МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

КУРСОВАЯ РАБОТА

Сравнительный анализ нейросетей на основе задачи классификации музыкальных произведений по жанрам

студента 3 курса 341 группы
направления 02.03.02 Математическое обеспечение и администрирование информационных систем
факультета компьютерных наук и информационных технологий
Локтева Ильи Константиновича

Научный руководитель
ст. пр. Е.Е.Лапшева
Зав. кафедрой
к.ф.-м.н., доцент М.В.Огнева

Саратов 2021

СОДЕРЖАНИЕ

Список сокращений и специальных терминов

CNN — сверточная нейронная сеть,

RNN — рекуррентная нейронная сеть,

LSTM — нейронная сеть с архитектурой «длительная краткосрочная память»,

DFT — дискретное преобразование Фурье,

FFT — быстрое преобразование Фурье,

STFT — кратковременное преобразование Фурье,

MFCC — мел-кепстральные коэффициенты,

Sample rate — частота дискретизации, Гц.

ВВЕДЕНИЕ

На сегодняшний день неоспорим тот факт, что количество общедоступных данных стремительно возрастает с каждым годом. Ввиду этого, классификация данных становится одной из задач, для которых автоматизация процесса является необходимой, поскольку ручная обработка данных становится практически невозможной. Музыкальная индустрия не является исключением, поскольку автоматизация процесса способна существенно улучшить организацию звуковых данных и процесс доступа к ним конечного пользователя.

Нейронные сети являются мощным инструментом машинного обучения и могут быть использованы для обработки общих признаков данных, не требуя для этого написания сложных компьютерных программ. Одной из задач, для которых могут быть использованы нейронные сети, является задача классификации песен по музыкальным жанрам. Однако, данная задача входит в число других задач по обработке звуковых данных, и результаты ее исследования могут быть полезны для решения проблем звукового анализа в целом. Этим объясняется актуальность данной работы.

Целью данной работы является сравнение нейронных сетей на основе задачи классификации звуковых данных по музыкальным жанрам. Исследование проводится с использованием нейросетей двух архитектур: сверточная нейронная сеть (CNN — convolutional neural network) и архитектуры «долгая краткосрочная память» (LSTM — long short-term memory). В работе рассматриваются особенности, преимущества и недостатки этих архитектур.

Для достижения поставленной цели следует выполнить следующие задачи:

- Введение в общую теорию нейронных сетей, знакомство с видами архитектур CNN и LSTM.
- Рассмотрение теории компьютерного представления звуковых данных.
- Формирование инструментария (язык программирования, вспомогательные программные библиотеки, данные для обработки).
- Подготовка данных.
- Построение моделей нейронных сетей выбранных архитектур.

- Сравнение работы моделей нейронных сетей.

1 Теория нейронных сетей и обработки звука, выбор инструментов для реализации задачи

1.1 Основные понятия из теории нейронных сетей

Нейронная сеть (англ. neural network), также искусственная нейронная сеть (ИНС), представляет собой математическую модель, построенную по принципу сильно упрощенной биологической нейронной сети — сети нервных клеток живого организма [1]. В машинном обучении нейронные сети являются очень мощным инструментом, способным решать многочисленные задачи, в которых требуется смоделировать поведение нервной системы человека. Главной особенностью нейронных сетей, составляющей схожесть с их биологическим аналогом, является способность обучаться и исправлять ошибки. С каждым годом нейросети все активнее используются в различных видах систем. К задачам, с которыми способны эффективно справляться нейронные сети, относятся: прогнозирование, принятие решений, распознавание образов, оптимизация, анализ данных и другие задачи.

Основой нейронной сети являются два объекта: нейроны и связи между ними (синапсы). Нейроны разделяются на группы, которые называются слоями. Различают входной, выходной, а также промежуточные (скрытые) слои, число которых может быть различным. Общая схема нейронной сети представлена на рисунке 1.

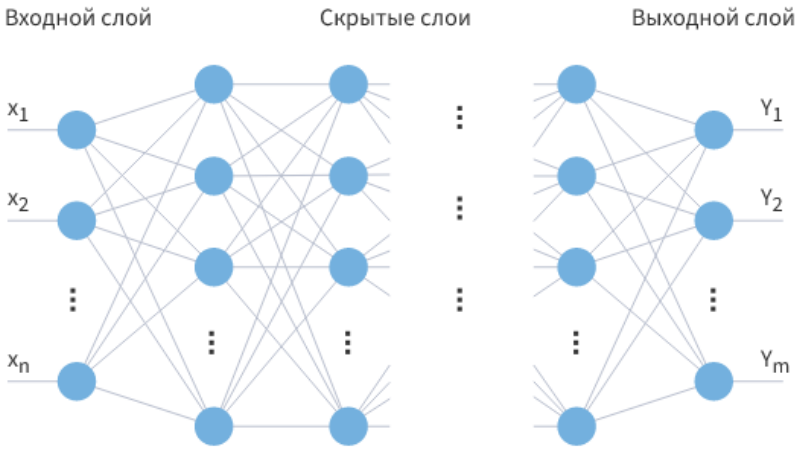


Рисунок 1 — Общая структура нейронной сети

Источник: Нейронная сеть (Neural network) [Электронный ресурс]. — URL: https://wiki.loginom.ru/articles/neural-network.html

Входной слой нейронов принимает входные данные, каждому нейрону присваивается определенное числовое значение. Этот слой является неизменным.

Связи (синапсы) между слоями имеют веса, которые изначально устанавливаются случайным образом. Веса связей могут изменяться, и будут изменяться на каждом шаге процесса, именуемого обучением нейронной сети.

Скрытый слой подразумевает, что каждый нейрон, составляющий его, является функцией, состоящей из двух компонент: сумматора и функции активации [2]. Если обозначить n нейронов предыдущего слоя x_i при $i \in [1, n]$, а w_i — веса связей, исходящих из каждого нейрона, то сумматор можно представить следующей формулой:

$$S = \sum_{i=1}^{n} x_i w_i$$

Результат работы сумматора подается в функцию активации (обозначим ее Y), подробное описание и виды которой представлены в пункте 1.1.1 текущей работы. Формула функции активации:

$$Y = f(S)$$

Выходной слой принимает в себя значения с последнего скрытого слоя.

Полный проход значений от входного слоя до выходного называется процессом прямого распространения ошибки (англ. forward propagation) [2].

Далее вычисляется ошибка, в основе подсчета которой лежит разница между полученным и ожидаемым значением. Для того, чтобы определить негативное влияние ошибки используется функция потерь (англ. loss function). «Потеря» — это количественная мера того, насколько критично получение ошибки определенного размера, на которую влияют негативные результаты, возникающие из-за неточного прогноза.

Существует множество видов функций потерь, но одними из самых распространенных являются Mean Squared Error (MSE), Root MSE и Arctan. Далее представлены формулы для вычисления ошибок каждого из указанных видов:

$$MSE = \frac{(i_1 - a_1)^2 + (i_2 - a_2)^2 + \cdots + (i_n - a_n)^2}{n}$$

$$MSE_R = \sqrt{\frac{(i_1 - a_1)^2 + (i_2 - a_2)^2 + \cdots + (i_n - a_n)^2}{n}}$$

$$Arctan = \frac{arctg^2(i_1 - a_1) + arctg^2(i_2 - a_2) + \cdots + arctg^2(i_n - a_n)}{n}$$

где i_n — ожидаемое значение, a_n — полученное значение, n — число нейронов выходного слоя.

Целью обучения нейронной сети является минимизация ошибки. Иными словами, это означает нахождение глобального минимума функции потерь, для чего используется метод градиентного спуска (англ. gradient descent). Метод градиентного спуска называют также алгоритмом обратного распространения

ошибки (англ. back propagation). График использования данного метода представлен на рисунке 2.

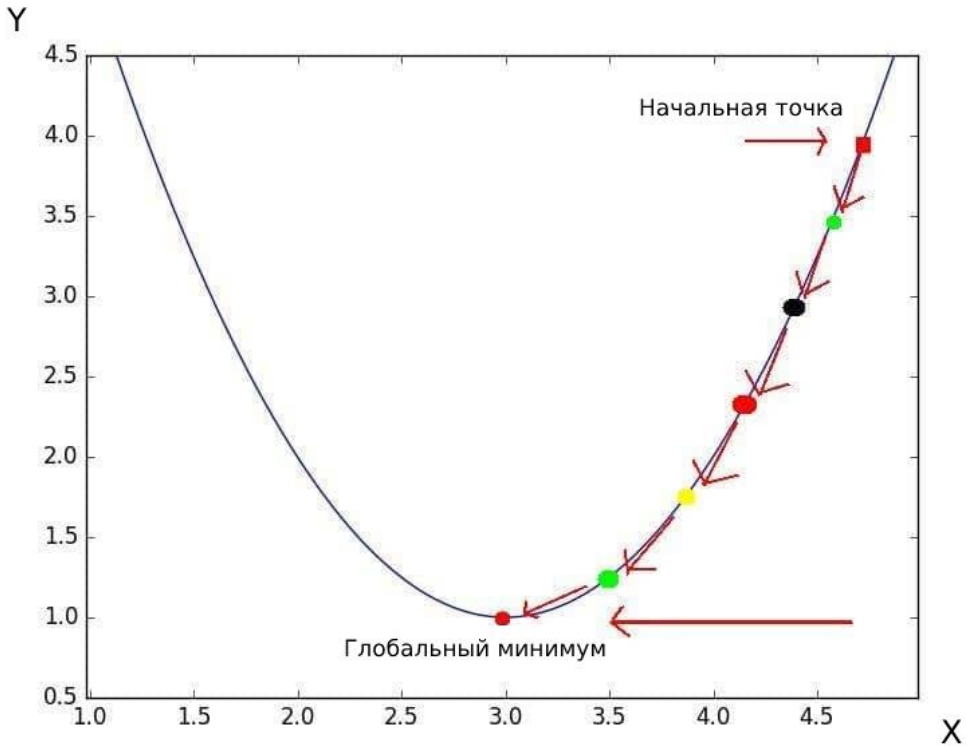


Рисунок 2 — График функции потерь

На графике в качестве начальной точки указано некоторое начальное значение функции потерь. Для того, чтобы осуществить «спуск» к глобальному минимуму, требуется повторное выполнение следующих шагов:

1) Вычисление частной производной ошибки по каждому из весов. Эта производная также называется градиентом и указывает направление движения в сторону глобального минимума. В точке глобального минимума градиент должен равняться нулю.

$$f' = \frac{d(Err)}{d(w_i)} \quad (1),$$

где *Err* — ошибка (разница между ожидаемым и полученным значением), w_i — значение веса при $i \in [1, n]$, где n — число весов.

2) Умножение полученной частной производной (1) на число η, называемое скоростью обучения (англ. learning rate). Это число является гипер-

параметром, от которого зависит то, насколько быстро будет достигнут глобальный минимум. В большинстве случаев подбирается экспериментально.

$$\eta * \frac{d(Err)}{d(w_i)} \quad (2)$$

3) Вычисление разницы между текущим значением веса и произведением (2).

$$w_{i+1} = w_i - \eta * \frac{d(Err)}{d(w_i)}$$

В результате применения пунктов 1-3 для каждого веса происходит обновление их значений [3 — с. 151].

Для обучения модели требуется повторно проводить тренировку сети на обновленных весах. Иными словами, происходит повторный запуск повторный запуск прямого и обратного распространения ошибки.

Эпоха (англ. epoch) — одно выполнение прямого и обратного распространения ошибки для всего датасета. Однако зачастую число данных велико, и невозможно обработать их одновременно. Для этого производится разбиение данных, с которым связано понятие батча.

Батч (англ. batch) — часть данных датасета, проходящих через нейронную сеть за один раз. Batch size — число данных, входящих в состав одного батча. Число батчей называют итерациями. Обработка всех батчей означает завершение одной эпохи [4].

Далее будут подробнее рассмотрены функции активации.

1.1.1 Функции активации

Функция активации — это один из основных элементов нейронной сети, который фактически определяет, какие нейроны будут активированы, и какая информация будет передаваться последующим слоям. Функция активации определяет выходное значение нейрона, принимая в качестве параметра результат работы сумматора [5].

Существует достаточно много видов функций активации. Ниже приведены некоторые из них.

Ступенчатая функция активации полагает, что нейрон считается активированным, если значение, поступающее на вход функции превышает некоторое пограничное значение. Например, если значение параметра больше 0, то результатом функции будет 1, иначе — 0. Использование данной функции возможен только для задач бинарной классификации. График ступенчатой функции активации приведен на рисунке 3.

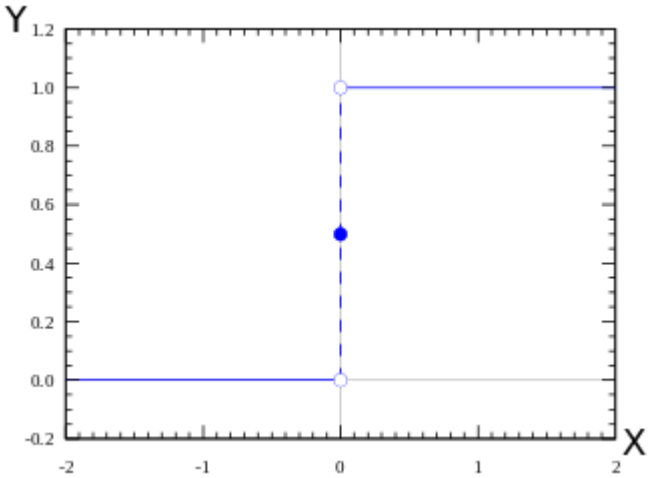


Рисунок 3 — график ступенчатой функции активации

Линейная функция активации представляет собой прямую линию, где значение функции пропорционально передаваемому аргументу. Преимущество данного вида функции активации заключается в том, что она уже способна выдавать диапазон значений. График функции представлен на рисунке 4.

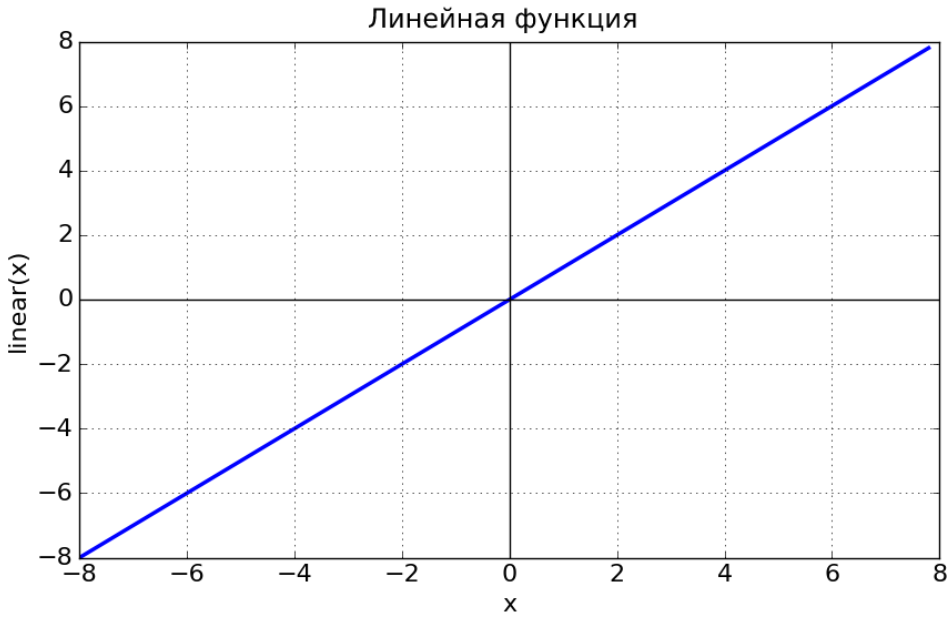


Рисунок 4 — График линейной функции активации

Линейная функция имеет свои недостатки: во-первых, к ней неприменим метод обратного распространения ошибки, поскольку производная для данной функции является константой. Во-вторых, выходные значения нейронов на каждом внутреннем слое будут представлять собой линейную комбинацию, т.е. значения на последнем слое зависят только от начальных значений, что делает бессмысленным использование многослойной сети [5].

Сигмоидная функция — гладкая нелинейная функция, которая является достаточно подходящей функцией для задач классификации. Она стремится привести значения к одной из сторон кривой, что позволяет находить четкие границы при предсказании. График сигмоиды представлен на рисунке 5.

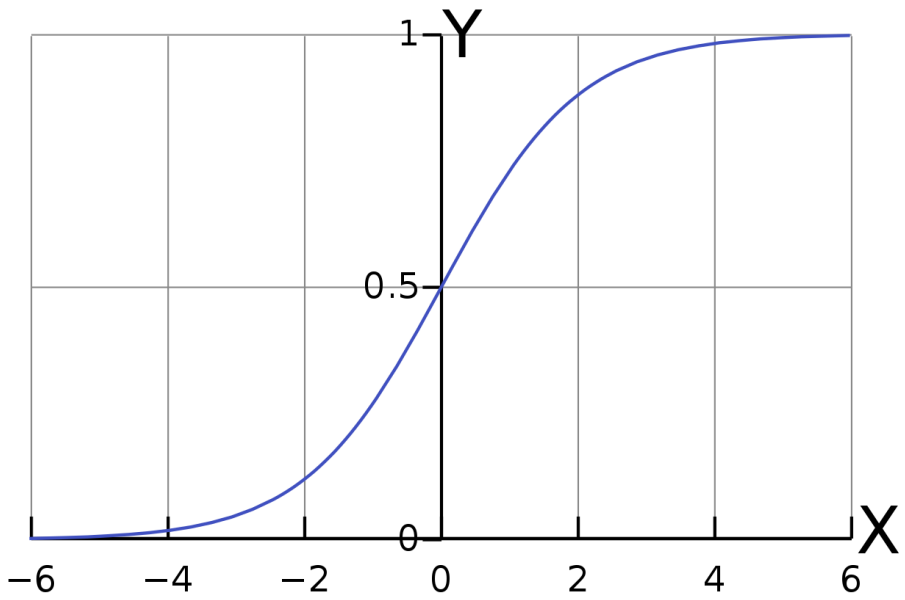


Рисунок 5 — График сигмоидной функции активации

Еще одним преимущество сигмоиды заключается в том, что она имеет фиксированный диапазон значений функции — [0,1], тогда как линейная функция изменяется в пределах (-∞, ∞). Данное сигмоиды очень полезно, так как позволяет избежать ошибок при наличии больших значений активации. На сегодняшний день сигмоида выделяется, как одна из самых часто используемых активационных функций при построении нейросетей.

Тем не менее, сигмоида имеет недостаток при приближении значений к почти горизонтальной части кривой на обеих сторонах. В этом случае значение градиента мало или исчезает. Нейросеть отказывается обучаться дальше или делает это крайне медленно

Гиперболический тангенс представляет собой скорректированную сигмоидную функцию. Данная функция также является нелинейной, хорошо подходит для комбинации слоев, и имеет диапазон значений (-1, 1). Это также решает проблему перегрузки фукнции при больших значениях. При этом важно отметить, что градиент функции гиперболического тангенса больше, чем у сигмоиды (производная круче). Выбор между сигмоидой и тангенсом зависит от требований к амплитуде градиента. Также как и сигмоиде, гиперболическому тангенсу свойственная проблема исчезновения градиента. Однако тангенс также

является очень популярной и используемой активационной функцией. График гиперболического тангенса представлен на рисунке 6.

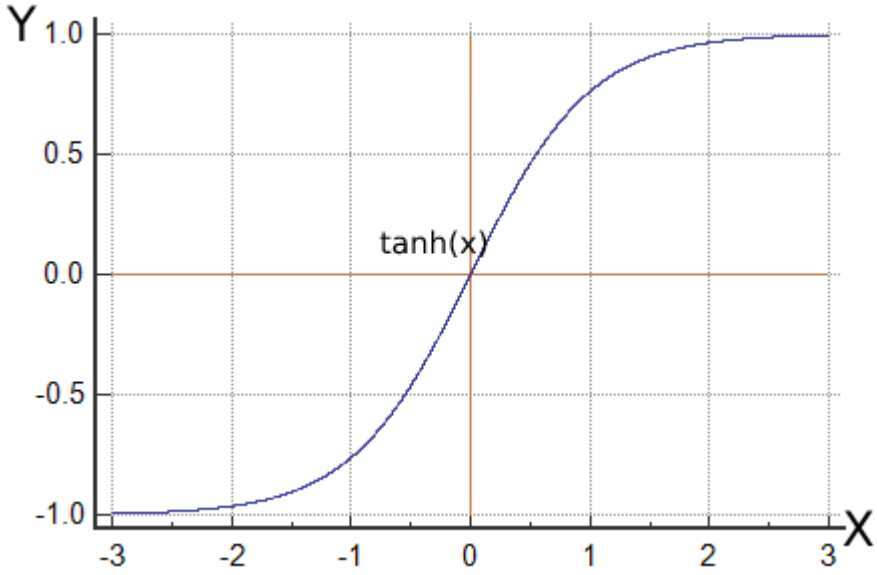


Рисунок 6 — График гиперболического тангенса

Функция ReLu представляет собой функцию, возвращающую переданное значение, если оно положительно, и 0 в противном случае. Данная функция является нелинейной, ее график представлен на рисунке 7.

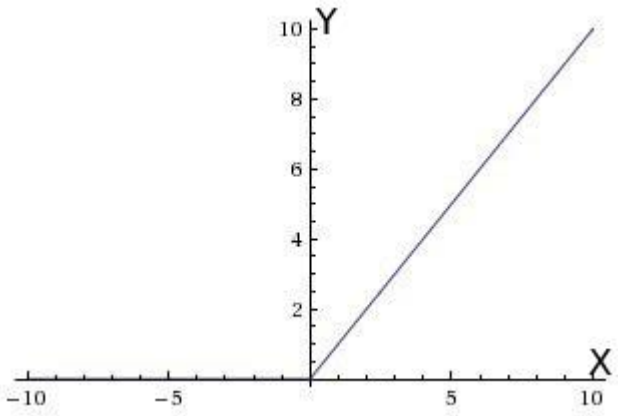


Рисунок 7 — график функции активации ReLu

Одной из особенностей функции ReLu является разреженность активации. Под этим понятием подразумевается то, что при большом числе (например, около 50%) активаций равных 0 активируется меньшее число нейронов, т.е. сеть становится легче.

Главный недостаток ReLu заключается в том, что для отрицательных значений функция ведет равна 0 и представляет собой горизонтальную линию, для которой градиент равен 0, а следовательно при обратном распространении ошибки веса не будут изменяться. Нейроны, находящиеся в таком состоянии не будут изменяться, несмотря на изменение ошибки или выбор других входных данных. Это явление называется проблемой умирающего ReLu. Одним из его решений может быть использование при отрицательных значений не прямой, а наклонной с малым угловым коэффициентом (около 0,01), что представляет функция Leaky ReLu [5].

Функция Softmax представляет собой обобщение логистической функции для многомерного случая. Функция имеет следующий вид:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_k}}$$

Функция Softmax преобразует вектор z размерности n в вектор σ той же размерности. Каждая координата σ представляет собой вещественное число в интервале [0;1].

При использовании в задачах классификации каждая координата соответствует вероятности принадлежности объекта каждому из классов. Функция Softmax часто используется на последнем слое глубоких сетей, где число нейронов равно числу классов.

1.1.2 Проблема переобучения нейронных сетей

Переобучение (англ. overfitting) представляет собой одну из проблем в реализации моделей нейронных сетей, при которой модель показывает хорошие результаты на обучающей выборке данных, но при этом плохо работает на непосредственно тестовых данных. Это происходит потому, что модель приспосабливается к обучающим примерам, но теряет способность к обобщению и не может классифицировать новые данные [6].

На рисунке 8 представлено сравнение ошибки при работе некоторой сети на тренировочных и тестовых данных:

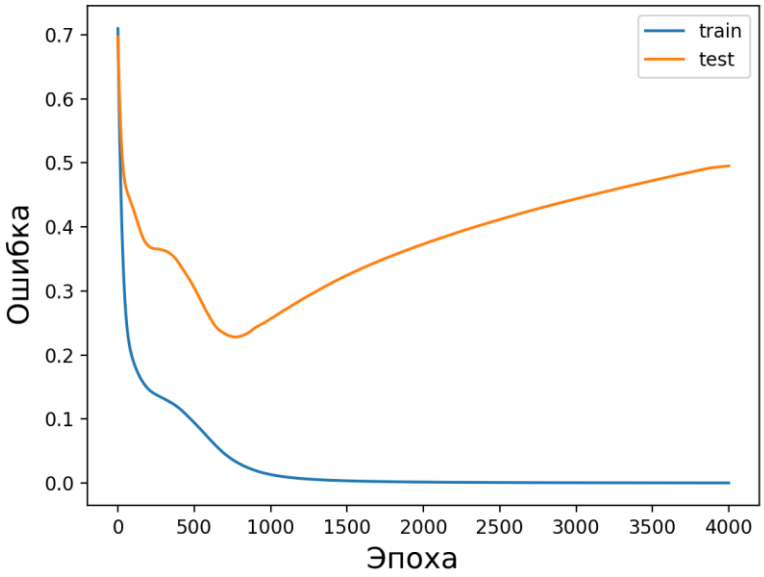


Рисунок 8 — Переобучение нейросети

На рисунке видно, что по мере обработки эпох для тренировочных данных сеть хорошо обучается и происходит уменьшение ошибки (кривая синего цвета). В то же время при обработке тестовых данных ошибка уменьшается до определенного момента, а затем начинает стабильно увеличиваться (кривая оранжевого цвета). В данном случае видна серьезная разница между результатом работы нейросети с тренировочными и новыми данными, что свидетельствует о переобучении.

Существует несколько распространенных способов решения проблемы переобучения нейросети:

Упрощение архитектуры предполагает исключение некоторых слоев сети. Этот метод может быть действенным, поскольку зачастую переобучение происходит из-за большого числа нейронов в сети. Данный способ может показать себя достаточно действенно на небольших нейросетях.

Аугментация данных (англ. data augmentation) представляет собой метод, при котором число данных в датасете искусственно увеличивается, т.е.

новые данные создаются из уже существующих. Например, для изображений это может быть сдвиг, поворот, смена цвета изображения и т.д. Для аудиоданных это может быть, например, фрагментация аудиозаписей на отрезки более короткой длины, изменение тональности аудиозаписи, растягивание записи по времени и т.д. Зачастую увеличение данных способствует более эффективной работе нейросети [6].

Ранняя остановка (англ. early stopping) — метод, при котором обучение сети заканчивается спустя некоторое число эпох до того момента, как нейросеть начнет ухудшать результат (увеличивать ошибку). Пример представлен на рисунке 9:

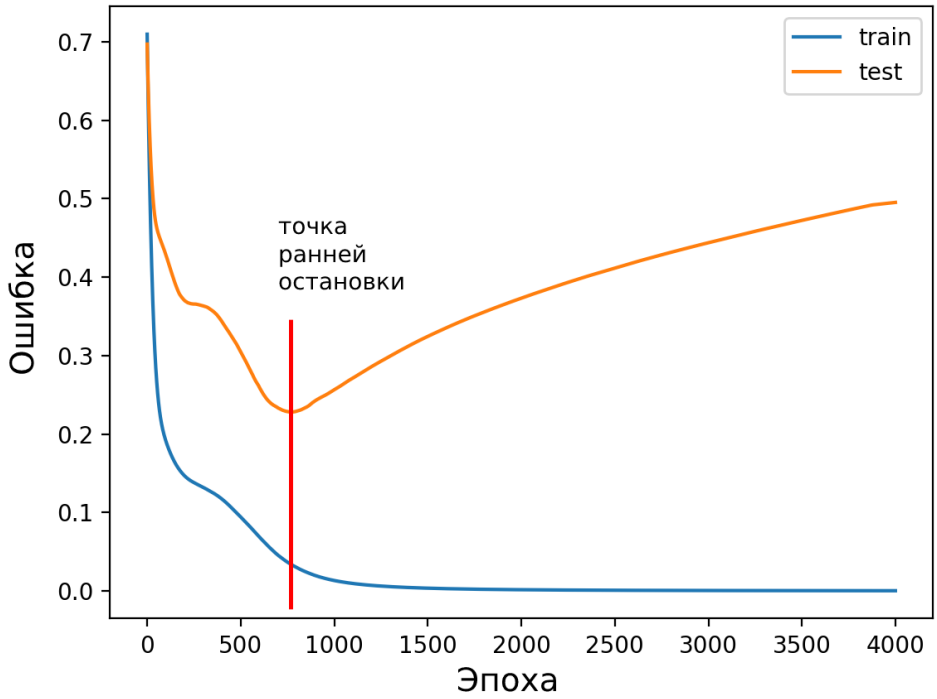


Рисунок 10 — Ранняя остановка обучения

Для реализации ранней остановки обычно используется некоторая метрика для отслеживания производительности и триггер остановки обучения, который будет сопровождать данную метрику и сработает при ее ухудшении. Метрикой производительности принято считать ошибку.

Дропаут (англ. dropout) представляет собой популярный метод решения переобучения сети, при котором часть нейронов в слоях деактивируется случайным образом на каждом этапе (эпохе) обучения. Вероятность деактивации нейрона определяется значением вероятность дропаута (англ. dropout probability). В основе метода лежит уменьшение числа нейронов в сети. Притом, нейроны, которые были неактивны в предыдущей эпохе, могут быть вновь активны в текущей. Принцип дропаута проиллюстрирован на рисунке 11:

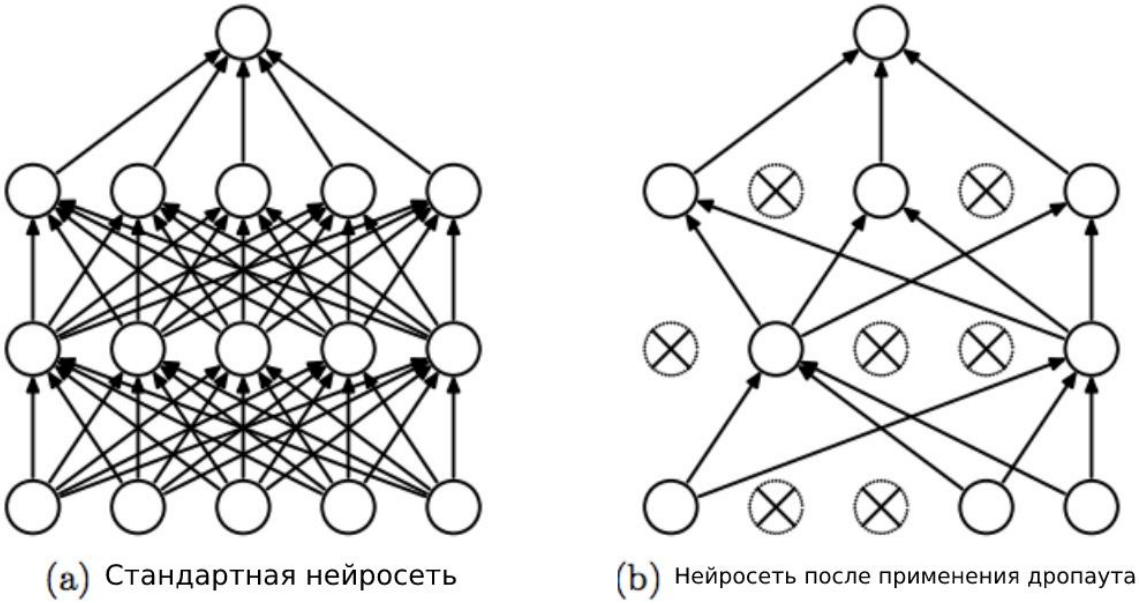


Рисунок 11 — Применение дропаута в нейросети

1.2 Архитектура сверточной нейронной сети CNN

Сверточная нейронная сеть (англ. convolutional neural network, CNN) — архитектура нейронных сетей, предложенная Яном Лекуном, основным приоритетом использования которой является эффективное распознавание изображений.

В основе CNN лежат понятия ядра свертки (англ. kernel) и пулинга (англ. pooling). Свертка и пулинг составляют процесс под названием feature learning. Ядро также называется фильтром и представляет собой сетку *NxN*, которая применяется к изображению. Веса ядер свертки являются обучаемыми параметрами. Притом каждое изображение представляется матрицей пикселей, где каждому пикселю соответствует числовое значение. В частности, для черно-

белых изображений каждому пикселю соответствует значение яркости из диапазона (0, 255), а для цветных изображений проводится разложение по каналам RGB (red, green, blue) [7]. На рисунке 12 представлен процесс свертки.

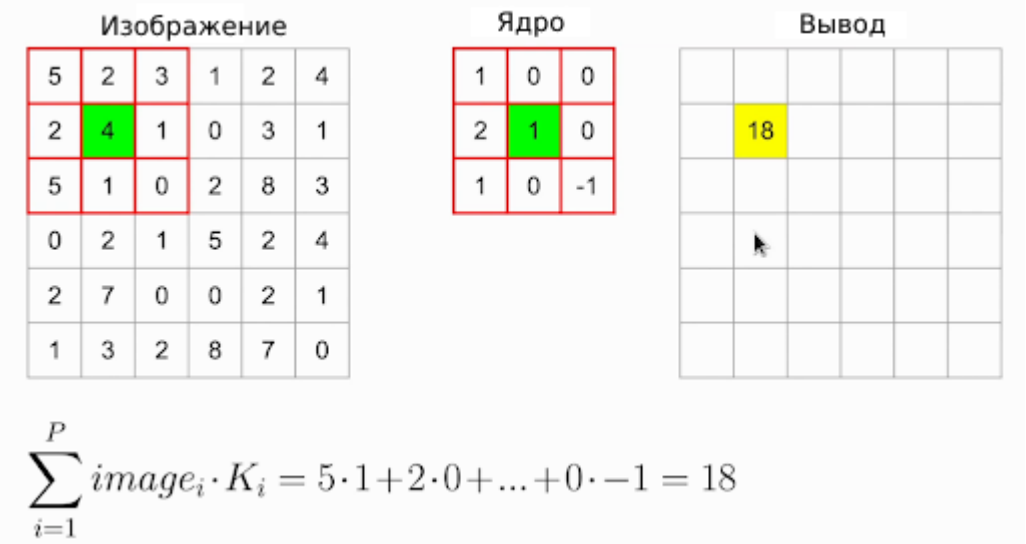


Рисунок 12 — Процесс свертки в CNN

Значения, которыми заполняется выходное изображение (также двумерная матрица), представляют собой сумму произведений соответствующих значений фрагмента матрицы изображения и ядра свертки. В процессе свертки сетка ядра смещается на заданный шаг, т.е выполняется сдвиг (англ. stride). Для заполнения границ выходного массива исходная матрица изображения может быть дополнена пикселями (англ. padding), что представлено на рисунке 13:

Рисунок 13 — Дополнение изображения

В случае цветного изображения ядра свертки будет иметь дополнительную размерность, называемую глубиной (англ. depth).

Пулинговый слой используется для уменьшения размера изображения. В основе процесса лежит разбиение изображения на отдельные блоки размером *WxH* (где *W* — ширина блока, *H* — высота блока), и выполнение некоторой функции для каждого блока. Как правило используется функция максимума max pooling или функция взвешенного среднего average pooling. Пример операции пулинга представлен на рисунке 14:

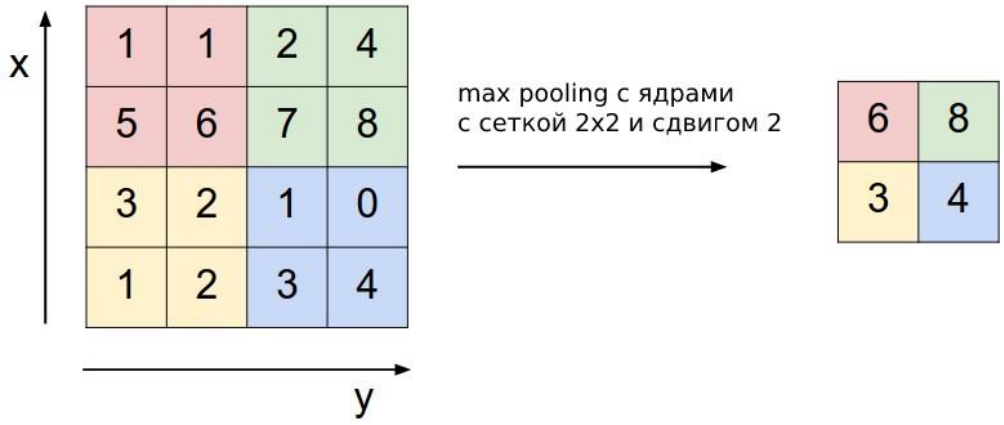
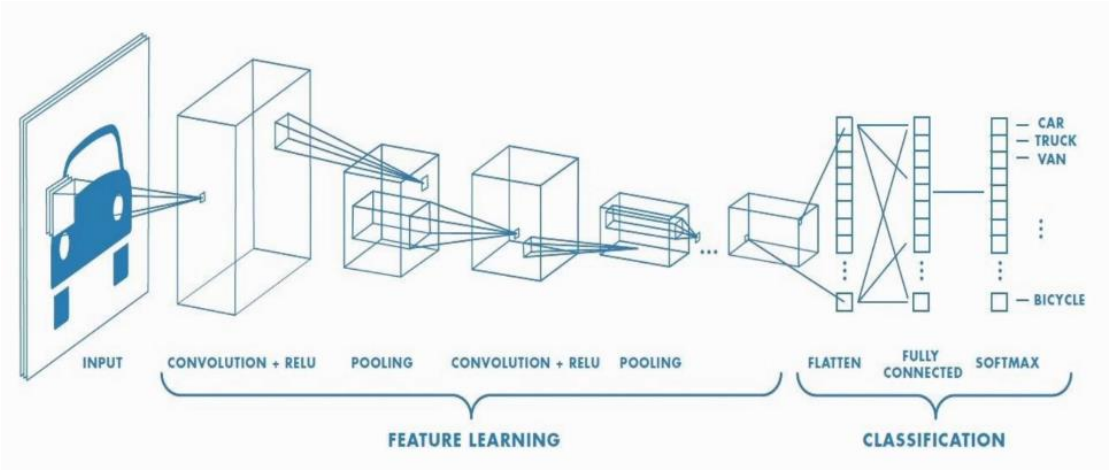


Рисунок 14 — Операция пулинга

Обучаемых параметров у пулингового слоя нет. Его основной целью является уменьшение изображения для того, чтобы последующие свертки выполнялись над большей областью исходного изображения, а также ускорение вычислений.

После выполнения свертки результирующая матрица, как правило, «распрямляется» в одномерный вектор, элементы которого затем поступают на вход полносвязных слоев нейронной сети, что представлено на общей схеме на рисунке 15:



В процессе feature learning происходит распознавание признаков изображения от менее существенных (например, отдельных линий), до более существенных (форм, элементов). В случае работы с аудиоданными, аудиофайлы тоже можно представлять в виде, пригодном для обработки с помощью CNN. Для этого следует использовать спектрограмму аудиофайла, где в качестве строк и столбцов матрицы выступят значения времени и частоты, а в качестве значения пикселя — громкость в данной точке. Получение спектрограммы будет рассмотрено далее.

1.3 Архитектура нейронной сети LSTM

LSTM (англ. long short term memory) — архитектура нейронной сети, которая является одним из видов рекуррентной нейронной сети RNN (англ. recurrent neural network). RNN представляет собой архитектуру нейронной сети, которая эффективно справляется с обработкой последовательных данных. Стандартная архитектура RNN подразумевает, что на каждом шаге обработки данных сеть сохраняет состояние, полученное на этом шаге и использует его в следующем шаге [8]. Структура рекуррентной сети представлена на рисунке 16:

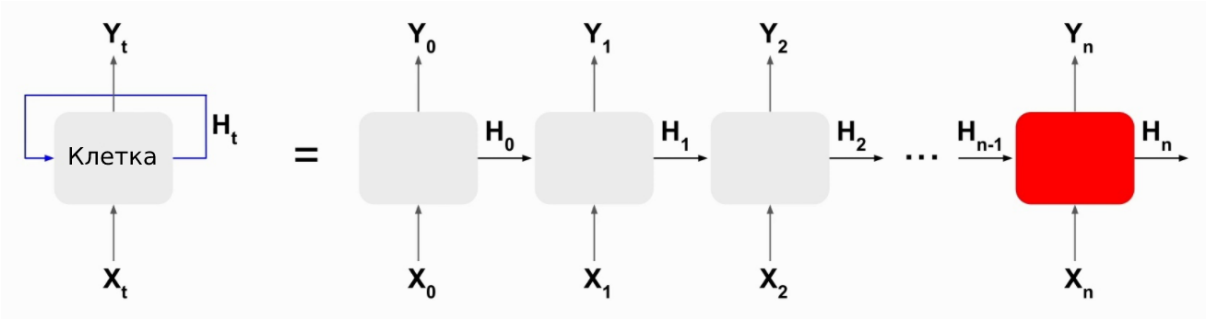


Рисунок 16 — Структура RNN

Основной единицей RNN является клетка, структура и формулы вычисления элементов которой представлены на рисунке 17:

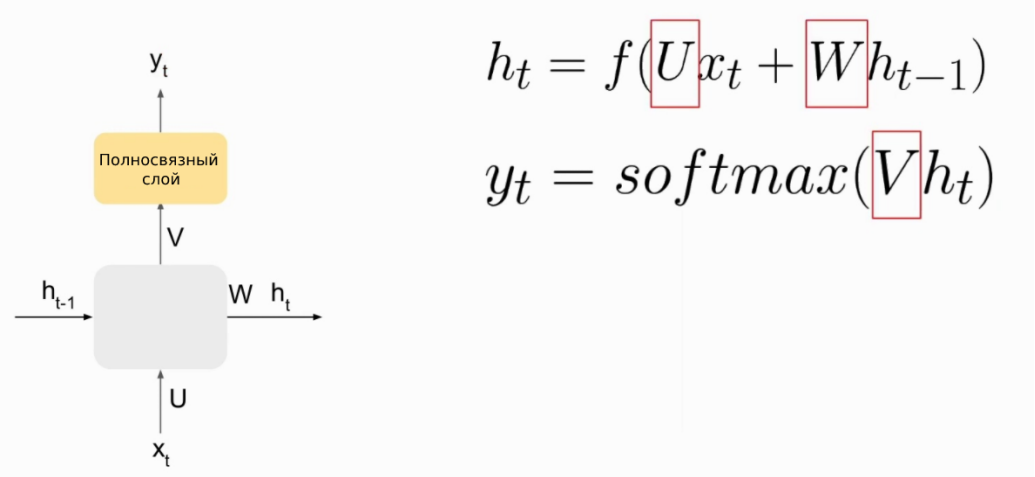


Рисунок 17 — Структура клетки RNN

Из формул видна зависимость текущего получаемого состояния h_t от предыдущего состояния h_{t-1}. Параметры U, W и V являются матрицами весов и являются теми параметрами, обновление которых с целью минимизации ошибки и является задачей тренировки нейронной сети.

Входные данные для RNN представлены тройкой:

$$(batchsize, s, d),$$

где *batchsize* — размер батча, s — число шагов, d — число размерностей.

Входные данные для каждой клетки представлены парой:

$$(batchsize, d),$$

Выходные данные каждой клетки представлены парой:

$$(batchsize, u),$$

где u — выходное число размерностей.

Основной проблемой классической RNN является то, что она не способна обращаться к информации, полученной на более чем одном предыдущем шаге, учиться выявлять шаблоны поведения, основанные на длительных зависимостях между данными, т.к. связь осуществляется только между двумя соседними состояниями, т.е. RNN не имеет долгосрочной памяти. Для устранения данных недостатков используется архитектура LSTM.

LSTM позволяет накапливать информацию и использует для этого отдельное состояние c_t. При этом на каждом шаге информация способна проходить фильтрацию, т.е. определяется, какая информация из предыдущего состояния имеет высокую значимость, а какая может быть исключена. То же самое происходит с новой информацией, получаемой на каждом шаге [9].

Клетка LSTM сети представлена на рисунке 18:

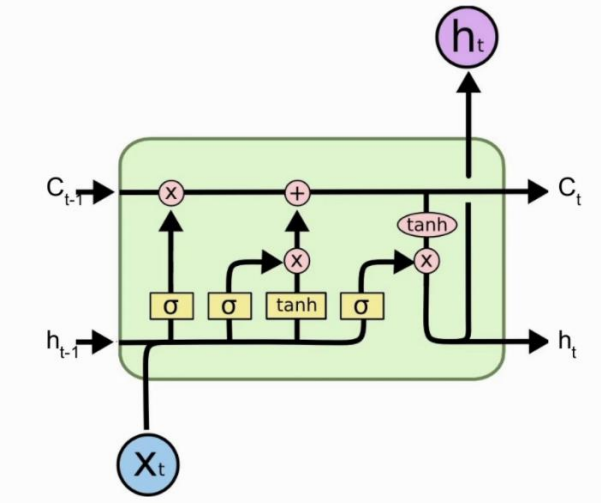


Рисунок 18 — Клетка LSTM сети

Первая сигмоидная функция определяет первый блок клетки, который называется блок очистки памяти (англ. forget gate). Здесь рассчитывается матрица, элементы которой в результате работы сигмоиды либо приближены к 0, либо приближены к 1. Эта матрица применяется к предыдущему состоянию памяти, таким образом, часть данных из предыдущего состояния сохраняется, а часть — очищается. Эти шаги представлены формулами (1) и (2):

$$f_t = \sigma(W_t[x_t, h_{t-1}] + b_t), \quad (1)$$

где W_t — матрица весов для данного рекуррентного слоя, b_f — смещение для данного рекуррентного слоя, $[x_t, h_{t-1}]$ — конкатенация входного вектора и вектора предшествующего состояния;

$$C_t^f = C_{t-1} * f_t \quad (2)$$

Таким образом, выполняется первый шаг преобразования состояния памяти.

Блок обновления памяти (англ. input gate) является вторым блоком клетки LSTM и представляет собой поэлементное умножение результатов работы второй сигмоидной функции и тангенциальной функции, а затем применение полученной матрицы к состоянию памяти C_t^f, полученному в результате формулы (2). Формулы вычислений подобны формулам (1) и (2) с соответствующими параметрами:

$$i_t = \sigma(W_i[x_t, h_{t-1}] + b_i)$$

$$C_t' = tanh(W_c[x_t, h_{t-1}] + b_c)$$

$$C_t^i = C_t' * i_t \quad (3)$$

И в конечном итоге результаты работы двух блоков объединяются:

$$C_t = C_t^f + C_t^i \quad (4)$$

Третий блок является блоком выдачи результата (англ. output gate). Здесь происходит формирование нового состояния h_t путем поэлементного объединения результата работы сигмоидной функции и тангенциальной функции над полученным состоянием памяти (3):

$$o_t = \sigma(W_o[x_t, h_{t-1}] + b_o)$$

$$h_t = o_t * \tanh(C_t).$$

Звуковые данные представляют собой последовательности значений, в которых присутствуют определенные структурные шаблоны и сходства. Применимость LSTM архитектуры для задачи классификации музыкальных произведений по жанрам обеспечена способностью нейросетей данной архитектуры выделять эти сходства. Это должно позволить достаточно эффективно классифицировать данные.

1.4 Теория обработки звуковых данных

Звук — это физическое явление, которое представляет собой механические упругие волны, распространяющиеся в различных средах. Для человека звук характеризуется двумя основными характеристиками — громкость и тон. Громкость зависит от параметра, именуемого амплитудой (англ. amplitude): чем больше амплитуда, тем громче звук. Амплитуда измеряется в децибелах. Тон определяется параметром, называемым частотой (англ. frequency) — число колебаний, совершаемых за некий промежуток времени, измеряется в герцах. Чем меньше период одного колебания, тем больше их число за промежуток времени и тем выше частота, следовательно, тем выше звук [10]. Амплитуда и частота могут быть явно проиллюстрированы на графике звуковой волны, что представлено на рисунке 19:

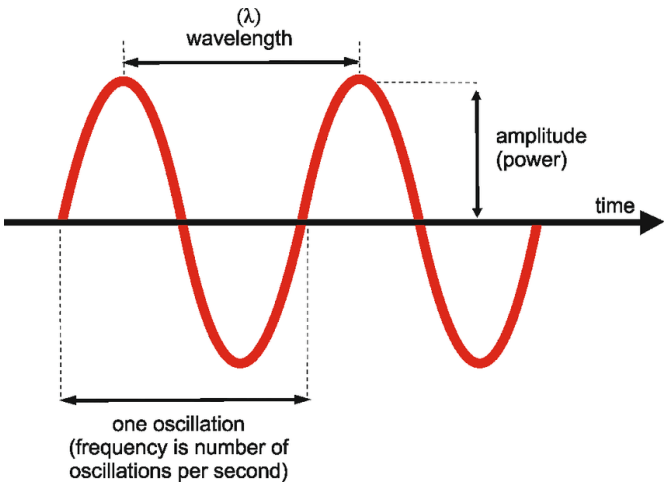


Рисунок 19 — Амплитуда и частота звуковой волны

Источник: Wavelength, amplitude and frequency [Электронный ресурс]. — URL: https://www.researchgate.net

В реальной жизни амплитуда и частота звука зачастую непостоянны и изменяются во времени. Сложные звуки могут сочетать в себе несколько звуковых волн одновременно, например, звук аккорда на пианино представляет собой комбинацию звуковых волн каждой ноты.

Дискретное преобразование Фурье (англ. Discrete Fourier Transform, DFT) — математическое преобразование, позволяющее разделить звуковой сигнал, распространяющийся во времени, на множество входящих в него частот, которые представляют собой частоты звуковых волн, формирующих исходный звук. Результат представляет собой функцию зависимости амплитуды частот, входящих в звук, что также называется звуковым спектром [11]. График звукового спектра представлен на рисунке 20:

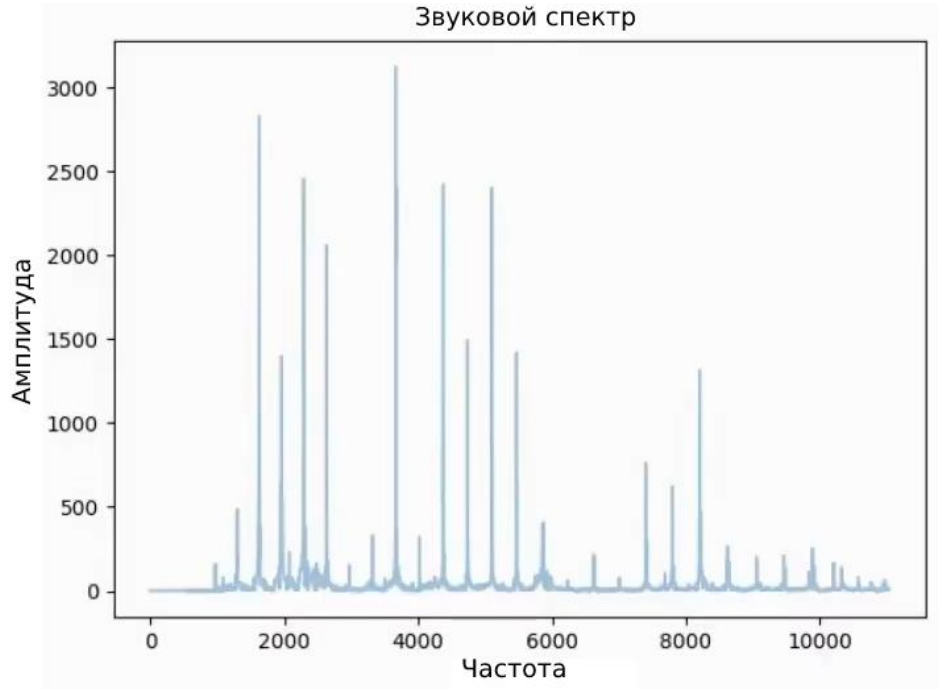


Рисунок 20 — Звуковой спектр

Быстрое преобразование Фурье (англ. Fast Fourier Transform, FFT) — это алгоритм, идентичный в своей сути дискретному преобразованию Фурье, но являющийся его оптимизированной версией, требующей меньше вычислений и уменьшающей сложность отдельных преобразований.

При использовании дискретного преобразования Фурье происходит переход из временной области (англ. time domain) в частотную область (англ. frequency domain). Это значит, что информация о времени теряется и не может быть использована. При этом время является одним из ключевых параметров, который требуется для выделения признаков для подачи на вход нейронной сети.

Кратковременное преобразование Фурье (англ. Short-time Fourier Transform, STFT) — преобразование, вычисляющее некоторое число FFT с определенным интервалом и тем самым позволяющее получить информацию об изменении в частотном спектре с течением времени. Каждый интервал зависит от величины, которая в теории аналого-цифровых преобразований называется частотой дискретизации (англ. sample rate), т.е. каждый интервал должен содержать равное фиксированное число единиц, называемых сэмплами. Одной из основных особенностей STFT является получение спектрограммы — графика,

который предоставляет зависимость трех переменных: времени, частоты и амплитуды звука. Пример спектрограммы приведен на рисунке 21:

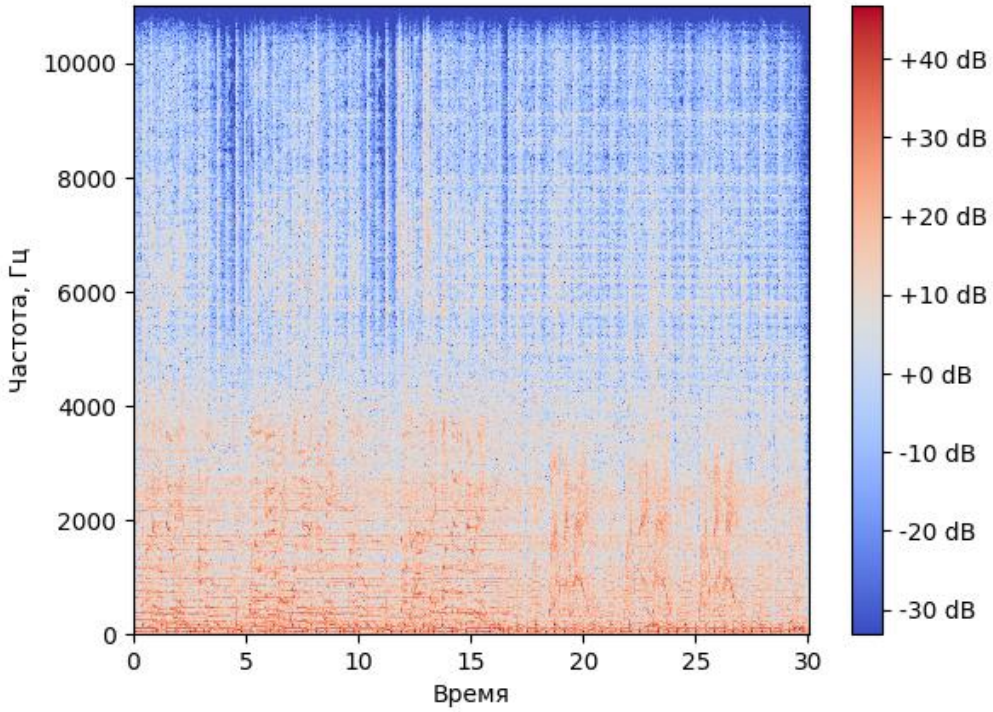


Рисунок 21 — Спектрограмма

1.4.1 Мел-частотные кепстральные коэффициенты

При работе с аудиоданными, как и в любой задаче машинного обучения, требуется выделить признаки, которые можно будет эффективно использовать в качестве данных. Одними из наиболее эффективных признаков для задач, связанных с анализом звуковых данных, являются мел-частотные кепстральные коэффициенты (англ. Mel Frequency Cepstral Coefficients, MFCCs). Они являются признаками частотной области, позволяют зафиксировать тембр и текстуру звука, рассчитываются на каждом интервале исходного сигнала и также могут фиксироваться в достаточно большом количестве (обычно в диапазоне от 13 до 40 коэффициентов) [12]. Эти особенности делают MFCC очень эффективными при решении таких задач классификации, как определение музыкального инструмента, распознавание жанра музыкального отрывка, определение настроения музыкального отрывка, распознавание речи и др. Спектрограмма с выделенными MFCC на оси частот называется мел-спектрограммой. На рисунке

22 приведен пример мел-спектрограммы для 13 MFCC некоторого звукового файла:

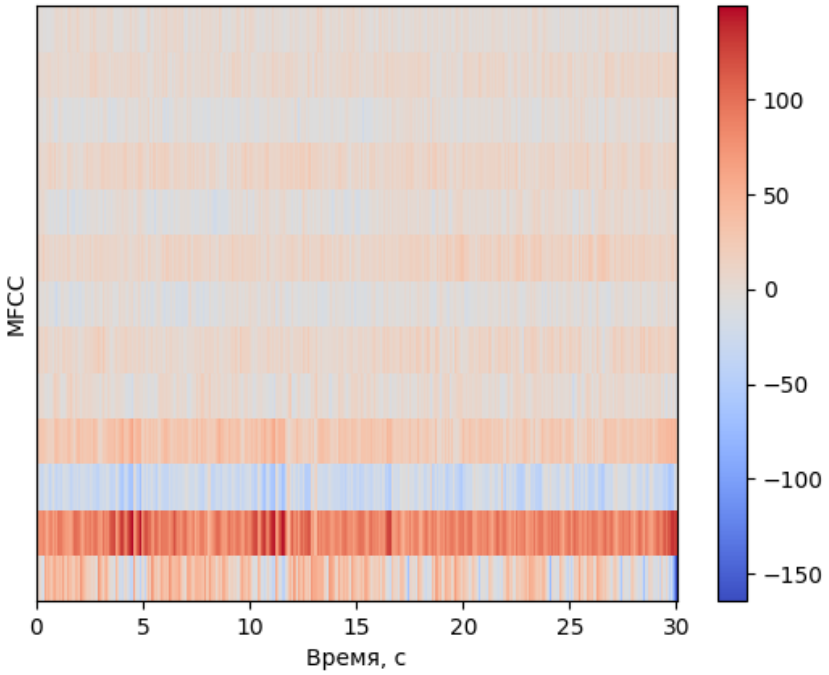


Рисунок 22 — Пример MFCC

По мел-спектрограмме по аналогии с частотами в обычной спектрограмме можно определить интенсивность участия каждого коэффициента в аудиофайле в разные моменты времени.

1.5 Выбор методов и средств реализации программного продукта

1.5.1 Язык программирования

Одну из основных ролей в исследовании поставленной задачи играет выбранный язык программирования. В данной работе в качестве языка программирования выбран Python 3.6. Данный язык является одним из самых распространенных языков для исследования и решения задач машинного обучения. Одним из основных преимуществ Python является наличие большого множества вспомогательных библиотек, которые облегчают процесс построения моделей нейронных сетей.

1.5.2 Библиотека TensorFlow

TensorFlow — это открытая библиотека, используемая при разработке систем, применяющих технологии машинного обучения. Она содержит в себе реализацию эффективных и мощных алгоритмов, рассчитанных на решение задач машинного обучения, среди которых особо выделяются принятие решений и распознавание образов.

Разработчики TensorFlow стремились обеспечить ее гибкость, расширяемость и переносимость. В результате этого подхода, ее широко используют в самых различных вычислительных средах — от тех, которые формируются мобильными устройствами, до сред, представленных огромными кластерами. С момента своего перехода в открытый доступ, Tensorflow стала одной из самых популярных библиотек машинного обучения. Её всё чаще используют при проведении исследований, разработке реальных приложений и в обучении.

Для построения моделей нейронных сетей в работе используется модуль Keras, который представляет собой реализацию TensorFlow спецификации Keras. Keras имеет два вида API для построения архитектур нейронных сетей: Sequential и Functional.

Sequential API позволяет создать модель нейронной сети с последовательным расположением внутренних слоев от входного к выходному.

Functional API позволяет создать модель нейронной сети более сложной структуры с нелинейным расположением внутренних слоев, общими слоями, несколькими слоями входа и выхода и т.д. Такая модель представляет собой произвольный направленный ациклический граф [13].

1.5.3 Библиотека Librosa

Librosa — вспомогательная библиотека Python, представляющая возможности для обработки аудиоданных. Она позволяет извлекать информацию о загружаемых звуковых файлах, в том числе получить из них признаки, подходящие для подачи на входной слой нейронной сети.

1.5.4 Библиотека NumPy

NumPy — библиотека языка Python, позволяющая осуществлять работу с многомерными массивами и матрицами, а также эффективно выполнять над ними различные математические операции.

В данной работе библиотека NumPy в первую очередь используется для хранения данных, например, входных данных или массивов ожидаемых значений.

1.5.5 Библиотека Json

Библиотека Json представляет собой модуль для обеспечения хранения данных. Он позволяет кодировать и декодировать данные в удобном формате. В данной работе модуль Json используется для хранения данных, получаемых в процессе подготовки звуковых файлов.

1.5.6 Библиотека os

Библиотека os позволяет получить доступ к функционалу операционной системы. В данной работе эта библиотека используется при подготовке входных данных для прохода по директориям, содержащим звуковые файлы, и считывания этих файлов.

1.6 Набор данных

В качестве источника звуковых данных используется GTZAN dataset — массив данных в свободном доступе, созданный для нужд машинного обучения и содержащий большое количество музыкальных композиций. Данные разделены на категории, соответствующие 10 музыкальным жанрам, каждая категория содержит 100 аудиофайлов. Длительность каждого аудиофайла составляет 30 секунд.

GTZAN dataset является одним из самых популярных массивов звуковых данных. Данные для набора GTZAN собирались в период с 2000 по 2001 год с

различных источников, таких как радиопередачи, микрофонные записи, компакт-диски и т.д., с целью обеспечения широкого разнообразия качества аудиофайлов [14].

2 Создание и сравнение моделей нейронных сетей

2.1 Формальная постановка задачи

В рамках поставленной задачи требуется сделать следующее:

1) Считать и с помощью библиотеки Librosa преобразовать звуковые данные из набора, выделив MFCC коэффициенты. Полученный результат сохранить в файле, удобном для чтения.

2) С помощью библиотеки TensorFlow создать модель нейронной сети архитектуры CNN.

3) С помощью библиотеки TensorFlow создать модель нейронной сети архитектуры LSTM.

4) На основе имеющихся данных выполнить тренировку каждой из моделей и оценить их работу на тестовых данных.

5) Сравнить результаты работы нейронных сетей и сделать выводы.

2.2 Архитектура реализации

Код программы разбит на три части, каждая из которых хранится в отдельном файле: preprocessing.py, cnn_classifier.py и lstm_classifier.py. Файл preprocessing.py содержит код обработки аудиофайлов с целью их преобразования в формат входных данных нейронных сетей. Файл cnn_classifier.py содержит код построения модели рекуррентной нейронной сети CNN. Файл lstm_classifier.py содержит код построения модели нейронной сети LSTM.

2.2.1 Предварительная подготовка аудиоданных

В файле preprocessing.py определяются следующие константы: dataset_path — директории с папками, содержащими аудиофайлы; json_path — путь к файлу,

в который будут сохранены данные; sample_rate — фиксированная частота дискретизации; track_duration — длительность каждого аудиофайла; samples_per_track — число сэмплов в аудиофайле, что равно частоте дискретизации, умноженной на длительность аудиофайла.

Создание входных данных реализует функция save_mfcc(), которая принимает на вход параметры: n_mfcc — число выделяемых MFCC коэффициентов, n_fft — длина интервала в сэмплах для проведения быстрого преобразования Фурье, hop_length — смещение между интервалами, num_segments — число сегментов, на которые будет разделен каждый аудиофайл,используется для аугментации данных. Последний булевый параметр pred — в значении True указывает, что будет выполняться предобработка только одного аудиофайла для выполнения предсказания.

В начале работы функции определяется словарь data, который содержит три ключа: mapping — соответствует названиям имеющихся жанров; labels — соответствует индексу из списка жанров mapping, который определяется для каждого вектора MFCC каждого сегмента аудиофайла; mfcc — содержит MFCC векторы для каждого сегмента аудиофайла. Таким образом в результате заполнения словаря data будут получены размеченные данные. Фрагмент функции представлен в листинге 1.

Листинг 1 — Функция save_mfcc()

```
def save_mfcc(dataset_path, json_path, num_mfcc=13, n_fft=2048,
hop_length=512, num_segments=5):
    # словарь для хранения данных
    data = {
        "mapping": [],
        "labels": [],
        "mfcc": []
    }
```

```
    samples_per_segment = int(SAMPLES_PER_TRACK / num_segments)
    num_mfcc_vectors_per_segment = math.ceil(samples_per_segment
/ hop_length)
```

Следующим шагом устанавливается число сэмплов и число MFCC векторов в каждом сегменте аудиофайла. После этого выполняется проход в цикле по папке с файлами каждого жанра и с помощью методов из библиотеки Librosa load() и feature.mfcc() выполняется чтение каждого аудиофайла и посегментное выделение MFCC коэффициентов с заданными параметрами. Полученные данные из словаря data преобразуются в файл расширения .json. Фрагмент кодом с обработкой представлен на листинге 2.

Листинг 2 — Фрагмент функции save_mfcc()

```
for d in range(num_segments):

    # начало очередного сегмента
    start = samples_per_segment * d

    # конец очередного сегмента
    finish = start + samples_per_segment

    # получаем MFCC для заданного сегмента и параметров
    mfcc = librosa.feature.mfcc(signal[start:finish],
sample_rate, n_mfcc=num_mfcc, n_fft=n_fft,
                                hop_length=hop_length)
    mfcc = mfcc.T

    # сохраняем, если полученный вектор имеет заданную длину
    if len(mfcc) == num_mfcc_vectors_per_segment:
        data["mfcc"].append(mfcc.tolist())
```

```
            data["labels"].append(i - 1)
            print("{}, segment:{}".format(file_path, d + 1))
```

Полный код файла preprocessing.py приведен в приложении А.

2.2.2 Построение CNN модели нейронной сети

В файле cnn_classifier.py происходит построение модели рекуррентной нейронной сети. Первым шагом с помощью метода split_dataset() происходит разделение данных на тренировочные, проверочные и тестовые. Проверочные данные необходимы для выполнения функции тестовых в ходе тренировки сети, в то время как тестовая часть будет зарезервирована и будет представлять собой абсолютно новые данные для сети. В данной реализации сети 25% данных отводится под зарезервированные тестовые данные, 20% от оставшихся тренировочных отводится под проверочные данные.

Данные получаются с помощью метода load_data(), который обращается к созданному ранее json файлу. Функция представлена на листинге 3.

Листинг 3 — Функция load_data()

```
def load_data(dataset_path):
    with open(dataset_path, "r") as fp:
        data = json.load(fp)

    inputs = np.array(data["mfcc"])
    targets = np.array(data["labels"])

    return inputs, targets
```

В методе build_model() происходит построение модели CNN сети. Создаваемая модель является моделью Sequential типа. Передаваемая переменная input_shape определяет вид входных данных сети. Для CNN этот вид

фактически представляет собой четырехразмерный массив, где первая размерность определяет число сэмплов, вторая — количество шагов, третья — число MFCC, а четвертая — глубину, в данном случае равную 1, поскольку данные рассматриваются как изображение с одним каналом. Модель содержит три рекуррентных слоя, один скрытый полносвязный и выходной. Каждый из рекуррентных слоев имеет 32 ядра и сетки ядра имеют размерность 3x3, за исключением последнего слоя — его сетка имеет размерность 2x2. В качестве функций активации используется ReLu. Размеры сеток пулинговых слоев аналогичны размерам для рекуррентных слоев. Горизонтальный и вертикальный шаг равен 1.

Скрытый полносвязный слой имеет 64 нейрона, в качестве функции активации используется ReLu. Для избежания переобучения применяется метод дропаута с вероятностью 0.3.

Гипер-параметры, относящиеся к слоям нейросети, подобраны экспериментально из числа часто используемых значений. Их изменение не привело к существенному изменению точности работы нейросети.

Выходной слой имеет 10 нейронов, соответствующих 10 имеющимся музыкальным жанрам, функцией активации является Softmax.

Фрагмент функции build_model() представлен на листинге 4.

Листинг 4 — Фрагмент функции build_model()

```
# 1 сверточный слой
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape))
model.add(keras.layers.MaxPool2D((3, 3), strides=(2, 2),
padding='same'))
model.add(keras.layers.BatchNormalization())  # нормализация
активаций в текущем слое

# 2 сверточный слой
```

```
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape))
model.add(keras.layers.MaxPool2D((3, 3), strides=(2, 2),
padding='same'))
model.add(keras.layers.BatchNormalization())

# 3 сверточный слой
model.add(keras.layers.Conv2D(32, (2, 2), activation='relu',
input_shape=input_shape))
model.add(keras.layers.MaxPool2D((2, 2), strides=(2, 2),
padding='same'))
model.add(keras.layers.BatchNormalization())

# распрямление результата и передача в полносвязный слой
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))

# выходной слой
model.add(keras.layers.Dense(10, activation='softmax'))
```

С помощью метода compile() выполняется компиляция модели. Скорость обучения модели устанавливается равной 0.0001, что является одним из часто используемых значений, в качестве функции потери используется разреженная категориальная кроссэнтропия. Для начала тренировки нейронной сети используется функция fit(). На вход подаются тренировочные и проверочные массивы. Гипер-параметр batch size устанавливается равным 32, а число эпох — 30. Это часто используемые стандартные значения, их экспериментальное изменение в рамках данного исследования не привело к изменениям в результатах тренировки сети.

Последним шагом является оценка работы модели на тестовых данных, что выполняется с помощью функции evaluate().

Полный код файла cnn_classifier.py приведен в приложении Б.

2.2.3 Построение LSTM модели нейронной сети

В файле lstm_classifier.py происходит построение модели нейронной сети LSTM архитектуры. Основные гипер-параметры остаются теми же, что используются для построения рекуррентной модели в пункте 2.2.2 текущей работы, однако присутствуют изменения.

Всего модель содержит два LSTM слоя и выходной слой. Выбранное число элементов в LSTM слоях — 64. В качестве функции активации используется функция ReLu. Вероятность дропаута на втором слое устанавливается равной 0.3. Функция build_model() представлена на листинге 5.

Листинг 5 — Функция build_model()

```
def build_model(input_shape):
    # создание модели
    model = keras.Sequential()

    # 2 LSTM слоя
    model.add(keras.layers.LSTM(64, input_shape=input_shape,
return_sequences=True))
    model.add(keras.layers.LSTM(64))

    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.5))

    # выходной слой
    model.add(keras.layers.Dense(10, activation='softmax'))

    return model
```

Основным отличием от CNN является то, что переменная input_shape имеет трехмерную размерность, поскольку отсутствует параметр глубины, характерный для CNN.

Полный код файла lstm_classifier.py приведен в приложении В.

2.3 Тестирование и сравнение работы нейросетей

Основные результаты работы нейросетей можно оценить по метрикам точности и ошибки. Для имеющихся данных результаты тренировки рекуррентной нейросети представлены на рисунках 23 и 24:

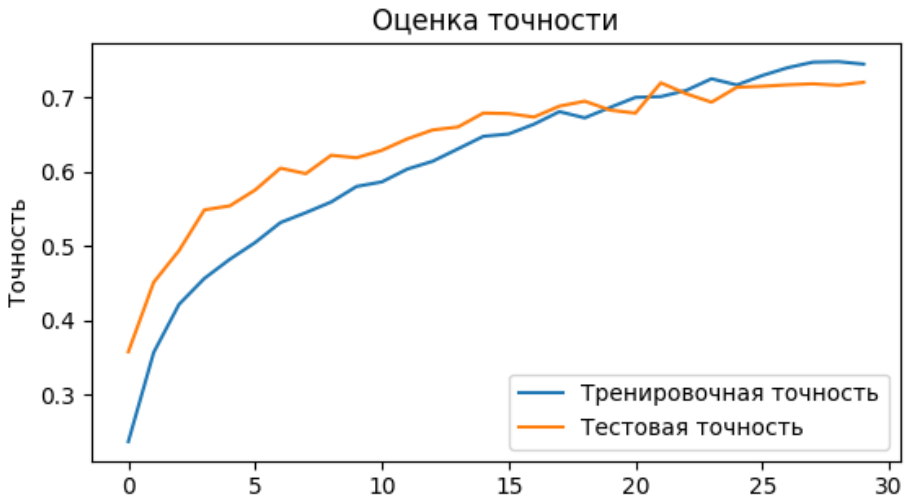


Рисунок 23 — Оценка точности CNN

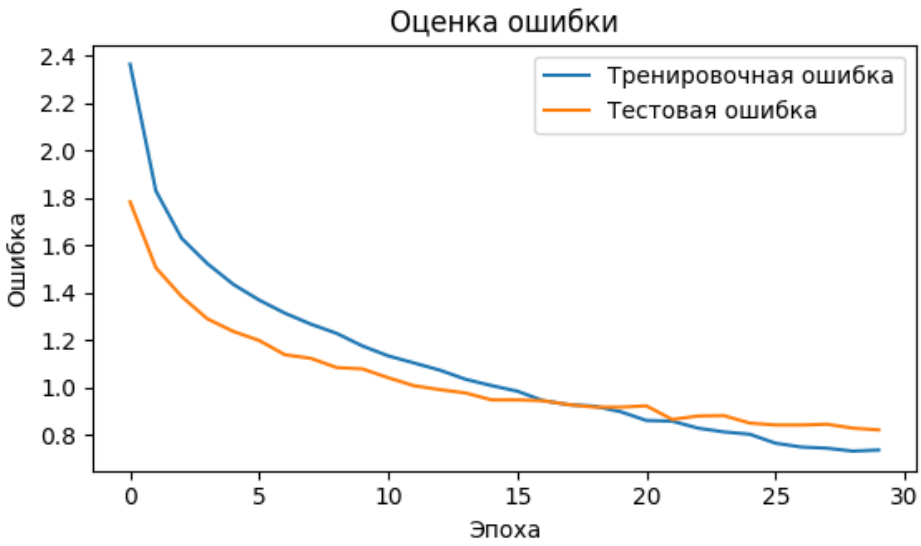


Рисунок 24 — Оценка ошибки CNN

Тестовая точность модели CNN составила приблизительно 71%.

Время тренировки составило 2 минуты 3 секунды.

Аналогичные результаты для модели архитектуры LSTM приведены на рисунках 25 и 26 соответственно:

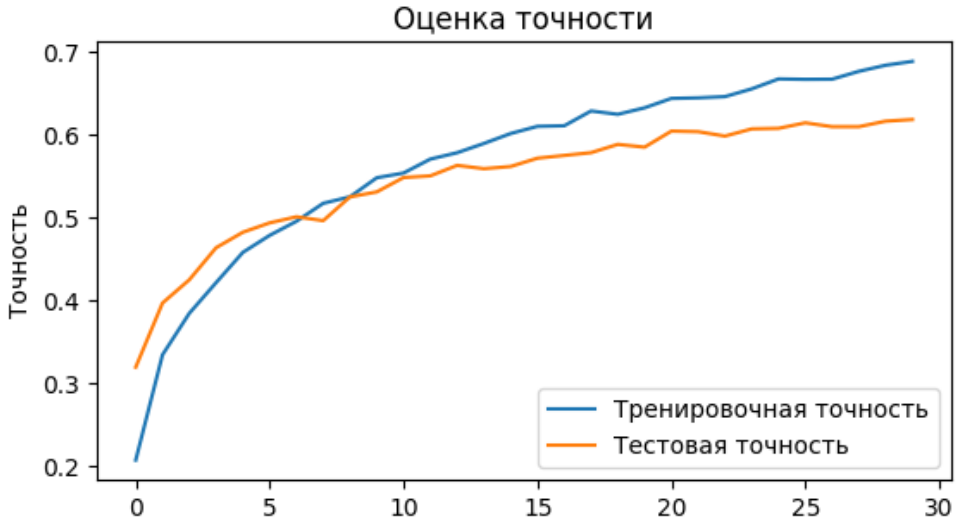


Рисунок 25 — Оценка точности LSTM

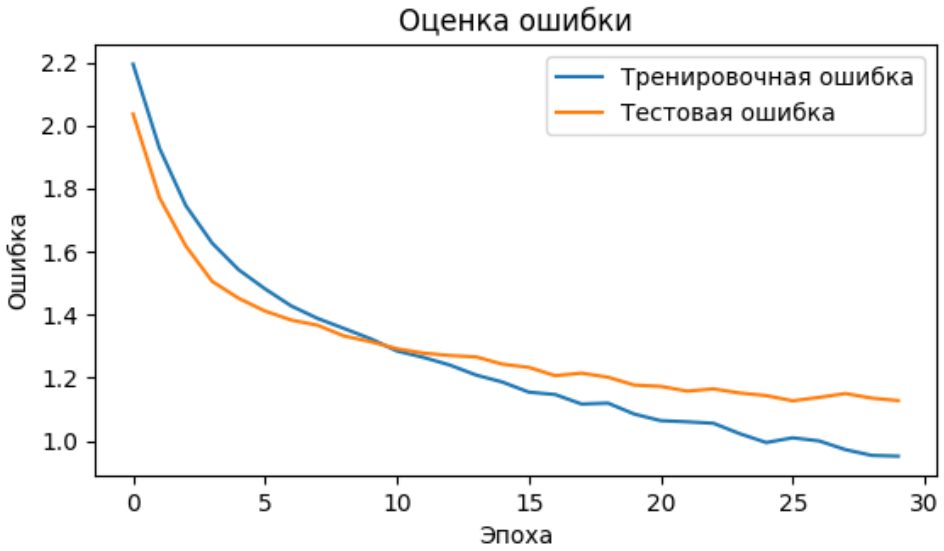


Рисунок 26 — Оценка ошибки LSTM

Тестовая точность модели LSTM составила приблизительно 62%.

Время тренировки составило 8 минут 9 секунд.

Как видно по графикам полученных метрик, CNN модель в процессе тренировки достаточно стабильно улучшает точность и уменьшает ошибку,

притом разница между тренировочными и проверочными метриками является весьма незначительной.

В то же время LSTM модель начиная с приблизительно 10 эпохи увеличивает разницу между тестовой и проверочной точностью и ошибкой. Иными словами, на проверочных данных модель работает хуже, т.е. имеет место некоторое переобучение модели.

Разница тестовой точности моделей составила около 10%. На рисунке 27 представлено сравнение двух графиков:

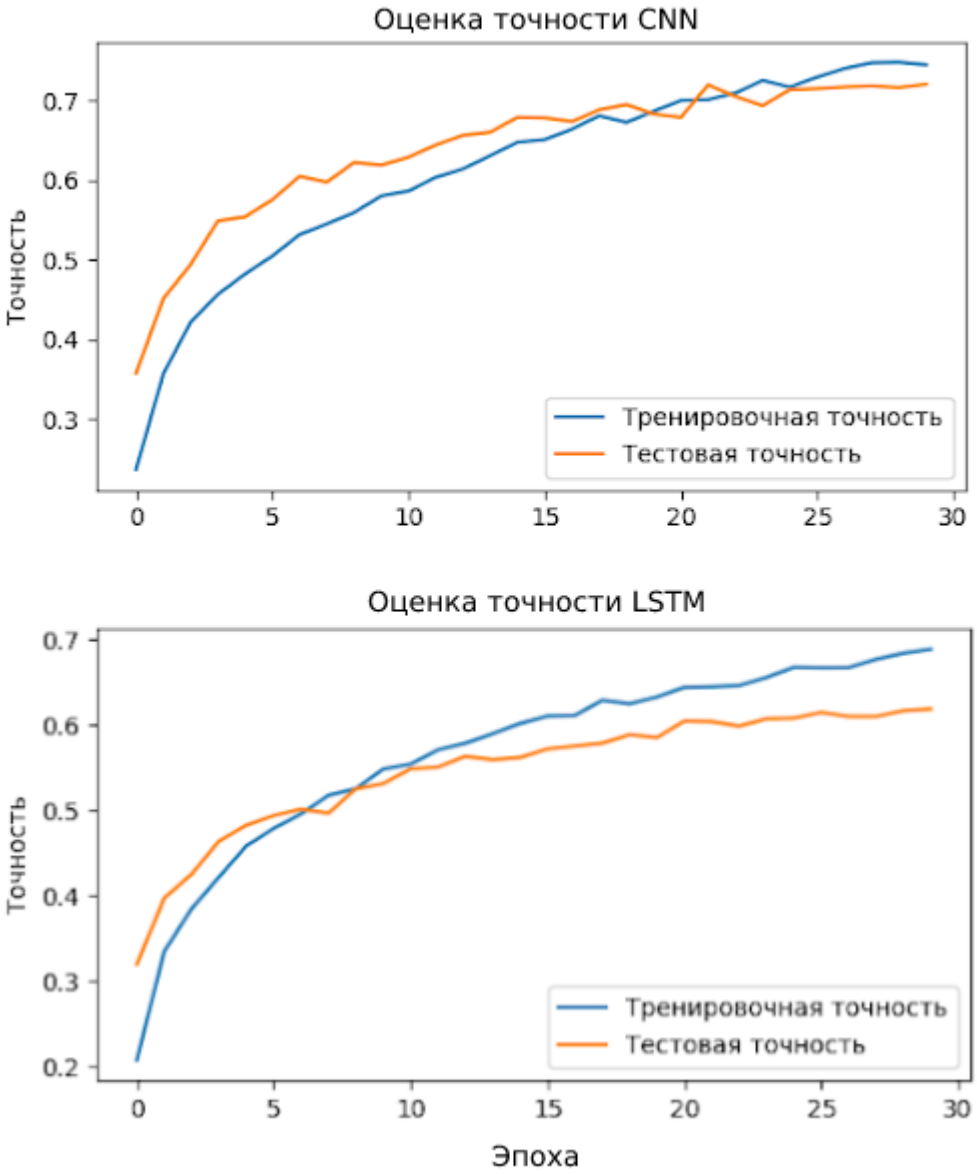


Рисунок 27 — Сравнение точности двух моделей

В качестве попытки изменить результат работы LSTM модели, вероятность дропаута была увеличена до 0.5 и данные перераспределены так, чтобы увеличить число тренировочных данных. Результаты представлены на рисунках 27 и 28:

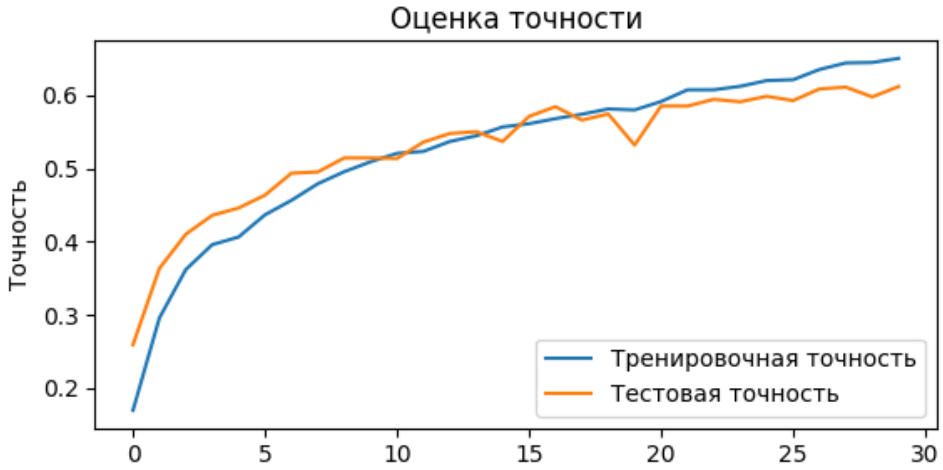


Рисунок 27 — Оценка точности LSTM с измененными параметрами

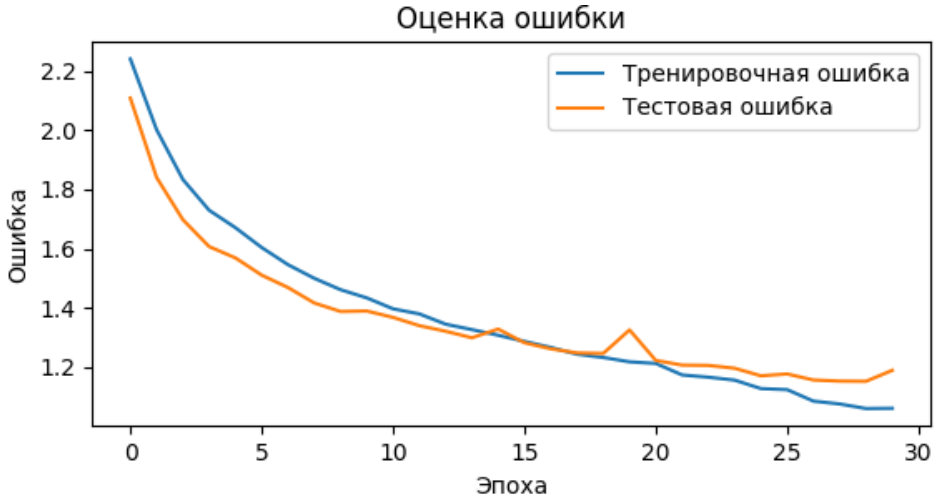


Рисунок 27 — Оценка ошибки LSTM с измененными параметрами

Из результатов видно, что это привело к несущественному улучшению тренировочных результатов, точка начала увеличения разницы сместилась. Однако тестовая точность сохранила значение в районе 60%.

Таким образом модель CNN показала более точный результат в сравнении с моделью LSTM архитектуры.

2.3.1 Выполнение предсказаний на новых данных

Для выполнения предсказаний используется функция predict(), в качестве параметров которой передается *model* — созданная модель нейросети, *X* — индекс, соответствующий одному из выделенных списков с MFCC коэффициентами, *y* — индекс, соответствующий номеру ожидаемого музыкального жанра.

В качестве песен для выполнения предсказаний были выбраны 5 композиций не из исходного набора, для каждой из которых был проведен процесс предобработки. Результаты определения класса для каждой песни продемонстрированы далее:

Первая композиция: Slipknot - Duality, жанр — метал. Результаты предсказаний представлены на рисунке 28:

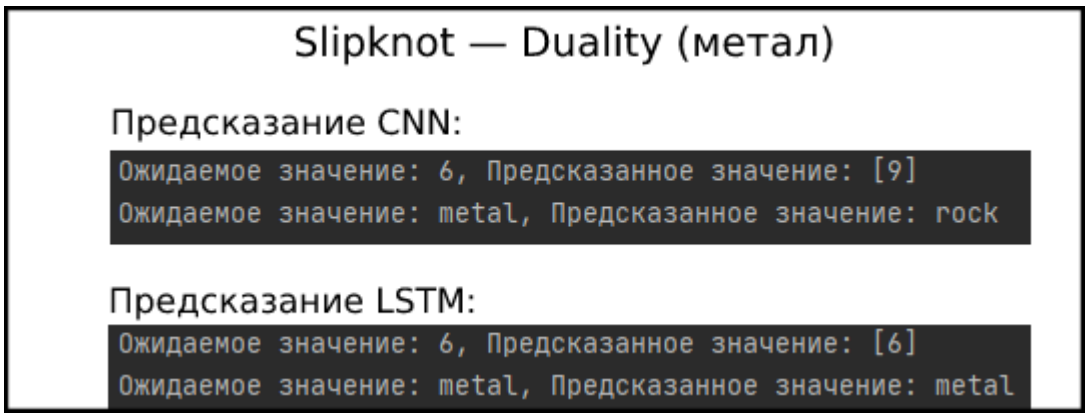


Рисунок 28 — Первое предсказание

Вторая композиция: Paul McCartney — Mull of Kintyre, жанр — кантри. Результаты предсказаний представлены на рисунке 29:

Рисунок 29 — Второе предсказание

Третья композиция: Jessica — Like a Burning Star, жанр — поп. Результаты предсказаний представлены на рисунке 30:

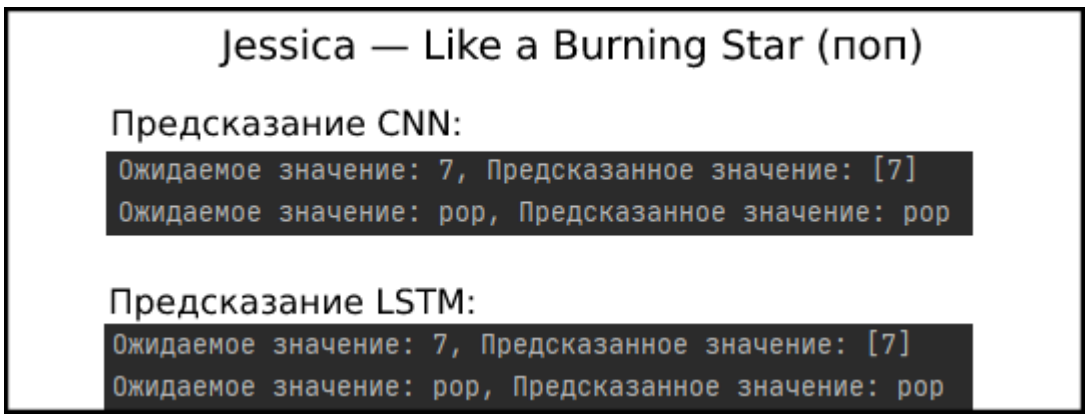


Рисунок 30 — Третье предсказание

Четвертая композиция: Ice Cube — It Was a Good Day, жанр — хип-хоп. Результаты предсказаний представлены на рисунке 31:

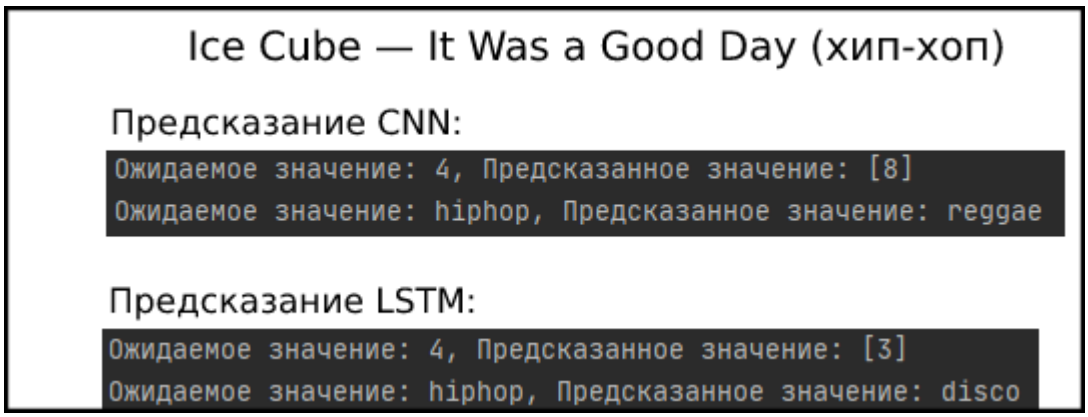


Рисунок 31 — Четвертое предсказание

Пятая композиция: Taster — Today, жанр — рок. Результаты предсказаний представлены на рисунке 32:

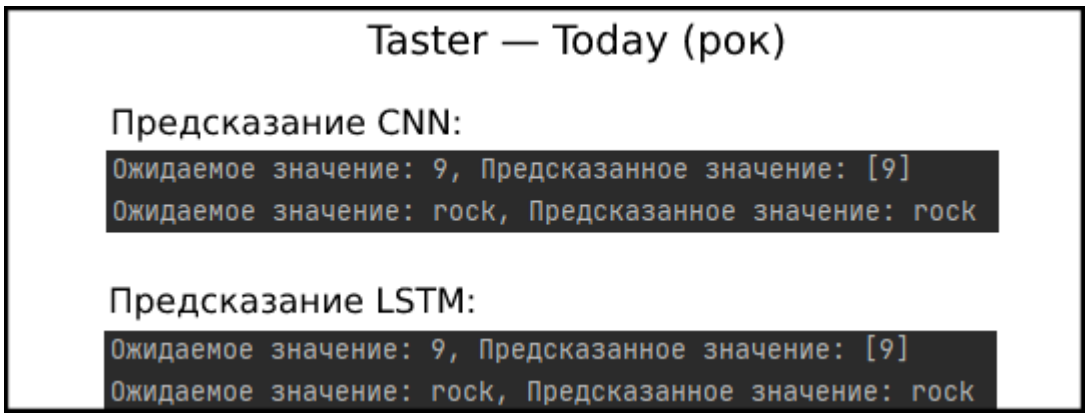


Рисунок 32 — Четвертое предсказание

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы было проведено ознакомление с задачей классификации музыкальных произведений по жанрам и с теорией нейронных сетей, включая описание архитектур CNN и LSTM. Был рассмотрен процесс компьютерного представления и подготовки звуковых данных для подачи на вход моделей нейронных сетей. Для каждой из архитектур была построена и обучена модель, для тренировки использовался общий набор данных GTZAN dataset.

Обе использованных архитектуры, CNN и LSTM, показали достаточно неплохие результаты, учитывая тот факт, что число имеющихся данных не слишком велико. При этом, модель архитектуры CNN достигла более точных результатов при выполнении классификации и тренировка данной модели осуществлялась гораздо быстрее.

Поскольку количество тренировочных данных из GTZAN dataset не столь велико, даже с учетом проведенной на начальном этапе аугментации данных, не стоит окончательно отбрасывать LSTM модель как неэффективную, поскольку увеличение объема данных может значительно улучшить точность работы обеих моделей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Нейронная сеть и ее имитация в науке [Электронный ресурс]. — URL: https://medium.com/eggheado-science/5d412f9af08c (дата обращения: 04.04.2021)
2. Нейронные сети — математический аппарат [Электронный ресурс]. — URL: https://basegroup.ru/community/articles/math (дата обращения: 05.04.2021)
3. Shalev-Shwartz S. Understanding Machine Learning. - 32 Avenue of the Americas, New York, NY, 2014. - 397 с. (дата обращения: 19.05.2021)
4. Эпоха, батч, итерация — в чем различия? [Электронный ресурс]. — URL: https://neurohive.io/ru/osnovy-data-science/jepoha-razmer-batcha-iteracija/ (дата обращения: 05.04.2021)
5. Функции активации [Электронный ресурс]. — URL: https://neurohive.io/ru/osnovy-data-science/activation-functions/ (дата обращения: 11.04.2021)
6. Методы борьбы с переобучением искусственных нейронных сетей [Электронный ресурс]. — URL: https://na-journal.ru/2-2019-tehnicheskie-nauki/1703-metody-borby-s-pereobucheniem-iskusstvennyh-neironnyh-setei (дата обращения: 11.04.2021)
7. Как работает сверточная нейронная сеть [Электронный ресурс]. — URL: neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set/ (дата обращения: 13.04.2021)
8. Рекуррентная сеть LSTM [Электронный ресурс]. — URL: http://mechanoid.su/neural-net-lstm.html (дата обращения: 14.04.2021)
9. Understanding LSTM network [Электронный ресурс]. — URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата обращения: 14.04.2021)
10. Understanding Sound [Электронный ресурс]. — URL: https://www.nps.gov/subjects/sound/understandingsound.html (дата обращения: 20.04.2021)

11. An Interactive Guide To The Fourier Transform [Электронный ресурс]. — URL: betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/ (дата обращения: 20.04.2021)

12. Мел-кепстральные коэффициенты (MFCC) [Электронный ресурс]. — URL: https://www.pvsm.ru/programmirovanie/4344 (дата обращения: 20.04.2021)

13. Keras Functional API в Tensorflow [Электронный ресурс] // URL: https://habr.com/ru/post/483664/ (дата обращения: 22.03.2021)

14. GTZAN dataset — music genre classification [Электронный ресурс] URL: www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification (дата обращения: 16.03.2021)

ПРИЛОЖЕНИЕ А

Код файла preprocessing.py

```
import os

import librosa
import math
import json

DATASET_PATH = "test_song"
JSON_PATH = "test_song.json"
SAMPLE_RATE = 22050
TRACK_DURATION = 30  # measured in seconds
SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION


def save_mfcc(dataset_path, json_path, num_mfcc=13, n_fft=2048,
hop_length=512, num_segments=5, pred=False):
   # словарь для хранения данных
   data = {
       "mapping": [],
       "labels": [],
       "mfcc": []
   }

   samples_per_segment = int(SAMPLES_PER_TRACK / num_segments)
   num_mfcc_vectors_per_segment = math.ceil(samples_per_segment /
hop_length)

   # проход по всем жанрам
   for i, (dirpath, dirnames, filenames) in
enumerate(os.walk(dataset_path)):

       # если не находимся в корневой директории
       if dirpath is not dataset_path:

           # сохраняем семантический лейбл (название жанра)
           semantic_label = dirpath.split("/")[-1]
           data["mapping"].append(semantic_label)
           print("\nProcessing: {}".format(semantic_label))

           # обрабатываем все файлы в папке текущего жанра
```

```
            for f in filenames:

                # сохраняем путь к текущему файлу
                file_path = os.path.join(dirpath, f)
                signal, sample_rate = librosa.load(file_path,
sr=SAMPLE_RATE)

                # обрабатываем сегменты аудиофайла, извлекаем MFCC
и сохраняем данные
                for d in range(num_segments):

                    # начало очередного сегмента
                    start = samples_per_segment * d

                    # конец очередного сегмента
                    finish = start + samples_per_segment

                    # получаем MFCC для заданного сегмента и
параметров
                    mfcc =
librosa.feature.mfcc(signal[start:finish], sample_rate,
n_mfcc=num_mfcc, n_fft=n_fft,

hop_length=hop_length)
                    mfcc = mfcc.T

                    # сохраняем, если полученный вектор имеет
заданную длину
                    if len(mfcc) == num_mfcc_vectors_per_segment:
                        data["mfcc"].append(mfcc.tolist())
                        data["labels"].append(i - 1)
                        print("{}, segment:{}".format(file_path, d
+ 1))
                        if pred:
                            filename =
file_path.split('\\')[2].split('.')[0]
                            json_path = filename + ".json"

    with open(json_path, "w") as fp:
        json.dump(data, fp, indent=4)


if __name__ == "__main__":
    save_mfcc(dataset_path=DATASET_PATH, json_path=JSON_PATH,
```

```
num_segments=10, pred=False)
```

ПРИЛОЖЕНИЕ Б

Код файла cnn_classifier.py

```
import json
import time
import os

import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

DATASET_PATH = "data.json"
MODEL_SAVE_PATH = "cnn_model.h5"
MAPPING = ''
DATA_IS_SET = False

def get_genre_name(index):
   s = MAPPING[index]
   splits = s.split('\\')
   return splits[1][0:]

def load_data(dataset_path):
   with open(dataset_path, "r") as fp:
       data = json.load(fp)

   inputs = np.array(data["mfcc"])
   targets = np.array(data["labels"])

   global MAPPING
   if not DATA_IS_SET:
       MAPPING = np.array(data["mapping"])

   return inputs, targets

def plot_history(history):
   fig, axs = plt.subplots(2)

   # точность
```

```
    axs[0].plot(history.history["accuracy"], label="Тренировочная
точность")
    axs[0].plot(history.history["val_accuracy"], label="Тестовая
точность")
    axs[0].set_ylabel("Точность")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Оценка точности")

    # ошибка
    axs[1].plot(history.history["loss"], label="Тренировочная ошибка")
    axs[1].plot(history.history["val_loss"], label="Тестовая ошибка")
    axs[1].set_ylabel("Ошибка")
    axs[1].set_xlabel("Эпоха")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Оценка ошибки")

    plt.show()

def split_dataset(test_size, validation_size):
    X, y = load_data(DATASET_PATH)

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)

    X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=validation_size)

    X_train = X_train[..., np.newaxis]
    X_validation = X_validation[..., np.newaxis]
    X_test = X_test[..., np.newaxis]

    return X_train, X_validation, X_test, y_train, y_validation,
y_test


def build_model(input_shape):
    # создание модели
    model = keras.Sequential()

    # 1 сверточный слой
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape))
    model.add(keras.layers.MaxPool2D((3, 3), strides=(2, 2),
padding='same'))
```

```
   model.add(keras.layers.BatchNormalization())  # нормализация
активаций в текущем слое

   # 2 сверточный слой
   model.add(keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape))
   model.add(keras.layers.MaxPool2D((3, 3), strides=(2, 2),
padding='same'))
   model.add(keras.layers.BatchNormalization())

   # 3 сверточный слой
   model.add(keras.layers.Conv2D(32, (2, 2), activation='relu',
input_shape=input_shape))
   model.add(keras.layers.MaxPool2D((2, 2), strides=(2, 2),
padding='same'))
   model.add(keras.layers.BatchNormalization())

   # распрямление результата и передача в полносвязный слой
   model.add(keras.layers.Flatten())
   model.add(keras.layers.Dense(64, activation='relu'))
   model.add(keras.layers.Dropout(0.3))

   # выходной слой
   model.add(keras.layers.Dense(10, activation='softmax'))

   return model

def predict(model, X, y):
   X = X[np.newaxis, ...]

   prediction = model.predict(X)

   pred_index = np.argmax(prediction, axis=1)
   print("Ожидаемое значение: {}, Предсказанное значение:
{}".format(y, pred_index))
   print("Ожидаемое значение: {}, Предсказанное значение:
{}".format(get_genre_name(y), get_genre_name(pred_index[0])))

if __name__ == "__main__":

   model = None

   if not os.path.exists(MODEL_SAVE_PATH):
       # создание тренировочных, проверочных и тестовых данных
```

```
        X_train, X_validation, X_test, y_train, y_validation, y_test =
split_dataset(0.25, 0.2)
        DATA_IS_SET = True

        # создание модели CNN
        input_shape = (X_train.shape[1], X_train.shape[2],
X_train.shape[3])
        model = build_model(input_shape)

        # компиляция модели
        optimizer = keras.optimizers.Adam(learning_rate=0.0001)
        model.compile(optimizer=optimizer,
loss="sparse_categorical_crossentropy",
                      metrics=['accuracy'])

        # тренировка CNN
        start_time = time.perf_counter()
        history = model.fit(X_train, y_train,
validation_data=(X_validation, y_validation), batch_size=32,
epochs=30)
        end_time = time.perf_counter()

        # вывод точности и ошибки
        plot_history(history)

        # применение модели для тестовых данных
        test_error, test_accuracy = model.evaluate(X_test, y_test,
verbose=1)
        print(f"Время тренировки = {end_time - start_time:0.4f} сек")
        print("Тестовая точность = {}".format(test_accuracy))

        model.save(MODEL_SAVE_PATH)
    else:
        model = keras.models.load_model(MODEL_SAVE_PATH)
        load_data("data.json")
        DATA_IS_SET = True

    X_pred, y_pred = load_data("today.json")
    X_pred = X_pred[..., np.newaxis]
    X = X_pred[0]
    predict(model, X, 9)
```

ПРИЛОЖЕНИЕ В

Код файла lstm_classifier.py

```
import json
import os
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt
import time

DATASET_PATH = "data.json"
MODEL_SAVE_PATH = "lstm_model.h5"
MAPPING = ''
DATA_IS_SET = False

def get_genre_name(index):
   s = MAPPING[index]
   splits = s.split('\\')
   return splits[1][0:]


def load_data(dataset_path):
   with open(dataset_path, "r") as fp:
       data = json.load(fp)

   inputs = np.array(data["mfcc"])
   targets = np.array(data["labels"])

   global MAPPING
   if not DATA_IS_SET:
       MAPPING = np.array(data["mapping"])

   return inputs, targets


def plot_history(history):
   fig, axs = plt.subplots(2)

   # точность
   axs[0].plot(history.history["accuracy"], label="Тренировочная
точность")
```

```
    axs[0].plot(history.history["val_accuracy"], label="Тестовая
точность")
    axs[0].set_ylabel("Точность")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Оценка точности")

    # ошибка
    axs[1].plot(history.history["loss"], label="Тренировочная
ошибка")
    axs[1].plot(history.history["val_loss"], label="Тестовая ошибка")
    axs[1].set_ylabel("Ошибка")
    axs[1].set_xlabel("Эпоха")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Оценка ошибки")

    plt.show()


def split_dataset(test_size, validation_size):
    X, y = load_data(DATASET_PATH)

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size)

    X_train, X_validation, y_train, y_validation =
train_test_split(X_train, y_train, test_size=validation_size)

    return X_train, X_validation, X_test, y_train, y_validation,
y_test


def build_model(input_shape):
    # создание модели
    model = keras.Sequential()

    # 2 LSTM слоя
    model.add(keras.layers.LSTM(64, input_shape=input_shape,
return_sequences=True))
    model.add(keras.layers.LSTM(64))

    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.5))

    # выходной слой
```

```
    model.add(keras.layers.Dense(10, activation='softmax'))

    return model


def predict(model, X, y):
    X = X[np.newaxis, ...]

    prediction = model.predict(X)

    pred_index = np.argmax(prediction, axis=1)
    print("Ожидаемое значение: {}, Предсказанное значение:
{}".format(y, pred_index))
    print("Ожидаемое значение: {}, Предсказанное значение:
{}".format(get_genre_name(y), get_genre_name(pred_index[0])))


if __name__ == "__main__":

    model = None

    if not os.path.exists("lstm_model.h5"):
        # создание тренировочных, проверочных и тестовых данных
        X_train, X_validation, X_test, y_train, y_validation, y_test
= split_dataset(0.2, 0.15)
        DATA_IS_SET = True

        # создание модели CNN
        input_shape = (X_train.shape[1], X_train.shape[2])
        model = build_model(input_shape)

        # компиляция модели
        optimizer = keras.optimizers.Adam(learning_rate=0.0001)
        model.compile(optimizer=optimizer,
loss="sparse_categorical_crossentropy",
                      metrics=['accuracy'])
        model.summary()

        # тренировка CNN
        start_time = time.perf_counter()
        history = model.fit(X_train, y_train,
validation_data=(X_validation, y_validation), batch_size=32,
epochs=30)
        end_time = time.perf_counter()
```

```
        # вывод точности и ошибки
        plot_history(history)

        # применение модели для тестовых данных
        test_error, test_accuracy = model.evaluate(X_test, y_test,
verbose=1)
        print(f"Время тренировки = {end_time - start_time:0.4f} сек")
        print("Тестовая точность = {}".format(test_accuracy))

        model.save(MODEL_SAVE_PATH)
    else:
        model = keras.models.load_model(MODEL_SAVE_PATH)
        load_data("data.json")
        DATA_IS_SET = True

    X_pred, y_pred = load_data("today.json")
    X = X_pred[0]
    predict(model, X, 9)
```