

а) используется спектр сигнала, что позволяет учесть волновую природу звука;

б) спектр проецируется на мел-шкалу, что позволяет учесть восприятие частот человеческим ухом;

в) возможность сжать количество информации количеством вычисляемых коэффициентов.

Осталось понять, как преобразовать сигнал в набор коэффициентов. Первым делом нам нужен спектр исходного сигнала, который мы получаем с помощью преобразования Фурье. Для того, чтобы не потерять временную составляющую, воспользуемся оконным преобразованием Фурье. Теперь полученный спектр нам нужно расположить на мел-шкале. Для этого мы используем окна, равномерно расположенные на мел-оси (см. рисунок 3.1).

Если перевести график, изображенный на рисунке 3.1, в частотную шкалу, можно увидеть график, изображенный на рисунке 3.2.

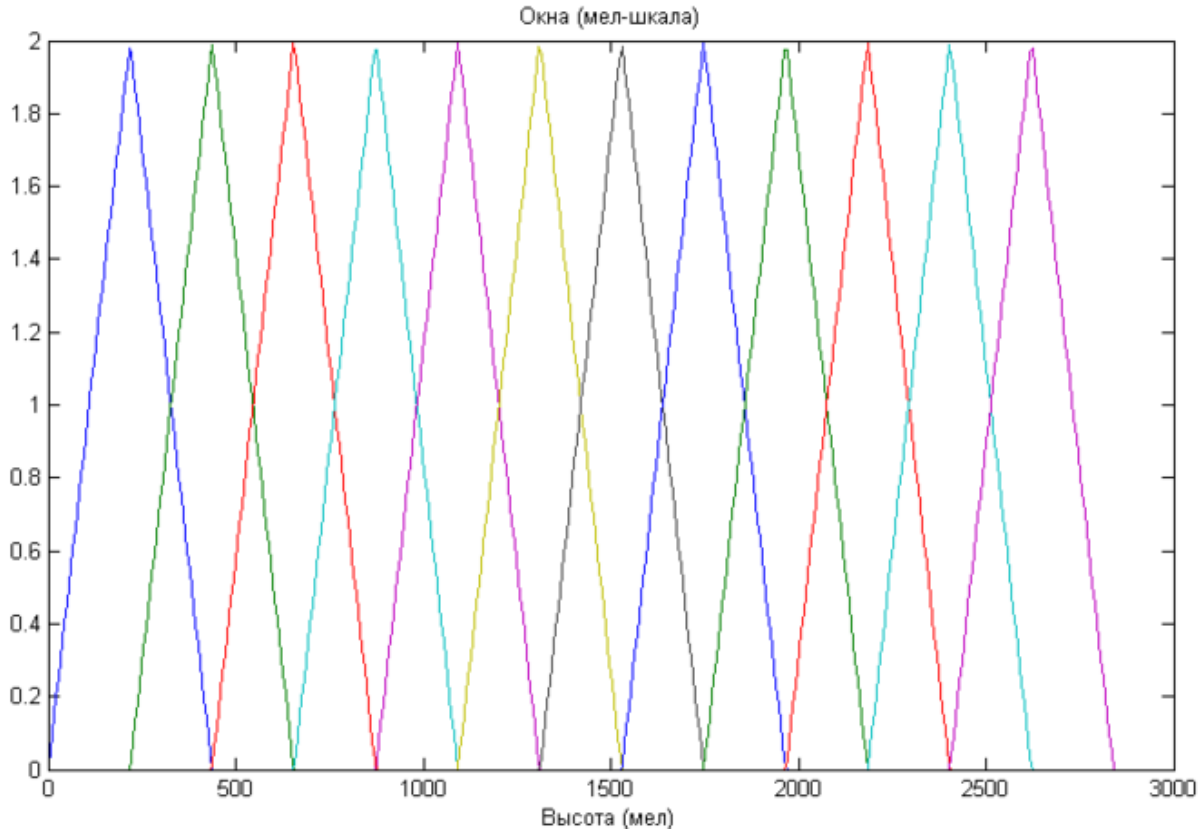

Рисунок 3.1 – Окна на мел оси

На этом графике заметно, что окна «собираются» в области низких частот, обеспечивая более высокое «разрешение» там, где оно необходимо для извлечения. Простым перемножением векторов спектра сигнала и оконной функции найдем энергию сигнала, которая попадает в каждое из окон анализа. Мы получили некоторый набор коэффициентов, но это еще не те коэффициенты, которые мы ищем. Пока их можно было бы назвать мел-частотными спектральными коэффициентами. Возводим их в квадрат и логарифмируем. Нам осталось только получить из них кепстральные, или «спектр спектра». Для этого мы могли бы еще раз применить преобразование Фурье, но лучше использовать дискретное косинусное преобразование.

Таким образом мы имеем очень небольшой набор значений, который при извлечении успешно заменяет тысячи отсчетов речевого сигнала.

Преобразовав сигнал, необходимо создать инструмент, который будет извлекать характеристики из полученных данных. Для этого решено использовать методы, основанные на машинном обучении. Для того, чтобы применять машинное обучение, необходимо построить модель, которая

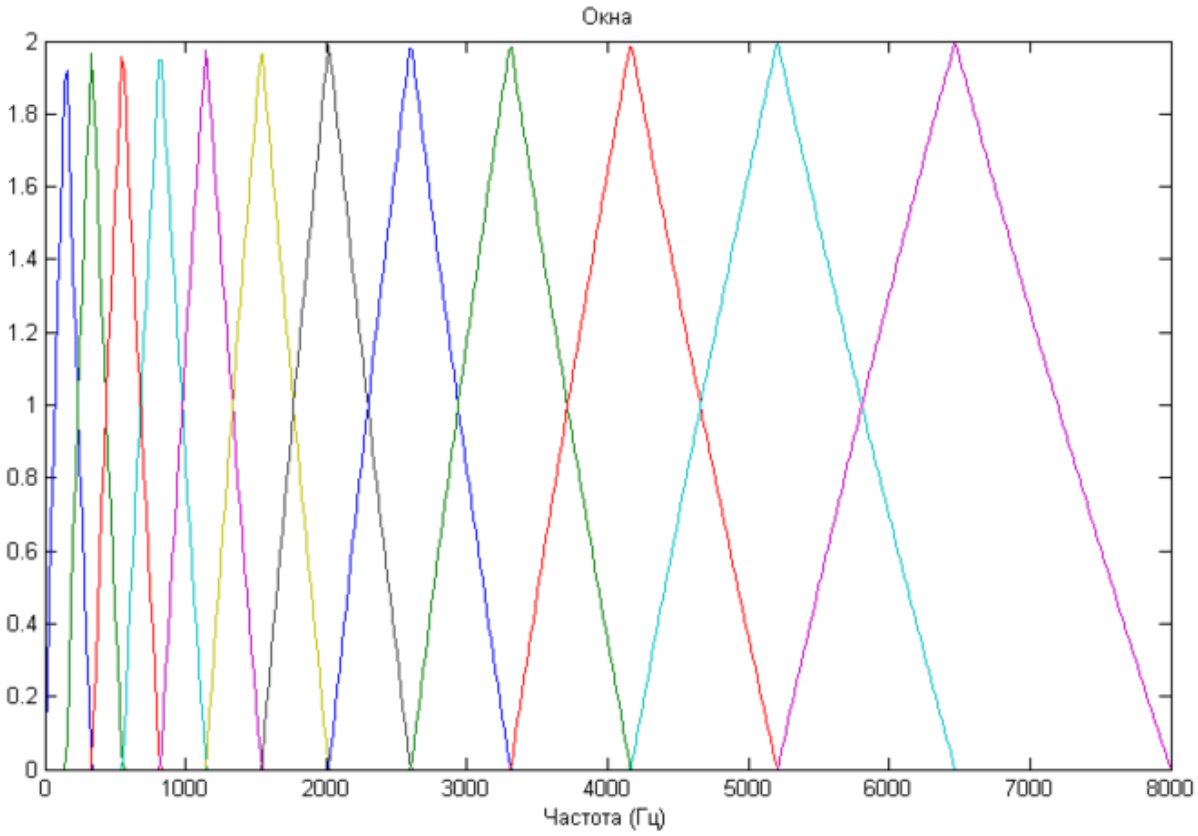

Рисунок 3.2 – Окна на мел оси, переведенные в частотную шкалу

будет являться искомым инструментом. Для построения модели необходимо разработать структуру модели, а так же выбрать библиотеку для обучения.

В качестве библиотеки для обучения была выбрана TensorFlow от компании Google. Она предоставляет большой набор инструментов, при помощи которых можно обучать модели на различных системах. Так же TensorFlow помогает работать с высокой степенью оптимизации, что ускоряет процесс обучения и обработки данных.

Для того, чтобы ускорить процесс прототипирования и повысить уровень абстракции, поверх TensorFlow будет использоваться библиотека Keras.

Связка Keras – TensorFlow позволит оптимально расходовать ресурсы системы, на которых будет запускаться сервис, при этом будет сохранен высокий уровень абстракции, что повысит читабельность и универсальность кода.

Построим архитектуру модели, которая будет извлекать высокоуровневые характеристики из музыкальных композиций. Для извлечения признаков хорошо себя показывают глубинные нейронные сети. Для извлечения признаков из матриц больших размерностей, к которым относится полученный список мел-кепстральных коэффициентов, лучше

всего использовать сверточные нейронные сети. Но для извлечения временных паттернов лучше всего себя зарекомендовали рекуррентные нейронные сети. Для получения наилучшего результата, скомбинируем сверточные нейронные сети и рекуррентные нейронные сети в глубинной нейронной сети. В результате получим нейронную сеть, в которой будет 6 слоев (см. рисунок **??**). Первые 4 слоя являются четырехслойной сверточной нейронной сетью, которая будет извлекать локальные признаки. Следующие 2 слоя являются двухслойной рекуррентной нейронной сетью, которая предназначена для агреггирования временных шаблонов. Это более эффективный подход, чем, например, использовать усреднение результатов.

Полученная нейронная сеть может быть больших размеров. Для того, чтобы сократить время на загрузку нейронной сети из дискового пространства в память, следует держать сеть в оперативной памяти. Наиболее эффективно хранить сеть в отдельном процессе, в который будут поступать данные для обработки. Это снизит затраты по памяти, а так же позволит обрабатывать большее количество данных за единицу времени. Процесс, который хранит и использует нейронную сеть, решено реализовать в виде демона.

Для того, чтобы предоставить API, необходимо реализовать сервер, который обеспечит сетевое взаимодействие по заданному протоколу. Для реализации части сервера, которая отвечает за бизнес-логику, будет использоваться микрофреймворк Flask. Flask является микрофреймворком, что обеспечит необходимую функциональность с потреблением минимального количества ресурсов. Эта часть сервера будет использоваться для маршрутизации запросов, сериализации и десериализации данных, а так же для обмена данными с демоном, который отвечает за обработку данных. Для того, чтобы повысить пропускную способность, часть сервера, которая отвечает за сетевое взаимодействие, будет использоваться HTTP сервер Gunicorn. Он поддерживает использование нескольких рабочих процессов для обработки запросов и одного мастер-процесса для балансировки нагрузки между рабочими процессами.

Необходимо организовать передачу данных между демон-процессом, который отвечает за обработку данных, и сервером. Для этого следует использовать очередь задач, которая будет использоваться для последовательной обработки данных, так же необходимо использовать кеширующую систему для того, чтобы обеспечить обратное взаимодействие. В качестве очереди задач будет использоваться RedisMQ, которая использует в качестве хранилища данных базу данных Redis. В качестве кеша, который будет обеспечивать обратное взаимодействие, так же будет использоваться Redis.

Преимущество Redis в данном случае заключается в том, что эта технология хранит данные в оперативной памяти, увечиливая её расход, зато это сокращает время, которое потребуется для обмена данными.

Для того, чтобы контроллировать и запускать демон и сервер, следует использовать систему контроля процессов. Такой системой является Supervisor. С его помощь процессы сервера и демона будут запускаться, так же они будут перезапускаться в случае неожиданного завершения работы какого-либо из процессов. Supervisor так же позволит хранить логи приложений в указанном месте для того, чтобы обеспечить мониторинг работы системы, что поможет так же и для отладки процессов в случае их аварийного завершения.

Чтобы предоставить сервис в виде модуля, который удобно встраивать в сторонние системы, необходимо все компоненты, описанные выше, поместить в контейнер, который будет предоставляться пользователю. В качестве системы, которая обеспечит управление и развертывание контейнеров, следует использовать Docker. Преимущество Docker заключается в том, что он предоставляет неполную виртуализацию, что позволяет существенно экономить вычислительные ресурсы. Так же Docker обеспечит переносимость контейнера на другие системы, что позволит развертывать сервис с минимальными затратами. Особенности виртуализации Docker позволяют уменьшить размер контейнера, что является дополнительным удобством при переносе сервиса.

3.3 Построение модели машинного обучения

Подробнее рассмотрим построение модели для определения жанра композиции. Для этого используют много подходов: метод опорных векторов, линейные регрессии, байесовы сети, нейронные сети и др. Остановимся на нейронных сетях, как на самом перспективном и передовом методе обработки данных.

Для того, чтобы повысить точность классификации, необходимо иметь достаточно высокий уровень абстракции. Поэтому небходимо воспользоваться методами глубого обучения. Глубокое обучение – это часть более широкого семейства методов машинного обучения – обучения представлениям, где векторы признаков располагаются сразу на множестве уровней. Эти признаки определяются автоматически и связывают друг с другом, формируя выходные данные. На каждом уровне представлены абстрактные признаки, основанные на признаках предыдущего уровня. Таким образом, чем глубже мы продвигаемся, тем выше уровень абстракции. В

нейронных сетях множество слоев представляет собой множество уровней с векторами признаков, которые генерируют выходные данные.

Мы хотим, чтобы гибкость семейства функций росла при увеличении количества данных. В нейронных сетях мы изменяем количество скрытых элементов в зависимости от объема данных. Наши модели машинного обучения должны стать композиционными. Натуральные языки используют композиционность, чтобы представлять более сложные идеи. В глубоком обучении мы используем:

а) распределенные представления;

б) глубокую архитектуру.

Давайте предположим, что данные поступают к нам не все сразу, а партиями. Партии могут поступать или параллельно, или последовательно. Параллельное поступление партий являет собой распределенное представление (обучение представлениям). Последовательное поступление данных похоже на обучение представлениям, но с несколькими уровнями. Композиционность дает нам возможность эффективного описания мира вокруг нас.

Нейронные сети разделяются на много типов:

а) сети прямого распространения;

б) нейронные сети Хопфилда;

в) цепи Маркова;

г) сверточные нейронные сети;

д) рекуррентные нейронные сети;

е) развертывающие нейронные сети;

ж) сети с долгой краткосрочной памятью и др.

Для того, чтобы выбрать что-то, необходимо формализовать задачу. В исходном виде задачей является определение высокоуровневой характеристики музыкальной композиции, которая называется жанр. В более формальном виде задачу можно написать так: определить к какому классу принадлежит данная музыкальная композиция. Эта задача относится к задаче классификации.

В результате применения преобразования к композиции, мы получим матрицу большой размерности. Для того, чтобы классифицировать такую матрицу, необходимо извлечь из нее признаки. Для того, чтобы извлекать признаки из матрицы большого размера, стоит использовать сверточную нейронную сеть.

Для того, чтобы понять выбор данной архитектуры, надо понять как она работает. В отличие от обычной нейронной сети, слои CNN состоят из нейронов, расположенных в 3-х измерениях: ширине, высоте и глубине,

то есть измерениях, которые формируют объем. Матрица предыдущего слоя сканируется небольшим окном и пропускается сквозь набор весов, а результат отображается на соответствующий нейрон текущего слоя. Таким образом, набор плоскостей представляет собой карты признаков, и каждая плоскость находит свои участки матрицы в любом месте предыдущего слоя. Входной слой предназначен лишь для подачи входной матрицы на следующий за ним слой и распределения входной матрицы по плоскостям. Следующий за входным слой называется свёрточным. Каждый нейрон в плоскости свёрточного слоя получает свои входы от некоторой области предыдущего слоя (локальное рецептивное поле), то есть входная матрица предыдущего слоя сканируется небольшим окном и пропускается сквозь набор весов, а результат отображается на соответствующий нейрон свёрточного слоя (см. рисунок 3.3). В итоге матрица будет преобразована в вектор признаков, который можно использовать для дальнейше классификации.

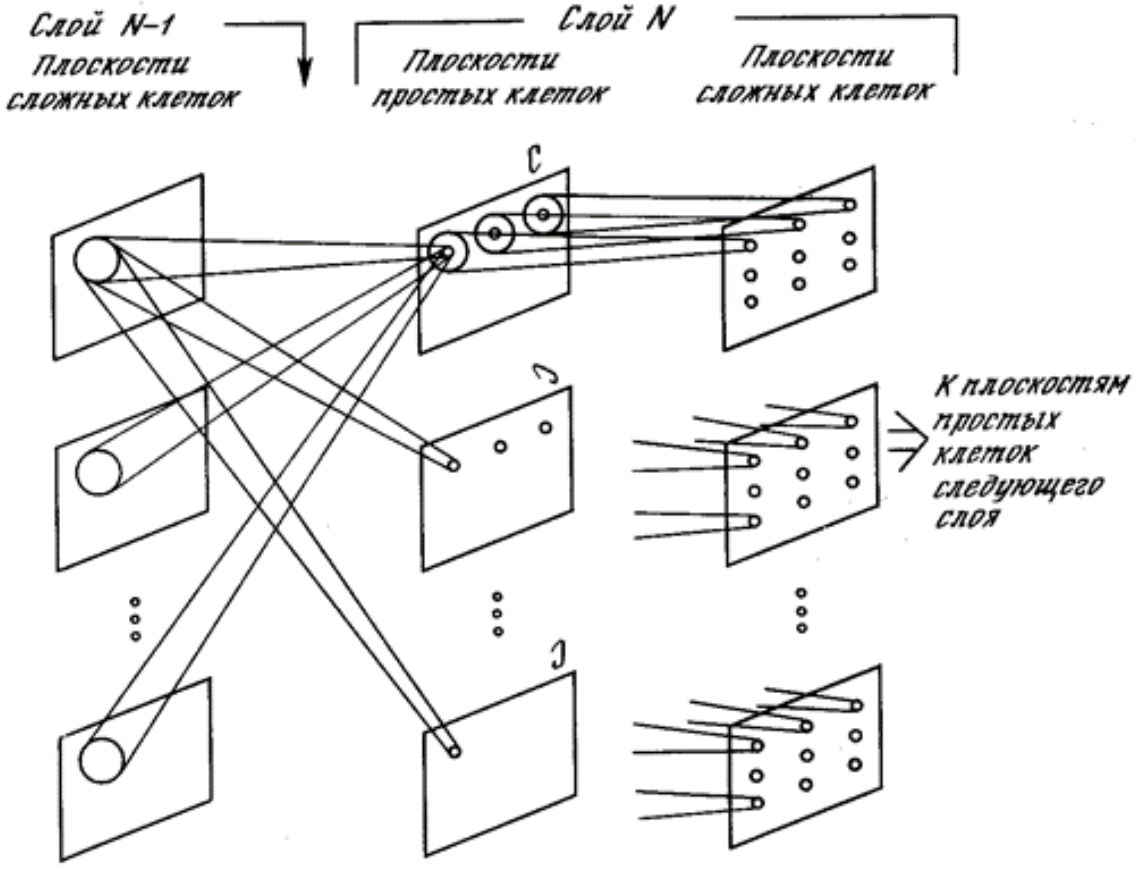

Рисунок 3.3 – Принцип извлечения признаков сверточной нейронной сетью

Для реализации необходимо разобраться из чего состоят слои сверточной нейронной сети. Слои сверточной нейронной сети состоят из:

а) слой свертки, умножает значения фильтра на исходные значения элементов матрицы (поэлементное умножение), после чего все эти умножения суммируются;

б) слой активации, применяет функцию активации к каждому элементу;

в) слой пулинга, выполняет операцию по понижающей дискретизации пространственных размеров (ширина и высота).

Для того, чтобы оптимизировать обучение нейронной сети, необходимо выбрать функцию активации. Обычно в качестве функции активации используются сигмоидальные функции, например гиперболический тангенс. Использование гиперболического тангенса нецелесообразно, потому что на его вычисление уходит слишком много ресурсов. Поэтому стоит обратить внимание на другие функции активации. В нашем случае используется ELU (см. рисунок 3.4). ELU имеет отрицательные значения, что позволяет ей приближать средние активации к нулю. Ноль означает ускорение обучения, потому что они приближают градиент к естественному градиенту ELU.

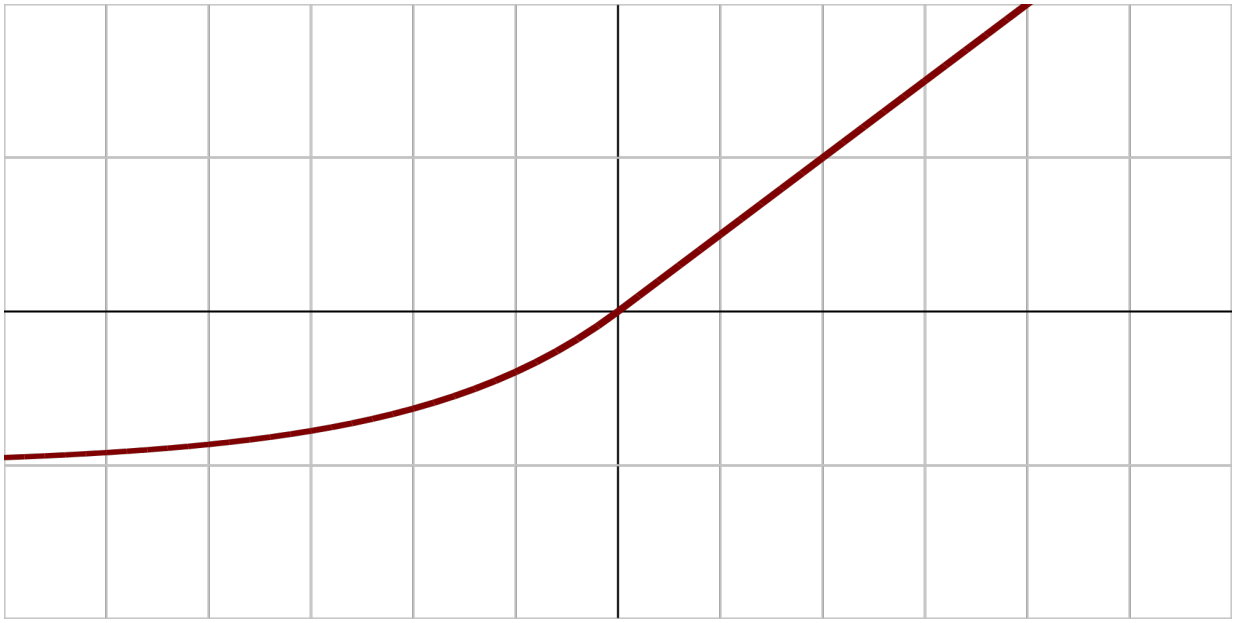

Рисунок 3.4 – График функции активации ELU

В дальнейшем нам необходимо классифицировать композицию по полученным признакам. Полученные признаки являются временной последовательностью, то есть их нельзя обрабатывать отдельно друг от друга. Для того, чтобы обрабатывать такие последовательности необходимо, чтобы на обработку текущих значений вляли текущие значения. Таким свойством обладает рекуррентная нейронная сеть.

Основное отличие рекурентных сетей (Recurrent Neural Network, RNN) от традиционных заключается в логике работы сети, при которой

каждый нейрон взаимодействует сам с собой. На вход таким сетям как правило передаётся сигнал, являющийся некоторой последовательностью. Каждый элемент такой последовательности поочерёдно передаётся одним и тем же нейронам, которые своё же предсказание возвращают себе вместе со следующим её элементом, до тех пор пока последовательность не закончится. Такие сети, как правило, используются при работе с последовательной информацией – в основном с текстами и аудио/видео-сигналами. Элементы рекурентной сети изображают как обычные нейроны с дополнительной циклической стрелкой, которая демонстрирует то, что кроме входного сигнала нейрон использует также своё дополнительное скрытое состояние. Если «развернуть» такое изображение, получится целая цепочка одинаковых нейронов, каждый из которых получает на вход свой элемент последовательности, выдаёт предсказание и передаёт его дальше по цепочке как своего рода ячейку памяти. Нужно понимать, что это абстракция, поскольку это один и тот же нейрон, который отрабатывает несколько раз подряд (см. рисунок 3.5).

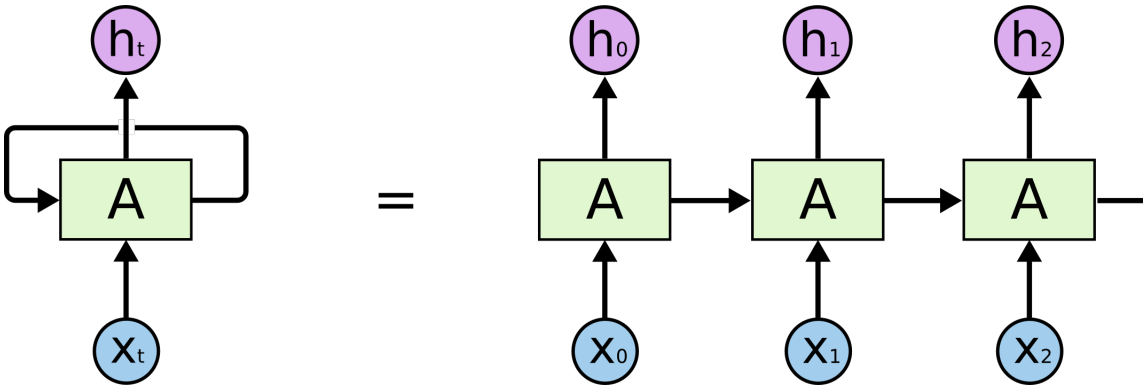

Рисунок 3.5 – Принцип работы рекуррентной нейронной сети

Моделирование памяти в нейронной сети подобным образом вводит новое измерение в описание процесса её работы — время. Пусть нейронная сеть получает на вход последовательность данных, например, текст пословно или слово побуквенно. Тогда каждый следующий элемент этой последовательности поступает на нейрон в новый условный момент времени. К этому моменту в нейроне уже есть накопленный с начала поступления информации опыт.

Итак, выбрав компоненты для построения архитектуры, необходимо определиться с составом нейронной сети. Выбрав слишком глубокую нейронную сеть, необходимо будет иметь большой объем данных, а так же большие вычислительные мощности. Так как объем данных, которыми мы обладаем не очень велик, а, главное, объем мощностей ограничен, то стоит

ограничить количество слоев. Исследования показывают, что количество слоев, большее 6 требует значительных вычислительных мощностей для обучения. Меньшее количество слоев может привести к недостаточному уровню абстракции.

Для того, чтобы извлечение жанра было успешным, необходимо классифицировать композицию по признакам, которые имеют высокий уровень абстракции. Поэтому количество слоев сверточной нейронной сети должно превышать количество слоев рекуррентной нейронной сети. Так же использование совсем небольшого количества слоев для рекуррентной нейронной сети может повлиять отрицательно на классификацию временных признаков, которые были извлечены сверточной нейронной сеть, поэтому количество слоев рекуррентной нейронной сети должно быть не меньше 2.

Определившись с ограничениями, которые мы накладываем на нейронную сеть, можем определить её точный состав. Вход нейронной сети будет представлен четырьмя слоями сверточной нейронной сети, которым будет поступать на вход матрица мел-спектральных коэффициентов. Слои реккурентной нейронной сети будут обрабатывать признаки, которые выделила сверточная нейронная сеть, и будет реализована в виде двух слоев.

Полученная нейронная сеть состоит из слоев: первые четыре – сверточные, последующие два – рекуррентные (см. рисунок 3.6).

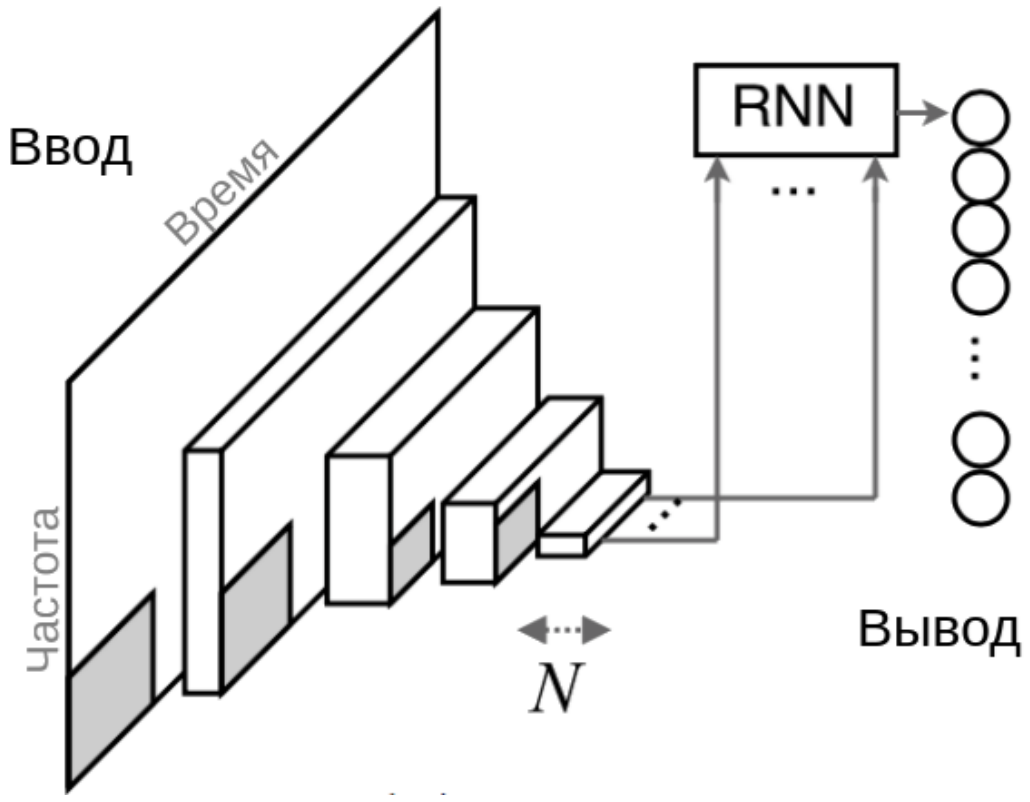

Рисунок 3.6 – Полученная структура нейронной сети

Так же на скорость обучения влияет выбор оптимизатора. Стохастический градиентный спуск, который стал классическим для применения на нейронных сетях, слишком неоптимизирован. Другим алгоритмом оптимизации, присутствующим в сообществе нейронных сетей, является Adam. Adam можно рассматривать как обобщение AdaGrad (AdaGrad - это Adam с определенным выбором параметров). Правило обновления для Adam определяется на основе оценки первого (среднего) и второго необработанного момента исторических градиентов (в рамках последнего временного окна через экспоненциальную скользящую среднюю). Эти статистические данные корректируются на каждой итерации (исправление необходимо из-за выбора инициализации).

Алгоритм оптимизации Adam:

а) вычислить градиент в текущее время;

б) обновить предварительную оценку первого момента;

в) обновить смещенную оценку второго необработанного момента;

г) вычислить оценку первого момента с исправлением смещения;

д) вычислить скорректированную с учетом смещения оценку второго необработанного момента;

е) обновить параметры.

Так же мы можем столкнуться с проблемой переобучения. Она является очень важным подводным камнем глубокого обучения. Отрицательный эффект переобучения заметно проявляется на сетях, подобных той, что мы собираемся построить, а значит, необходимо найти способ защититься от этого явления прежде, чем мы пойдем дальше. К счастью, существует очень простой метод, который мы и применим.

Переобучение – это излишне точное соответствие нейронной сети конкретному набору обучающих примеров, при котором сеть теряет способность к обобщению (см. рисунок 3.7). Другими словами, наша модель могла выучить обучающее множество (вместе с шумом, который в нем присутствует), но она не смогла распознать скрытые процессы, которые это множество породили.

У глубоких сверточных нейронных сетей масса разнообразных параметров, особенно это касается полносвязных слоев. Переобучение может проявить себя в следующей форме: если у нас недостаточно обучающих примеров, маленькая группа нейронов может стать ответственной за большинство вычислений, а остальные нейроны станут избыточны; или наоборот, некоторые нейроны могут нанести ущерб производительности, при этом другие нейроны из их слоя не будут заниматься ничем, кроме исправления их ошибок.

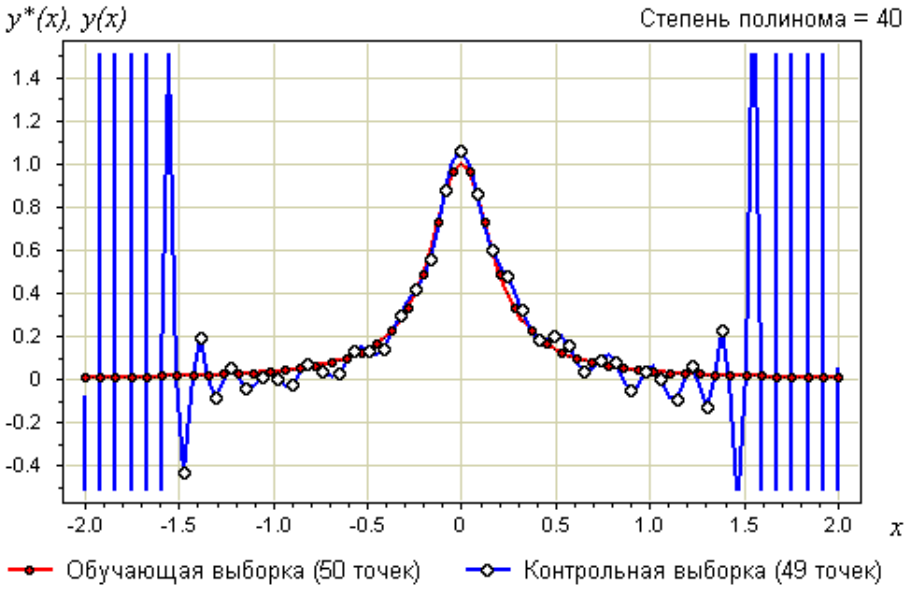

Рисунок 3.7 – Пример переобучения для полиномиальной функции

Чтобы помочь нашей сети не утратить способности к обобщению в этих обстоятельствах, мы вводим приемы регуляризации: вместо сокращения количества параметров, мы накладываем ограничения на параметры модели во время обучения, не позволяя нейронам изучать шум обучающих данных. Dropout с параметром p за одну итерацию обучения проходит по всем нейронам определенного слоя и с вероятностью p полностью исключает их из сети на время итерации (см. рисунок 3.8). Это заставит сеть обрабатывать ошибки и не полагаться на существование определенного нейрона (или группы нейронов), а полагаться на согласованное значение нейронов внутри одного слоя. Этот довольно простой метод позволяет эффективно бороться с проблемой переобучения, без необходимости использовать другие методы.

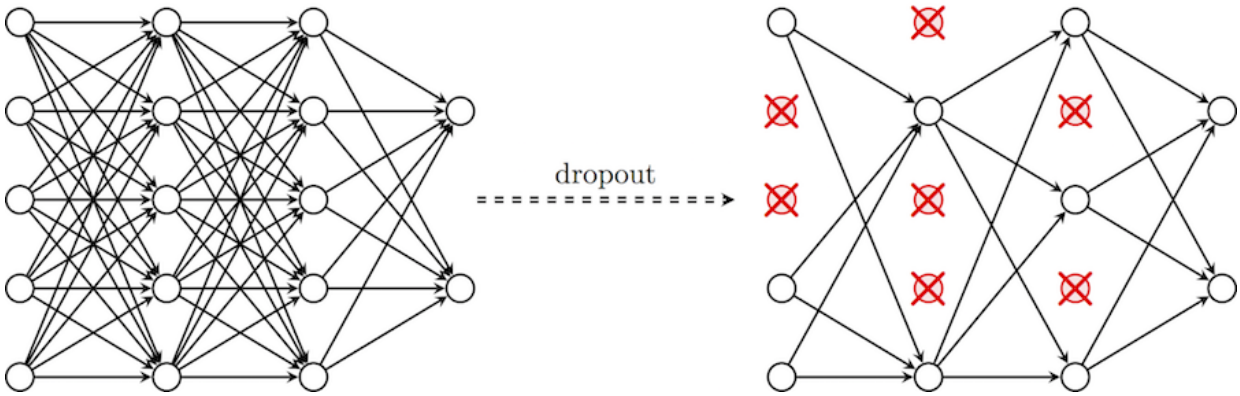

Рисунок 3.8 – Пример использования dropout

4 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

4.1 Essentia

Essentia – это библиотека C ++ с открытым исходным кодом, обладающая привязками Python, предназначенная для аудиоанализа. Она выпускается под лицензией Affero GPLv3 и также доступна под собственной лицензией по запросу. Библиотека содержит обширную коллекцию многократно используемых алгоритмов, реализующих функциональность ввода и вывода звука, стандартные блоки цифровой обработки сигналов, статистическую характеристику данных и большой набор спектральных, временных, тональных музыкальных обработчиков.

Essentia – это не фреймворк, а скорее набор алгоритмов, завернутый в библиотеку. Он сконструирован с упором на надежность, производительность и оптимальность предоставляемых алгоритмов, включая скорость вычислений и использование памяти, а также простоту использования. Поток анализа определяется и реализуется пользователем, в то время как Essentia заботится о деталях реализации используемых алгоритмов. Существует специальный режим потоковой передачи, в котором возможно соединять алгоритмы и запускать их автоматически, вместо того, чтобы явно указывать порядок выполнения с преимуществом менее стандартного кода и меньшим потреблением памяти. Ряд примеров предоставляется библиотекой, однако их не следует рассматривать как единственный правильный способ делать вещи. Большая часть алгоритмов Essentia хорошо подходит для приложений реального времени.

Предоставляемые функциональные возможности легко расширяемы и позволяют проводить как исследовательские эксперименты, так и разработку крупномасштабных промышленных приложений.

4.2 Python

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты – динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных

вычислений и удобные высокоуровневые структуры данных. Код в Python организовывается в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. Есть реализации интерпретаторов для JVM (с возможностью компиляции), MSIL (с возможностью компиляции), LLVM и других. Проект PyPy предлагает реализацию Python с использованием JIT-компиляции, которая значительно увеличивает скорость выполнения Python-программ.

Python – активно развивающийся язык программирования, новые версии (с добавлением/изменением языковых свойств) выходят примерно раз в два с половиной года. Вследствие этого и некоторых других причин на Python отсутствуют стандарт ANSI, ISO или другие официальные стандарты, их роль выполняет CPython

4.3 TensorFlow

TensorFlow – это библиотека программного обеспечения с открытым исходным кодом для машинного обучения по целому ряду задач и разработанная Google для удовлетворения своих потребностей в системах, способных создавать и обучать нейронные сети для обнаружения и расшифровки паттернов и корреляций, аналогичных обучению и рассуждениях, которые используют люди. В настоящее время он используется для исследований и производства продуктов Google, часто заменяя роль предшественника с закрытым исходным кодом, DistBelief. TensorFlow была первоначально разработана командой Google Brain для внутреннего использования Google, прежде чем выпустить ее под лицензией с открытым исходным кодом Apache 2.0 9 ноября 2015 года.

Начиная с 2011 года, Google Brain построила DistBelief как проприетарную систему машинного обучения, основанную на глубоком обучении нейронных сетей. Его использование быстро развивалось в различных компаниях родительской компании Alphabet как в исследовательских, так и в коммерческих целях. Google назначил нескольких компьютерных ученых, включая Джеффа Дина, для упрощения и реорганизации базы кода DistBelief в более быструю и более надежную библиотеку прикладного уровня, которая стала TensorFlow. В 2009 году команда во главе с Джеффри

Хинтоном реализовала обобщенное обратное распространение и другие улучшения, которые позволили создать нейронные сети с существенно большей точностью, например, на 25% сократить ошибки распознавания речи.

TensorFlow – это система машинного обучения второго поколения Google Brain, выпущенная как программное обеспечение с открытым исходным кодом 9 ноября 2015 года. Хотя эталонная реализация выполняется на отдельных устройствах, TensorFlow может работать на нескольких процессорах и графических процессорах (с дополнительными расширениями CUDA для универсальных вычислений на графических процессорах). TensorFlow доступен на 64-битных Linux, macOS и мобильных вычислительных платформах, включая Android и iOS.

Вычисления TensorFlow выражаются в виде графов прохождения данных с состоянием. Название TensorFlow происходит от операций, которые такие нейронные сети выполняют с многомерными массивами данных. Эти многомерные массивы называются тензорами. В июне 2016 года Джефф Дин из Google заявил, что 1500 репозиториев на GitHub упомянули TensorFlow, из которых только 5 пренадлежали Google.

В мае 2016 года Google объявил о своем тензорном процессоре (TPU) – специализированной ASIC, разработанной специально для машинного обучения и предназначенной для TensorFlow. TPU представляет собой программируемый ускоритель ИИ, предназначенный для обеспечения высокой пропускной способности арифметики с низкой точностью (например, 8-разрядной) и ориентированной на использование или запуск моделей, а не на их обучение. Google объявила, что они уже более года работают с моделями машинного обучения на основе TPU в своих центрах обработки данных, и обнаружили, что TPU обеспечивает на порядок более оптимизированную производительность на ватт для машинного обучения.

4.4 Keras

Keras – это высокоуровневый API нейронных сетей, написанный на Python и способный работать поверх TensorFlow или Theano. Он был разработан с упором на быстрое проведение экспериментов. Способность идти от идеи к результату с наименьшей задержкой является ключевой особенностью данной библиотеки.

Особенности Keras:

а) возможность легко и быстро создавать прототипы (благодаря удобству пользователя, модульности и расширяемости);

б) поддержка как сверточных нейронных сетей, так и рекуррентных нейронных сетей, а также их комбинаций;

в) работа без проблем на процессоре и графическом процессоре.

Руководящие принципы:

а) удобство для пользователя (Keras – это API, предназначенный для людей, а не для машин, это ставит удобство использования в центр внимания и обеспечивает ясную обратную связь из-за пользовательской ошибки);

б) модульность (под моделью понимается последовательность или граф автономных, полностью конфигурируемых модулей, которые могут быть подключены вместе с минимальными ограничениями);

в) легкая расширяемость (новые модули легко добавлять, как новые классы и функции, а существующие модули предоставляют достаточно примеров, чтобы иметь возможность легко создавать новые модули);

г) работа с Python (нет отдельных конфигурационных файлов моделей в декларативном формате, модели описаны в коде Python, который является компактным, более легким для отладки и позволяет легко расширяться).

4.5 Flask

Flask – это небольшой фреймворк, написанный на языке Python, с весьма большим сообществом и множеством модулей на все случаи жизни. В отличии от, скажем, Django, Flask не навязывает определенное решение той или иной задачи. Вместо этого, он предлагает использовать различные сторонние или собственные решения по вашему усмотрению.

Одним из проектных решений во Flask является то, что простые задачи должны быть простыми; они не должны занимать много кода, и это не должно ограничивать вас. Поэтому было сделано несколько вариантов дизайна, некоторые люди могут посчитать это удивительным и необщепринятым. Например, Flask использует локальные треды внутри объектов, так что вы не должны передавать объекты в пределах одного запроса от функции к функции, оставаясь в безопасном треде. Хотя этот очень простой подход и позволяет сэкономить много времени, это также может вызвать некоторые проблемы для очень больших приложений, поскольку изменения в этих локальных объектах-потоках может произойти где угодно в том же потоке. Для того, чтобы решить эти проблемы, разработчики не скрывают от вас локальные объекты-потоки, вместо этого охватывают их и предоставляем вам много инструментов, чтобы сделать работу с ними настолько приятным насколько это возможно.

4.6 Gunicorn

Gunicorn – это HTTP-сервер Python WSGI для UNIX. Это модель рабочих процессов, перенесенная из проекта Ruby Unicorn. Сервер Gunicorn в целом совместим с различными веб-фреймворками, прост в использовании, не нагружает ресурсы сервера и является достаточно быстрым.

Особенности:

а) поддерживает WSGI, web2py, Django и Paster;

б) автоматическое управление рабочими процессами;

в) простая конфигурация Python;

г) несколько рабочих конфигураций;

д) различные серверные триггеры для расширяемости;

е) совместимость с Python версий 2.6 и выше, а также Python версий 3.2 и выше.

4.7 Supervisor

Supervisor – это система клиент-сервер, которая позволяет своим пользователям управлять несколькими процессами в UNIX-подобных операционных системах.

Часто сложно получить точный статус (запущен или не запущен) в процессах в UNIX. Файлы с идентификаторами процессов часто не соответствуют действительности. Supervisord запускает процессы как подпроцессы, поэтому он всегда знает истинный статус своих подпроцессов, и предоставляет возможность узнать статусы для его дочерних процессов.

Supervisor предоставляет вам одно место для запуска, остановки и мониторинга процессов. Процессами можно управлять индивидуально или в группах. Вы можете настроить Supervisor для предоставления локальной или удаленной командной строки и веб-интерфейса.

Supervisor запускает свои подпроцессы через fork/exec, а подпроцессы не запускает как демоны. Операционная система сразу же сигнализирует Supervisor, когда процесс завершается, в отличие от некоторых решений, которые полагаются на неточные файлы идентификаторов процессов и периодический опрос процессов, чтобы перезапустить завершившиеся процессы.

4.8 Docker

Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной

системы. Позволяет поместить приложение со всем его окружением и зависимостями в контейнер, который может быть перенесен на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами. Изначально использовал возможности LXC, с 2015 года применял собственную библиотеку, абстрагирующую виртуализационные возможности ядра Linux – libcontainer. С появлением Open Container Initiative начался переход от монолитной к модульной архитектуре.

Цели использования Docker:

а) абстрагирование хост-системы от контейнеризованных приложений;

б) простота масштабирования;

в) простота управления зависимостями и версиями приложения;

г) чрезвычайно легкие, изолированные среды выполнения;

д) совместно используемые слои;

е) возможность компоновки и предсказуемость.

Приложения, реализующие этот подход к проектированию, должны иметь следующие характеристики:

а) они не должны полагаться на особенности хост-системы;

б) каждый компонент должен предоставлять консистентный API, который пользователи могут использовать для доступа к сервису;

в) каждый сервис должен принимать во внимание переменные окружения в процессе первоначальной настройки;

г) данные приложения должны храниться вне контейнера на примонтированных томах или в отдельных контейнерах с данными.

5 ТРЕНИРОВКА МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ

В нашем случае необходимо построить классификатор, который будет создавать соответствие между музыкальной композицией и определенным музыкальным жанром. Количество жанров будет ограничено, так как набор жанров известен заранее. Для такого типа классификаторов наилучшим образом подходит обучение с учителем.

Обучение с учителем — один из способов машинного обучения, в ходе которого испытуемая система принудительно обучается с помощью примеров «стимул-реакция». С точки зрения кибернетики, является одним из видов кибернетического эксперимента. Между входами и эталонными выходами (стимул-реакция) может существовать некоторая зависимость, но она неизвестна. Известна только конечная совокупность прецедентов — пар «стимул-реакция», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость (построить модель отношений стимул-реакция, пригодных для прогнозирования), то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов, так же как и в обучении на примерах, может вводиться функционал качества.

Для того, чтобы обучить модель, необходимо обладать достаточным количеством данных для обучения. В качестве данных используются заранее размеченные композиции, т.е. композиции с известным жанром. Композиции, на которых будет обучаться модель, являются композициями типичных представителей или основателей музыкальных направлений. Всего для обучения используется около одинадцати тысяч записей.

Для того, чтобы оптимизировать обучение, композиции были подвергнуты первичной обработке. Первичная обработка состояла в том, чтобы преобразовать композиции в векторы, которые будут поступать на вход модели. Для обработки использовался метод, который преобразовывает композицию в набор мел-частотных кепстральных коэффициентов, описанный в части 3.2.

Для обучения выборка разбивается на несколько фрагментов. Выборка разделяется на непересекающиеся подмножества элементов, на которых будет происходить обучение. В нашем случае мы будем разбивать выборку на 3 подмножества:

а) обучающее;
б) валидирующее;
в) тестирующее.

Элементы обучающего подмножества будут использоваться для того, чтобы проверять реакцию модели и, соответственно, корректировать веса так, чтобы повышать точность классификации.

Валидирующее подмножество так же используется для обучения модели. Это случаи, на которых не проверялась реакция модели. Они необходимы для того, чтобы модель не переобучилась определять жанр только для элементов обучающего подмножества. Это внесет поправку в алгоритм обучения, которая зависит только от значений метрики точности, что позволит использовать эти данные для валидации в дальнейшем.

Элементы тестирующего множества необходимы только для того, чтобы проверить итоговую точность построенных моделью предсказаний.

Обучение разделено на эпохи. Каждая эпоха состоит из нескольких шагов:

а) обучение на обучающей выборке в прямом порядке;
б) обучение на обучающей выборке в обратном порядке;
в) проверка на переобучение с помощью валидирующей выборки;
г) перемешивание элементов в обучающей выборке;
д) перемешивание элементов в валидирующей выборке.

Перемешивание элементов необходимо для того, чтобы исключить случайный эффект, когда модель начнет переобучаться из-за того, что в нее будут поступать циклические данные.

Количество эпох для тренировки модели вычисляется эмпирически. Модели построены так, чтобы иметь возможность их дообучить при необходимости.

Для обучения моделей необходимы ресурсы, которые значительно превышают ресурсы, которые необходимы для работы моделей в обычном режиме.

6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

6.1 Установка

Для того, чтобы установить и подготовить сервис для использования, необходимо установить Docker на используемую систему.

Далее мы опишем процесс установки Docker на системы семейства Linux/Debian.

Для начала обновим базу данных пакетов:

```
sudo apt-get update
```

Теперь установим Docker. Добавьте ключ GPG официального репозитория Docker в вашу систему:

```
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net
   :80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

Добавим репозиторий Docker в список источников пакетов утилиты APT:

```
sudo apt-add-repository 'deb https://apt.dockerproject.org/repo
    ubuntu-xenial main'
```

Обновим базу данных пакетов информацией о пакетах Docker из вновь добавленного репозитория:

```
sudo apt-get update
```

Далее, наконец-то, установим Docker:

```
sudo apt-get install -y docker-engine
```

После завершения выполнения этой команды Docker должен быть установлен, демон запущен, и процесс должен запускаться при загрузке системы. Проверим, что процесс запущен:

```
sudo systemctl status docker
```

Вывод должен быть похож на представленный ниже, сервис должен быть запущен и активен:

```
docker.service - Docker Application Container Engine
Loaded: loaded (/lib/systemd/system/docker.service; enabled;
   vendor preset: enabled)
Active: active (running) since Sun 2016-05-01 06:53:52 CDT; 1
   weeks 3 days ago
  Docs: https://docs.docker.com
Main PID: 749 (docker)
```

При установке Docker мы получаем не только сервис (демон) Docker, но и утилиту командной строки docker или клиент Docker.

Далее необходимо загрузить контейнер на машину:

```
docker pull dshmatkov/genrify
```

Перед тем как загружать контейнер, необходимо авторизоваться у автора контейнера, так как он является приватным.

После нужно запустить контейнер:

```
docker run -p 127.0.0.1:80:8000 dshmatkov/genrify
```

Порт контейнера 8000 будет привязан к порту 80 по адресу 127.0.0.1 на локальной системе.

Далее можно приступить к использованию сервиса.

6.2 Использование

Для того, чтобы получать характеристики музыкальных композиций, необходимо пересылать файлы аудиозаписей в теле HTTP-запроса в контейнер на порт 8000.

В результате вы получите ответ от сервиса, который будет содержать искомые характеристики.

Для того, чтобы определить формат, в котором вы хотите получать ответ от сервиса, необходимо использовать заголовок Content-Type.

Для того, чтобы получать ответы в формате XML - необходимо выставить значение этого заголовка в application/xml.

Для того, чтобы получать ответы в формать JSON - необходимо выставить значение этого заголовка в application/json.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ВНЕДРЕНИЯ ПРОГРАММНОГО СРЕДСТВА

7.1 Введение и исходные данные

Целью дипломной работы является создание программного средства, которое позволит автоматизировать сбор характеристик музыкальных композиций. С помощью него значительно сокращается время сбора характеристик для дальнейшего анализа аудиокомпозиций. Автоматизация процессов сокращает риск возникновения человеческой ошибки, что повышает точность классификации и приводит к генерации более точных рекомендаций и увеличению аудитории медиасервисов.

Целью данного технико-экономического обоснования является определение экономической эффективности создания данного программного продукта и его дальнейшего применения. Экономическая эффективность sрассчитывается у разработчика и пользователя.

Программный комплекс относится к 1-й группе сложности. Для оценки экономической эффективности разработанного программного средства проводится расчет сметы затрат и цены программного продукта, а также прибыли от продажи одной системы (программы). Расчеты выполнены на основе методического пособия [3].

7.2 Расчет сметы затрат и цены программного продукта

Исходные данные, которые будут использоваться при расчете сметы затрат, представлены в таблице 7.1.

Объем программного средства определяется путем подбора аналогов на основании классификации типов программного средства, каталога функций, которые постоянно обновляются и утверждаются в установленном порядке. На основании информации и функциях разрабатываемого программного средства по каталогу функций определяется объем функций. Объем программного средства определяется на основе нормативных данных, приведенных в таблице 7.2.

Объем программного средства вычисляется по формуле:

$$V_O = \sum_{i=1}^{n} V_i \,, \tag{7.1}$$

Таблица 7.1 – Исходные данные

Наименование показателя	Буквенные обозначения	Единицы измерения	Количество
Коэффициент новизны	*Н*	единиц	0.9
Группа сложности		единиц	1
Дополнительный коэффициент сложности	*СЛ*	единиц	0.18
Поправочный коэффициент, учитывающий использование типовых программ	*Т*	единиц	0.8
Установленная плановая продолжительность разработки	*Р*	лет	1
Продолжительность рабочего дня	*Ч*	Ч	8
Тарифная ставка первого разряда	*М1*	руб	265
Коэффициент премирования	*П*	единиц	1.2
Норматив дополнительной заработной платы	*Д*	%	20
Отчисления в фонд социальной защиты населения	*СЗ*	%	34
Отчисления в Белгосстрах	*НС*	%	0.6
Расходы на научных командировки	*НКИ*	%	30
Прочие прямые расходы	*ЗИ*	%	20

где V_i – объем отдельной функции ПО; n – общее число функций; $V_{\text{О}}$ – общий объем ПС.

$V_{\text{О}} = 32710$ (строк исходного кода).

Исходя из режима работы в реальном времени, а также обеспечения существенного распараллеливания вычислений и реализации особо сложных инженерных и научных расчетов, применяется коэффициент $K_{\text{С}}$ к объему ПО, который определяется по формуле:

$$K_{\text{С}} = 1 + \sum_{i=1}^{n} K_i \,, \tag{7.2}$$

Таблица 7.2 – Характеристика функций и их объем

№ функции	Наименование (содержание) функции	Объём функции
101	Организация ввода информации	150
102	Контроль, предварительная обработка и ввод информации	450
111	Управление вводом/выводом	2400
201	Генерация структуры базы данных	4300
203	Формирование баз данных	2180
204	Обработка наборов и записей базы данных	2670
207	Манипулирование данными	9550
305	Обработка файлов	720
507	Обеспечение интерфейса между компонентами	970
701	Математическая статистика и прогнозирование	9320
Итого:		32 710

где K_i – коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;

$$K_{\text{С}} = 1 + 0.06 + 0.06 + 0.06 = 1.18$$

С учетом дополнительного коэффициента сложности $\text{К}_{\text{СЛ}}$ рассчитывается общая трудоемкость ПС по формуле:

$$\text{Т}_{\text{О}} = \text{Т}_{\text{Н}} + \text{К}_{\text{С}}\ , \tag{7.3}$$

где $\text{Т}_{\text{О}}$ – общая трудоемкость; $\text{Т}_{\text{Н}}$ – нормативная трудоемкость ПС; $\text{К}_{\text{С}}$ – дополнительный коэффициент сложности ПС. Нормативная трудоемкость ПО ($\text{Т}_{\text{Н}}$) – 1087 чел./дн.; коэффициент сложности ($\text{К}_{\text{С}}$) – 1.12; коэффициент, учитывающий степень использования при разработке ПО стандартных модулей ($\text{К}_{\text{Т}}$) – 0.8; коэффициент новизны разрабатываемого ПО ($\text{К}_{\text{Н}}$) – 0.9.

Общая трудоемкость вычисляется по формуле:

$$\text{Т}_{\text{О}} = \text{Т}_{\text{Н}} \cdot \text{К}_{\text{С}} \cdot \text{К}_{\text{Т}} \cdot \text{К}_{\text{Н}} \tag{7.4}$$

$$\text{Т}_{\text{О}} = 830 \cdot 1.18 \cdot 0.8 \cdot 0.9 = 705(\text{чел./дн.})$$

На основе уточненной трудоемкости разработки ПС и установленного периода разработки, общая плановая численность разработчиком равна:

$$Ч_Р = \frac{Т_О}{Т_Р \cdot Ф_{ЭФ}}\ , \tag{7.5}$$

где $Ф_{ЭФ}$ – эффективный фонд времени работы одного работника в течение года (дн.); $Т_О$ – общая трудоемкость разработки проекта (чел./дн.); $Т_Р$ – срок разработки проекта (лет).

Эффективный фонд времени работы одного работника ($Ф_{ЭФ}$) рассчитывается по формуле:

$$Ф_{ЭФ} = Д_Г - Д_П - Д_В - Д_О\ , \tag{7.6}$$

где $Д_Г$ – количество дней в году; $Д_П$ – количество праздничных дней в году; $Д_В$ – количество выходных дней в году; $Д_О$ – количество дней отпуска.

$Ф_{ЭФ} = 238$ дней в году.

Срок разработки установлен 12 месяцев ($Т_Р = 1$ год).

$$Ч_Р = \frac{705}{1.0 \cdot 238} \approx 2.96 \approx 3$$

Реализацией проекта занимались 3 человека. В соответствии с численностью и выполняемым функциями устанавливается штатное расписание группы специалистов-разработчиков.

Расчет основной заработной платы осуществляется в следующей последовательности. Определим месячные ($Т_М$) и часовые ($Т_Ч$) тарифные ставки начальника отдела (тарифный разряд – 15; тарифный коэффициент – 3.48), инженера-программиста 1-й категории (тарифный разряд – 14; тарифный коэффициент – 3.25). Месячный тарифный оклад ($Т_{МО}$) определяется путем умножения действующей месячной тарифной ставки 1-го разряда ($Т_{М1}$) на тарифный коэффициент ($Т_{Ki}$), соответствующий установленному тарифному разряду специалиста:

$$Т_{МОi} = Т_{М1} \cdot Т_{Ki} \tag{7.7}$$

Часовая тарифная ставка рассчитывается путем деления месячной тарифной ставки на установленный при сорокачасовой рабочей неделе в восьмичасовом рабочем дне фонд рабочего времени – 168 часов:

$$Т_Ч = Т_М / 168\ , \tag{7.8}$$

где $Т_Ч$ – часовая тарифная ставка (руб); $Т_М$ – месячная тарифная ставка(руб).

Таблица 7.3 – Расчет утонченной трудоемкости ПС и численности исполнителей по стадиям

Показатели	Стадии					Итого
	ТЗ	ЭП	ТП	РП	ВН	
1 Коэффициенты удельныхвесов трудоемкости стадии разработки ПО (d)	0,10	0,08	0,09	0,58	0,15	0,10
2 Коэффициент сложности ПО ($К_С$)	0,18	0,18	0,18	0,18	0,18	
3 Коэффициент, учитывающий использование стандартных модулей				0,8		
4 Коэффициент, учитывающий новизну ПО ($К_Н$)	0,9	0,9	0,9	0,9	0,9	
5 Численность исполнителей, чел. ($Ч_И$)	3	3	3	3	3	3
6 Сроки разработки, лет						1

Месячная и часовая тарифные ставки начальника отдела:

$$Т_м = 265 \cdot 3.48 = 768.5(\text{руб})$$

$$Т_ч = 768.5/168 = 4.58(\text{руб})$$

Месячная и часовая тарифные ставки инженера-программиста 1-й категории равны соответственно:

$$Т_м = 265 \cdot 3.25 = 717.7(\text{руб})$$

$$Т_ч = 717.7/168 = 4.27(\text{руб})$$

Расчет месячных и почасовых тарифных ставок сведен в таблицу 7.4.

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле:

$$З_О = \sum_{i=1}^{n} З_{Ci} \cdot Ф_{Pi} \cdot К_i \,, \qquad (7.9)$$

Таблица 7.4 – Штатное расписание группы разработчиков

Должность	Количество ставок	Тарифный разряд	Тарифный коэффициент	Месячная тарифная ставка (руб)	Часовая тарифная ставка(руб)
Начальник отдела (ведущий инженер программист)	1,0	15	3,48	768,5	4,57
Инженер-программист 1-й категории	1,0	14	3,25	717,7	4,27
Инженер-программист 1-й категории	1,0	14	3,25	717,7	4,27

где n – количество исполнителей, занятых разработкой конкретного ПО; $З_{Ci}$ – среднедневная заработная плата i-го исполнителя (д.е.); $Ф_{Pi}$ – плановый фонд рабочего времени i-го исполнителя (дн.); К – коэффициент премирования (1.2).

$$4.57 \cdot 8 \cdot 238 \cdot 1.2 + 4.27 \cdot 8 \cdot 238 \cdot 1.2 + 4.27 \cdot 8 \cdot 238 \cdot 1.2 = 31736.4(\text{руб})$$

Дополнительная заработная плата исполнителей проекта определяется по формуле:

$$З_Д = \frac{З_О \cdot Н_Д}{100} \tag{7.10}$$

$$\frac{31736.4 \cdot 20}{100} = 6347.28(\text{руб})$$

Отчисления в фонд социальной защиты населения и на обязательное страхование ($З_С$) определяются в соответствии с действующими законодательными актами по формуле:

$$З_{СЗ} = \frac{(З_О + З_Д) \cdot Н_{СЗ}}{100}, \tag{7.11}$$

где $Н_{СЗ}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34% + 0.6%).

$$\frac{(31736.4 + 6347.28) \cdot 34.6}{100} = 13176.95(\text{руб})$$

Расходы по статье «Машинное время» ($Р_М$) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле:

$$Р_М = Ц_М \cdot Т_Ч \cdot Т_Р \tag{7.12}$$

$$Р_М = 2665.6(\text{руб})$$

Затраты по статье «Накладные расходы» ($Р_Н$), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды ($Р_Н$), определяются по формуле:

$$Р_Н = \frac{З_{СЗ} \cdot Н_{РН}}{100}, \tag{7.13}$$

$$\frac{40578.05 \cdot 100}{100} = 40578.05(\text{руб})$$

Полная себестоимость:

$$С_П = З_О + З_Д + Р_М + З_{СЗ} + Р_Н, \tag{7.14}$$

$$С_П = 85662.63(\text{руб})$$

Прогнозируемая прибыль ПС рассчитывается по формуле:

$$П_{ПС} = \frac{С_П \cdot У_Р}{100}, \tag{7.15}$$

$$\frac{85662 \cdot 25}{100} = 21415.66(\text{руб})$$

Прогнозируемая отпускная цена ПС вычисляется по формуле:

$$Ц_П = С_П + П_{ПС}, \tag{7.16}$$

$$Ц_П = 107078(\text{руб})$$

Налог на добавленную стоимость (НДС):

$$\text{НДС} = \frac{\text{Ц}_\text{П} \cdot \text{Н}_\text{ДС}}{100}, \tag{7.17}$$

где $\text{Н}_\text{ДС}$ – норматив НДС (%).

$$\text{НДС} = 19274.09(\text{руб})$$

Отпускная цена:

$$\text{Ц}_\text{О} = \text{Ц}_\text{П} + \text{НДС} \tag{7.18}$$

$$\text{Ц}_\text{О} = 126352(\text{руб})$$

Кроме того, организация-разработчик осуществляет затраты на сопровождение ПС ($\text{Р}_\text{С}$):

$$\text{Р}_\text{С} = \frac{\text{С}_\text{П} \cdot \text{Н}_\text{С}}{100}, \tag{7.19}$$

где $\text{Н}_\text{С}$ – норматив расходов на сопровождение и адаптацию (20%).

$$\text{Р}_\text{С} = 17132.53(\text{руб})$$

7.3 Оценка экономической эффективности применения ПС у пользователя

Таблица 7.5 – Исходные данные для расчета экономического эффекта

Наименование показателей	Обозначения	Единицы измерения	Значения показателя в базовом варианте	Значения показателя в новом варианте	Наименование источника информации
1	2	3	4	5	6

Продолжение таблицы 7.5

1	2	3	4	5	6
1 Капитальные вложения, включая затраты пользователя на приобретение	$К_{ПР}$	руб.		126352	Договор заказчика с разработчиком
2 Затраты на сопровождение ПО	$К_{С}$	руб.		17132.53	Договор заказчика с разработчиком
3 Время простоя сервиса, обусловленное ПО, в день	$П_1$,$П_2$	мин	120	10	Расчетные данные пользователя и паспорт ПО
4 Стоимость одного часа простоя	$С_{П}$	руб.	30.1	30.1	Расчетные данные пользователя и паспорт ПО
5 Среднемесячная ЗП одного программиста	$З_{СМ}$	руб.	881.57	881.57	Расчетные данные пользователя
6 Коэффициент начислений на зарплату	$К_{Н}$		1.2	1.2	Расчитывается по данным пользователя

Продолжение таблицы 7.5

1	2	3	4	5	6
7 Среднемесячное количество рабочих дней	$Д_Р$	день	21	21	Принято для расчета
8 Количество типовых задач, решаемых за год	$З_{х1}$,$З_{х2}$	задача	1800	1800	План пользователя
9 Объем выполняемых работ	A_1,A_2	задача	1800	1800	План пользователя
10 Средняя трудоемкость работ на задачу	$Т_{С1}$,$Т_{С2}$	Человеко-часов	8	0.5	Расчитывается по данным пользователя
11 Количество часов работы в день	$Т_Ч$	ч	8	8	Принято для рассчета
12 Ставка налога на прибыль	$Н_П$	%		18	

Экономия затрат на заработную плату в расчете на 1 задачу ($С_{зе}$):

$$С_{ЗЕ} = \frac{З_{СМ} \cdot (Т_{с1} - Т_{с2})}{Д_Р \cdot ч}, \qquad (7.20)$$

где $З_{см}$ – среднемесячная заработная плата одного программиста (руб.); $Т_{с1}$, $Т_{с2}$ – снижение трудоемкости работ в расчете на 1 задачу (человеко-часов); $Т_ч$ – количество часов работы в день (ч); $Д_р$ – среднемесячное количество рабочих дней.

$$С_{ЗЕ} = 39.36(\text{руб.})$$

Экономия заработной платы при использовании нового ПО (тыс руб.):

$$С_З = С_{ЗЕ} \cdot А_2 , \quad (7.21)$$

где $С_з$ – экономия заработной платы; $А_2$ – количество типовых задач, решаемых за год (задач).

$$С_З = 70840(\text{руб.})$$

Экономия с учетом начисления на зарплату ($С_н$):

$$С_Н = С_З \cdot К_{НЗ} \quad (7.22)$$

$$С_Н = 85008(\text{руб.})$$

Экономия за счет сокращения простоев сервиса ($С_с$) рассчитывается по формуле:

$$С_С = \frac{(П_1 - П_2) \cdot Д_{РГ} \cdot С_П}{60} , \quad (7.23)$$

где $Д_{РГ}$ – плановый фонд работы сервиса (дней).

$$С_С = 1158(\text{руб.})$$

Общая готовая экономия текущих затрат, связанных с использованием нового ПО ($С_О$), рассчитывается по формуле:

$$С_О = С_Н + С_С \quad (7.24)$$

$$С_О = 86167(\text{руб.})$$

Внедрение нового ПО позволит пользователю сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении ($П_ч$), которая определяется по формуле:

$$П_ч = С_О - \frac{С_О \cdot Н_П}{100} \quad (7.25)$$

$$П_ч = 70657(\text{руб.})$$

В процессе использования нового ПО чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2017-й год) путем умножения результатов и затрат за каждый год на коэффициент

дисконтирования . В данном примере используются коэффициенты: 2017 г. – 1, 2018-й – 0.8696, 2019-й – 0.7561, 2020 г. – 0.6575. Все рассчитанные данные экономического эффекта сводятся в таблицу 7.4.

Таблица 7.6 – Расчет экономического эффекта от использования нового программного средства

Показатели	Единицы измерения	2017 г.	2018 г.	2019 г.	2020 г.
1	2	3	4	5	6
Результаты					
Прирост прибыли за счет экономии затрат ($П_ч$)	руб.		70 657	70 657	70 657
То же с учетом фактора времени	руб.		61 143,32	53 423,75	46 456,97
Затраты					
Приобретение ПО ($К_{пр}$)	руб.	126 352,38			
Сопровождение ($К_с$)	руб.	17 132,53			
Всего затрат	руб.	143 484,91			
Экономический эффект					
Превышение результата над затратами	руб.	−143 484,91	61 443,32	53 423,75	46 456,97

Продолжение таблицы 7.6

1	2	3	4	5	6
То же с нарастающим итогом	руб.	−143 484,91	−82 041,6	−28 617,84	17 839,14
Коэффициент приведения	единицы	1	0,8696	0,7561	0,6575

7.4 Вывод по технико-экономическому обоснованию

Исходя из расчётов затрат на разработку, можно сделать вывод, что для разработки сервиса для извлечения характеристик музыкальной композиции понадобится один год, в разработке участвуют три программиста. Полная себестоимость составляет 85662.63 руб., отпускная цена программного продукта с уровнем рентабельности 25% - 126352 руб. Также организация-разработчик должна осуществить затраты на сопровождение ПС в размере 17132.53 руб.

Исходя из таблицы расчёта экономического эффекта использования сервиса для извлечения характеристик музыкальной композиции, можно прийти к выводу, что данный продукт позволяет существенно сократить затраты на анализ музыкальных композиций. Положительный экономический эффект заключается в значительном уменьшении трудоёмкости извлечения характеристик музыкальных композиций. Продукт является экономически выгодным благодаря тому, что его окупаемость достигается по истечении трех лет. В конце этого срока чистая прибыль составит 17839.14 рублей.

ЗАКЛЮЧЕНИЕ

Данный дипломный проект относится к области извлечения музыкальной информации. Был проведен анализ продуктов, которые являются аналогами реализуемого проекта, а так же предметной области в целом. По результатам анализа был сделан вывод, что на данный момент в этой области ведуться активные исследования и разработки, так же были оценены аналоги, выявлены их достоинства и недостатки.

На основании проведенного анализа предметной области были выдвинуты требования к программному средству. В качестве технолгогий разработки были выбраны наиболее современные средства, применяемые для решения подобных задач. Спроектированное средство показало высокую точность классификации композиций по жанрам, что показывает эффективность выбранного подхода. Расширение распространения автоматизации на извлечение музыкальных характеристик высокого уровня позволяет сделать вывод о целесообразности разработки данного программного средства. Это было подтверждено так же в ходе выполнения экономического обоснования.

В итоге было разработано пограммное средство, позволяющее автоматизированно извлекать характеристики музыкальных композиций высокого уровня.

В дальнейшем планируется увеличивать точность извлечения характеристик. Так же планируется применить подходы, которые не требуют тренировки и способны самостоятельно обучаться извлекать характеристики в процессе работы. Кроме извлекаемых, существуют другие характеристики композиций, поддержку которых планируется внедрить в данное программное средство.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] van der Maaten, L.J.P. Visualizing High-Dimensional Data Using t-SNE / L.J.P. van der Maaten, G.E. Hinton. — Journal of Machine Learning Research, 2008.

[2] van den Oord, Aaron. Deep content-based music recommendation / Aaron van den Oord, Sander Dieleman, Benjamin Schrauwen. — Electronics and Information Systems department (ELIS), Ghent University. — http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf.

[3] Палицын, В.А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. — Мн : БГУИР, 2006. — 76 с.

[4] Croll, Alistar. Music science / Alistar Croll. — O'Reilly, 2015.

[5] Webster, Courtney. Embeding analytics in modern applications / Courtney Webster. — O'Reilly, 2016.

[6] Overton, Jerry. Going pro in data science / Jerry Overton. — O'Reilly, 2016.

[7] Brice, Richard. Music engineering / Richard Brice. — Elsevier / Newnes, 2001.

[8] Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and TensorFlow / Aurélien Géron. — O'Reilly Media, 2017.

[9] Julian, David. Designing Machine Learning Systems with Python / David Julian. — Packt Publishing, 2016.

[10] Gulli, Antonio. Deep Learning with Keras / Antonio Gulli, Sujit Pal. — Packt Publishing, 2017.

[11] Buduma, Nikhil. Fundamentals of Deep Learning / Nikhil Buduma. — O'Reilly Media, 2017.

[12] Fries, Bruce. Digital Audio Essentials / Bruce Fries, Marty Fries. — O'Reilly Media, 2005.

[13] Mueller, John Paul. Machine Learning For Dummies / John Paul Mueller, Luca Massaron. — Wiley / For Dummies, 2016.

[14] Parker, Michael. Digital Signal Processing / Michael Parker. — Elsevier / Newnes, 2010.

[15] Ganchev, T. Comparative evaluation of various MFCC implementations on the speaker verification task / T. Ganchev, N. Fakotakis, G. Kokkinakis. — University of Patras, 2005.

[16] Линдсей, П. Переработка информации у человека / П. Линдсей, Д. Норман. — Мир, 1974.

[17] ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. — М. : Издательство стандартов, 1991. — Введ. 01.01.1992.

[18] ГОСТ 7.0-99 Информационно-библиотечная деятельность, библиография. Термины и определения. — М. : Межгосударственный совет по стандартизации, метрологии и сертификации, 1999.

[19] Доманов, А. Т. Стандарт предприятия. Дипломные проекты (работы). Общие требования. / А. Т. Доманов, Н. И. Сорока. — Минск : БГУИР, 2010.

[20] ISO/IEC 2382-1:1993, Information technology — Vocabulary — Part 1: Fundamental terms [Электронный ресурс]. — Электронные данные. — Режим доступа: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:2382:ed-1:v1:en. — Дата доступа: 11.12.16.

[21] Волосевич, А. А. Архитектура программного обеспечения: Курс лекций для студентов специальности 1-40 01 03 Информатика и технологии программирования / А. А. Волосевич. — Минск : БГУИР, 2013.

[22] Garlan, David. An Introduction to Software Architecture / David Garlan, Mary Shaw / Ed. by V Ambriola, G Tortora. — New Jersey : World Scientific Publishing Company, 1994. — Vol. I.

[23] Пещенко, Е.А. Производственный календарь на 2017 год [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.mintrud.gov.by/system/extensions/spaw/uploads/files/Kommetarij-2017-RV.pdf. — Дата доступа: 06.04.17.

[24] Белстат. О начисленной средней заработной плате работников в феврале 2017 г. [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.belstat.gov.by/ofitsialnaya-statistika/solialnaya-sfera/trud/operativnaya-informatsiya_8/o-nachislennoi-srednei-zarabotnoi-plate-rabotnikov/o-nachislennoy-sredney-zarabotnoy-plate-rabotnikov-v-fevrale-2017-nbsp-g/. — Дата доступа: 07.04.17.

ПРИЛОЖЕНИЕ А
(обязательное)
Фрагменты исходного кода

```
from keras.models import Model, K
from keras.layers import Input, Dense
from keras.layers import Dropout, Reshape, Permute
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D,
   ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import ELU
from keras.layers.recurrent import GRU


GENRES = [
    'blues',
    'chillout',
    'classical',
    'country',
    'drumnbass',
    'electro',
    'folk',
    'house',
    'jazz',
    'metal',
    'pop',
    'reggae',
    'rock',
    'techno',
    'trance',
]


def MRCC(include_top=False):
    if K.image_dim_ordering() == 'th':
        input_shape = (1, 96, 1366)
    else:
        input_shape = (96, 1366, 1)
    melgram_input = Input(shape=input_shape)

    if K.image_dim_ordering() == 'th':
        channel_axis = 1
        freq_axis = 2
        time_axis = 3
    else:
        channel_axis = 3
```

```
    freq_axis = 1
    time_axis = 2

# Input block
x = ZeroPadding2D(padding=(0, 37))(melgram_input)
x = BatchNormalization(axis=time_axis, name='bn_0_freq')(x)

# Conv block 1
x = Convolution2D(64, 3, 3, border_mode='same', name='conv1
    ')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn1
    ')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='
    pool1')(x)
x = Dropout(0.1, name='dropout1')(x)

# Conv block 2
x = Convolution2D(128, 3, 3, border_mode='same', name='
    conv2')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn2
    ')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(3, 3), strides=(3, 3), name='
    pool2')(x)
x = Dropout(0.1, name='dropout2')(x)

# Conv block 3
x = Convolution2D(128, 3, 3, border_mode='same', name='
    conv3')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn3
    ')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(4, 4), strides=(4, 4), name='
    pool3')(x)
x = Dropout(0.1, name='dropout3')(x)

# Conv block 4
x = Convolution2D(128, 3, 3, border_mode='same', name='
    conv4')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn4
    ')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(4, 4), strides=(4, 4), name='
    pool4')(x)
x = Dropout(0.1, name='dropout4')(x)
```

```
    # reshaping
    if K.image_dim_ordering() == 'th':
        x = Permute((3, 1, 2))(x)
    x = Reshape((15, 128))(x)

    # GRU block 1, 2, output
    x = GRU(32, return_sequences=True, name='gru1')(x)
    x = GRU(32, return_sequences=False, name='gru2')(x)

    if include_top:
        x = Dense(15, activation='sigmoid', name='output')(x)

    x = Dropout(0.3)(x)

    model = Model(melgram_input, x)

    return model
```

Листинг 7.1 – Модель для определения жанра композиции

```
from __future__ import unicode_literals, absolute_import

import numpy
import logging
from essentia import standard as esd

logger = logging.getLogger(__name__)


def get_mfcc_bands(item_path):
    logger.info('Getting mfcc bands from %s', item_path)
    # loading record
    mono_loader = esd.MonoLoader(filename=item_path)
    load_result = mono_loader.compute()

    # cutting on frames
    frame_size = len(load_result) / 1366
    frames = [
        frame for frame in
        esd.FrameGenerator(
            load_result,
            frameSize=frame_size * 2,
            hopSize=frame_size
        )
    ][:1366]

    # windowing frames
    window = esd.Windowing(type='hann')
```

```
    windowed_frames = [window(frame) for frame in frames]

    # spectrum calculation
    spectrum = esd.Spectrum()
    specs = [spectrum(frame) for frame in windowed_frames]

    # calculation mfcc
    mfcc = esd.MFCC(inputSize=20000, numberCoefficients=96,
       numberBands=96)
    mfcc_coeffs = []
    mfcc_bands = []
    for spectrum in specs:
        mfcc_band, mfcc_coeff = mfcc(spectrum)
        mfcc_coeffs.append(mfcc_coeff)
        mfcc_bands.append(mfcc_band)

    bands_array = numpy.array(mfcc_bands)

    return bands_array
```

Листинг 7.2 – Код для извлечение мел-частотных кепстральных коэффициентов

```
from __future__ import absolute_import, unicode_literals

import logging
import numpy
import pickle
import sys
import csv
from diplom.models.CRNN.model import MRCC
from diplom.models.CRNN.utils import category_to_ar

logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logger = logging.getLogger(__name__)
csv.field_size_limit(sys.maxsize)


def process_row(row):
    inp = pickle.loads(row[0])
    genre = row[1]

    return numpy.expand_dims(inp.T, axis=2), numpy.array(
       category_to_ar(genre))


def get_data_generator(filename, row_count=100):
    """
```

```
        :type filename: unicode | str
        :type row_count: int
        :rtype: tuple
        """
        with open(filename, 'r') as f:
            while True:
                f.seek(0)
                reader = csv.reader(f)
                a = []
                b = []
                for row in reader:
                    x, y = process_row(row)
                    a.append(x)
                    b.append(y)
                    if len(a) >= row_count:
                        xx = numpy.array(a)
                        yy = numpy.array(b)
                        a = []
                        b = []
                        yield xx, yy
                        xx = None
                        yy = None

                if a:
                    xx = numpy.array(a)
                    yy = numpy.array(b)
                    a = []
                    b = []
                    yield xx, yy
                    xx = None
                    yy = None


def get_train_gen():
    generator = get_data_generator('calculated_mfccs/fit.csv')
    samples = 64
    return generator, samples


def get_validate_gen():
    generator = get_data_generator('calculated_mfccs/validate.
       csv')
    samples = 15
    return generator, samples


def get_evaluate_gen():
```

```
    generator = get_data_generator('calculated_mfccs/test.csv')
    samples = 21
    return generator, samples


def run():
    logger.info('Creating model')
    model = MRCC(include_top=True)

    logger.info('Compiling model')
    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy'],
    )

    logger.info('Fit model')
    train_generator, train_samples = get_train_gen()
    validate_generator, validate_samples = get_validate_gen()
    model.fit_generator(
        train_generator,
        epochs=50,
        samples_per_epoch=train_samples,
        verbose=2,
        validation_data=validate_generator,
        validation_steps=validate_samples,
    )

    logger.info('Saving model')
    model.save('saved_model', overwrite=True)
    model.save_weights('saved_weights', overwrite=True)

    evaluate_generator, evaludate_samples = get_evaluate_gen()
    result = model.evaluate_generator(
        evaluate_generator,
        steps=evaludate_samples
    )
    print result


if __name__ == '__main__':
    run()
```

Листинг 7.3 – Код для тренировки модели

```
import csv
import logging
import os
```

```
import random
import sys

logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logger = logging.getLogger(__name__)

csv.field_size_limit(sys.maxsize)


def mix_batches(batch1, batch2):
    logger.info('Reading datasets')
    with open(batch1, 'r') as in1:
        with open(batch2, 'r') as in2:
            reader1 = csv.reader(in1)
            reader2 = csv.reader(in2)
            gen1 = reader1.__iter__()
            gen2 = reader2.__iter__()

            new_batch = []

            try:
                while True:
                    value = next(gen1)
                    new_batch.append(value)

                    value = next(gen2)
                    new_batch.append(value)
            except StopIteration:
                pass

            try:
                while True:
                    value = next(gen1)
                    new_batch.append(value)
            except StopIteration:
                pass


            try:
                while True:
                    value = next(gen2)
                    new_batch.append(value)
            except StopIteration:
                pass

    logger.info('Writing datasets')
```

```
    with open('{}.tmp'.format(batch1), 'w') as out1:
        with open('{}.tmp'.format(batch2), 'w') as out2:
            writer1 = csv.writer(out1)
            writer2 = csv.writer(out2)

            for i, item in enumerate(new_batch):
                if i % 2 == 0:
                    writer1.writerow(item)
                else:
                    writer2.writerow(item)

    logger.info('Moving files')
    os.remove(batch1)
    os.remove(batch2)

    os.rename('{}.tmp'.format(batch1), batch1)
    os.rename('{}.tmp'.format(batch2), batch2)


def mix_randomly(batch1, batch2):

    full_batch = []

    logger.info('Reading datasets')
    with open(batch1, 'r') as inf:
        reader = csv.reader(inf)
        for row in reader:
            full_batch.append(row)

    with open(batch2, 'r') as inf:
        reader = csv.reader(inf)
        for row in reader:
            full_batch.append(row)

    logger.info('Shuffle dataset')
    random.shuffle(full_batch)

    logger.info('Writing dataset')
    with open('{}.tmp'.format(batch1), 'w') as in1:
        with open('{}.tmp'.format(batch2), 'w') as in2:
            writer1 = csv.writer(in1)
            writer2 = csv.writer(in2)

            for i, item in enumerate(full_batch):
                if i % 2 == 0:
                    writer1.writerow(item)
                else:
```

```
                    writer2.writerow(item)

    logger.info('Moving files')
    os.remove(batch1)
    os.remove(batch2)

    os.rename('{}.tmp'.format(batch1), batch1)
    os.rename('{}.tmp'.format(batch2), batch2)


def run(src, dest):

    dir_content = sorted(os.listdir(src))

    logger.info('Making mixing things')

    pairs = []
    for i in dir_content:
        for j in dir_content:
            if i == j: continue
            pairs.append((i, j))

    # logger.info('Mixing')
    # for i in xrange(3):
    #     logger.info('# %s mix', i)
    #     for batch1, batch2 in pairs:
    #         logger.info('Mixing %s, %s', batch1, batch2)
    #         batch1_path = os.path.join(src, batch1)
    #         batch2_path = os.path.join(src, batch2)
    #         mix_batches(batch1_path, batch2_path)

    logger.info('Random mix')
    for batch1, batch2 in pairs:
        logger.info('Mixing %s, %s', batch1, batch2)
        batch1_path = os.path.join(src, batch1)
        batch2_path = os.path.join(src, batch2)
        mix_randomly(batch1_path, batch2_path)

    logger.info('Merging dataset')
    with open(dest, 'w') as outf:
        writer = csv.writer(outf)
        for batch in dir_content:
            logger.info('Merge %s', batch)
            batch_path = os.path.join(src, batch)

            with open(batch_path, 'r') as inf:
                reader = csv.reader(inf)
```

```
                for row in reader:
                    writer.writerow(row)


if __name__ == '__main__':
    src = sys.argv[1]
    dest = sys.argv[2]

    run(src, dest)
```

Листинг 7.4 – Код для перемешивания большой выборки данных