Федеральное государственное бюджетное образовательное учреждение

«Сибирский государственный университет

телекоммуникаций и информатики»

Факультет мобильной радиосвязи и мультимедиа

Кафедра систем автоматизированного проектирования

РГЗ

По дисциплине: Метрология

По теме: Многофакторный эксперимент – Разработка метрики и исследование точности распознавания рукописного текста

Выполнил: студент группы РИ-88

Веретин Максим Петрович

Проверил: старший преподаватель

Шыырап Юрий Монгушевич

Новосибирск 2020

Оглавление

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ 3](#_Toc59193722)

[ВВЕДЕНИЕ 3](#_Toc59193723)

[ТЕМАТИЧЕСКИЙ ОБЗОР 4](#_Toc59193724)

[Схема работы 7](#_Toc59193725)

[Функция активации 8](#_Toc59193726)

[Ошибка 9](#_Toc59193727)

[АНАЛИЗ АНАЛОГОВ И ПРОТОТИПОВ 10](#_Toc59193728)

[Классификация нейросетей: 10](#_Toc59193729)

[Методы распознавания 12](#_Toc59193730)

[РАЗРАБОТКА ПРОГРАММНОЙ ЧАСТИ 18](#_Toc59193731)

[Обзор модели 18](#_Toc59193732)

[Операции 20](#_Toc59193733)

[Реализация с использованием TensorFlow 22](#_Toc59193734)

[МНОГОФАКТОРНЫЙ ЭКСПЕРИМЕНТ 26](#_Toc59193735)

[Разработка метрики измерения точности распознавания рукописного текста 26](#_Toc59193736)

[Обучение нейронной сети 27](#_Toc59193737)

[Анализ 27](#_Toc59193738)

[SimpleHTR эксперименты 28](#_Toc59193739)

[ЗАКЛЮЧЕНИЕ 32](#_Toc59193740)

[ПРИЛОЖЕНИЕ 33](#_Toc59193741)

[Код на Python SamplePreprocessor.py 33](#_Toc59193742)

[Код на Python DataLoader.py 34](#_Toc59193743)

[Код на Python Model.py 37](#_Toc59193744)

[Код на Python main.py 44](#_Toc59193745)

[Код на Python analyze.py 48](#_Toc59193746)

[СПИСОК ЛИТЕРАТУРЫ 51](#_Toc59193747)

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Целью работы является разработка и исследование методов, алгоритмов и программ распознавания рукописного текста, обеспечивающих высокое качество их распознавания. В качестве результата распознавания мы должны получить текст в цифровом формате.

Системы автономного распознавания рукописного текста (далее HTR - Handwritten Text Recognition) преобразуют текст, содержащийся в отсканированных изображениях, в цифровой текст, пример показан на рис. 1. В данной работе будет разработана нейронная сеть (NN), которая обучается на изображениях слов из набора данных IAM (IAM Handwriting Database - база данных рукописного ввода, находящаяся в открытом доступе). Также будет проведена статистическая обработка данных эксперимента на предмет корректности распознавания текста.

Image for post

Рисунок 1 - Изображение слова (взятое из IAM) и его транскрипция в цифровой текст.

# ВВЕДЕНИЕ

Существует два вида распознавания: онлайновый и оффлайновый. Онлайновый — распознавание текста при написании стилусом или пальцем на экране или планшете. Первым КПК, который мог распознавать рукописный текст — Apple Newton (1993 год). Этот метод является более простым, т.к. в данном случае можно проследить процесс написания текста и на основании полученных данных построить алгоритм распознавания.

Оффлайновый — распознавание уже написанного текста на бумаге. Текст предоставляется в виде скана или фотографии документа, страницы книги и т.п.

Сложность задачи распознавания рукописного текста состоит в том, что каждое рукописное слово и буква – уникальны. Эту уникальность создают огромное разнообразие почерков, форм, размеров букв и многообразие языков. Так же бумага с текстом может содержать различные дефекты бумаги (например, посторонние пятна), что так же усложняет весь процесс.

На данный момент достигнутая точность распознавания рукописного текста даже ниже, чем для рукописного «печатного» текста. Более высокие показатели могут быть достигнуты только с использованием контекстной и грамматической информации. Например, в процессе распознания искать целые слова в словаре легче, чем пытаться проанализировать отдельные символы из текста. Знание грамматики языка может также помочь определить, является ли слово глаголом или существительным. Формы отдельных рукописных символов иногда могут не содержать достаточно информации, чтобы точно (с уровнем более 98 %) распознать весь рукописный текст. В приведенной трактовке HTR понимается как автоматическое распознавание с помощью специальных программ изображений символов печатного или рукописного текста, например, введенного в компьютер с помощью сканера, и преобразование его в формат, пригодный для обработки текстовыми процессорами, редакторами текстов и т.д.

# ТЕМАТИЧЕСКИЙ ОБЗОР

Нейронная сеть (далее НС) — это последовательность нейронов, соединенных между собой синапсами. Структура нейросетей была позаимствована прямиком из биологии. Благодаря такой структуре, машина обретает способность анализировать и даже запоминать различную информацию. Также, нейросети способны не только анализировать входящую информацию, но и воспроизводить ее из своей памяти. Другими словами, нейросеть это машинная интерпретация мозга человека, в котором находятся миллионы нейронов, передающих информацию в виде электрических импульсов.

Нейрон — это вычислительная единица, нечто, содержащее число. Нейрон получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной (синий), скрытый (красный) и выходной (зеленый), а также нейрон смещения (о котором чуть позже). В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Соответственно, есть входной слой, который получает информацию, n скрытых слоев (обычно их не больше 3), которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные данные (input data) и выходные данные (output data). В случае входного нейрона: input=output. В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации (пока что просто представим ее f(x)) и попадает в поле output.

Нейроны оперируют числами в диапазоне [0,1] или [-1,1]. Для обработки чисел, выходящих из этого диапазона, применяют метод нормализации – делят единицу на это число. Подробнее об этом будет рассказано далее.



Рисунок 2 - виды нейронов в нейросети

Синапс — это связь между двумя нейронами. У синапсов есть 1 параметр — вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. Допустим, есть 3 нейрона, которые передают информацию следующему. Тогда у нас есть 3 веса, соответствующие каждому из этих нейронов. У того нейрона, у которого вес будет больше, та информация и будет доминирующей в следующем нейроне (пример — смешение цветов). Именно благодаря этим весам, входная информация обрабатывается и превращается в результат.

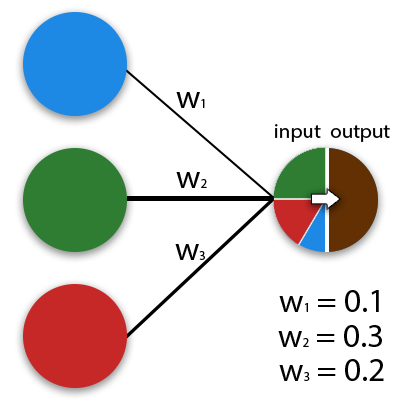


Рисунок 3 - пример того, как работают синапсы в нейросети

В нейронных сетях есть ещё один вид нейронов — нейрон смещения. Он отличается от основного вида нейронов тем, что его вход и выход в любом случае равняется единице. При этом входных синапсов такие нейроны не имеют. Расположение таких нейронов происходит по одному на слой и не более, также они не могут соединяться синапсами друг с другом. Размещать такие нейроны на выходном слое не целесообразно.

Бывают ситуации, в которых нейросеть просто не сможет найти верное решение из-за того, что нужная точка будет находиться вне пределов досягаемости. Именно для этого и нужны нейроны смещения, чтобы иметь возможность сместить область определения.

То есть вес синапса меняет изгиб графика функции, тогда как нейрон смещения позволяет осуществить сдвиг по оси координат Х, таким образом, чтобы нейросеть смогла захватить область недоступную ей без сдвига. При этом сдвиг может быть осуществлён как вправо, так и влево. Схематически нейроны сдвига обычно не обозначаются, их вес учитывается по умолчанию при расчёте входного значения. Также нейроны смещения позволят получить результат в том случае, когда все остальные нейроны выдают 0 в качестве выходного параметра. В этом случае независимо от веса синапса на каждый следующий слой будет передаваться именно это значение.

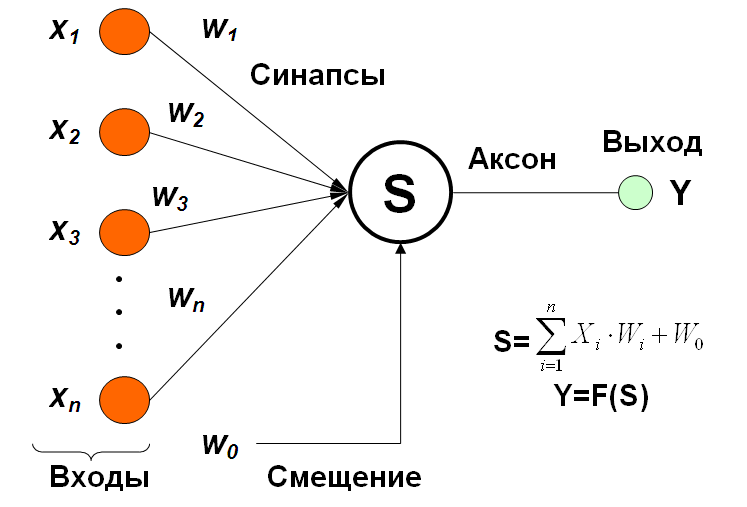


Рисунок 4 - схематическое изображение нейрона смещения

Схема работы

Общая схема или алгоритм, следующий:

— на входной слой нейронов происходит поступление определённых данных;

— информация передаётся с помощью синапсов следующему слою, причём каждый синапс имеет собственный коэффициент веса, а любой следующий нейрон способен иметь несколько входящих синапсов;

— данные, полученные следующим нейроном — это сумма всех данных для нейронных сетей, которые перемножены на коэффициенты весов (каждый на свой);

— полученное в итоге значение подставляется в функцию активации, в результате чего происходит формирование выходной информации;

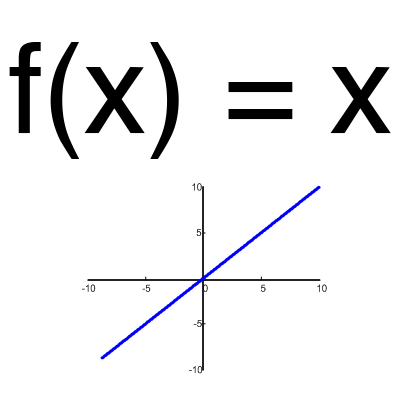
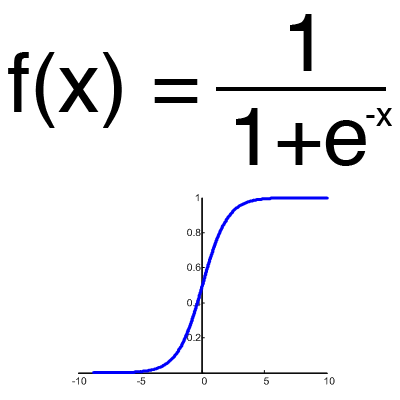
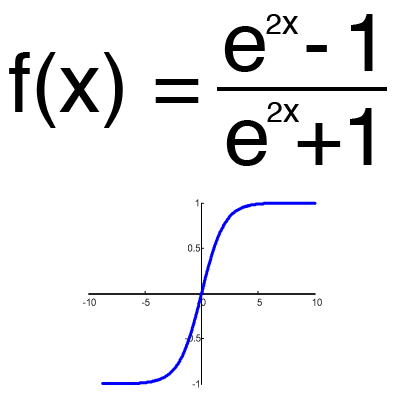
— информация передаётся дальше до тех пор, пока не дойдёт до конечного выхода.

Запустив такую сеть в первый раз, мы увидим, что ответ далек от правильно, потому что сеть не натренирована. Чтобы улучшить результаты мы будем ее тренировать. Но, прежде чем узнать, как это делать, давайте введем несколько терминов и свойств нейронной сети.

Функция активации

Функция активации — это способ нормализации входных данных. То есть, если на входе у вас будет большое число, пропустив его через функцию активации, вы получите выход в нужном вам диапазоне. Функций активации достаточно много поэтому мы рассмотрим самые основные: линейная, сигмоид (логистическая) и гиперболический тангенс. Главные их отличия — это диапазон значений.

* линейная функция - почти никогда не используется, за исключением случаев, когда нужно протестировать нейронную сеть или передать значение без преобразований;
* сигмоид – самая распространенная функция активации, ее диапазон значений [0,1]. Именно на ней показано большинство примеров в сети, также ее иногда называют логистической функцией. Соответственно, если в вашем случае присутствуют отрицательные значения (например, акции могут идти не только вверх, но и вниз), то вам понадобиться функция, которая захватывает и отрицательные значения.
* гиперболический тангенс – его имеет смысл использовать только тогда, когда ваши значения могут быть и отрицательными, и положительными, так как диапазон функции [-1,1]. Использовать эту функцию только с положительными значениями нецелесообразно так как это значительно ухудшит результаты вашей нейросети.

а) б) в)

Рисунок 5 – графическое представление и формула для:

а) линейной функции, б) сигмоида, в) гиперболического тангенса

Тренировочный сет — это последовательность данных, которыми оперирует нейронная сеть.

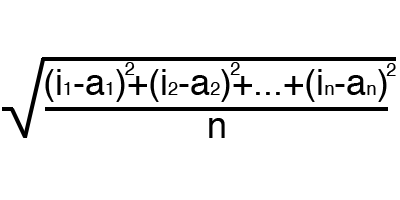
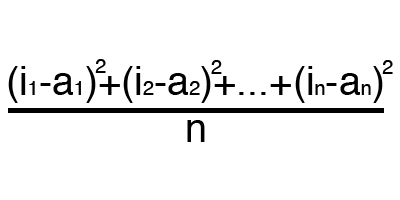
Итерация – это своеобразный счетчик, который увеличивается каждый раз, когда нейронная сеть проходит один тренировочный сет. Другими словами, это общее количество тренировочных сетов, пройденных нейронной сетью.

Эпоха – величина, которая устанавливается равной нулю при инициализации нейронной сети. Имеет верхний порог, задаваемый вручную. Чем больше эпоха, тем лучше натренирована сеть и соответственно, ее результат. Эпоха увеличивается каждый раз, когда мы проходим весь набор тренировочных сетов, в нашем случае, 4 сетов или 4 итераций.

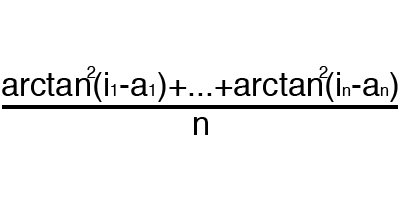
Важно не путать итерацию с эпохой и понимать последовательность их инкремента. Сначала n раз увеличивается итерация, а потом эпоха. Другими словами, нельзя сначала тренировать нейросеть только на одном сете, потом на другом и т. д. Нужно тренировать каждый сет один раз за эпоху. Так можно избежать ошибок в вычислениях.

Ошибка

Ошибка — это процентная величина, отражающая расхождение между ожидаемым и полученным ответами. Ошибка формируется каждую эпоху и должна идти на спад. Если этого не происходит, значит, вы что-то делаете не так. Ошибку можно вычислить разными путями, но мы рассмотрим лишь три основных способа: Mean Squared Error (далее MSE), Root MSE и Arctan. Здесь нет какого-либо ограничения на использование, как в функции активации, и вы вольны выбрать любой метод, который будет приносить вам наилучший результат. Стоит лишь учитывать, что каждый метод считает ошибки по-разному. У Arctan, ошибка, почти всегда, будет больше, так как он работает по принципу: чем больше разница, тем больше ошибка. У Root MSE будет наименьшая ошибка, поэтому, чаще всего, используют MSE, которая сохраняет баланс в вычислении ошибки.



а) б)



в)

Рисунок 6 – формулы для вычисления процента ошибок:

а) MSE, б), Root MSE, в) Arctan

# АНАЛИЗ АНАЛОГОВ И ПРОТОТИПОВ

Классификация нейросетей:

Существует множество видов нейросетей, прежде чем говорить о них, стоит отметить, что каждая нейронная сеть включает в себя первый слой нейронов, называемый входным. Этот слой не выполняет каких-либо преобразований и вычислений, его задача в другом: принимать и распределять входные сигналы по остальным нейронам. Этот слой единственный, являющийся общим для всех типов нейросетей, а критерием для деления является дальнейшая структура:

1. **Однослойная структура нейронной сети**. Представляет собой структуру взаимодействия нейронов, в которой сигналы со входного слоя сразу направляются на выходной слой, который, собственно говоря, не только преобразует сигнал, но и сразу же выдаёт ответ. Как уже было сказано, 1-й входной слой только принимает и распределяет сигналы, а нужные вычисления происходят уже во втором слое. Входные нейроны являются объединёнными с основным слоем с помощью синапсов с разными весами, обеспечивающими качество связей.
2. **Многослойная нейронная сеть**. Здесь, помимо выходного и входного слоёв, имеются ещё несколько скрытых промежуточных слоёв. Число этих слоёв зависит от степени сложности нейронной сети. Она в большей степени напоминает структуру биологической нейронной сети. Такие виды были разработаны совсем недавно, до этого все процессы были реализованы с помощью однослойных нейронных сетей. Соответствующие решения обладают большими возможностями, если сравнивать с однослойными, ведь в процессе обработки данных каждый промежуточный слой — это промежуточный этап, на котором осуществляется обработка и распределение информации.

Кроме количества слоёв, нейронные сети можно классифицировать по направлению распределения информации по синапсам между нейронами:

1. **Нейросети прямого распространения (однонаправленные)**. В этой структуре сигнал перемещается строго по направлению от входного слоя к выходному. Движение сигнала в обратном направлении не осуществляется и в принципе невозможно. Сегодня разработки этого плана распространены широко и на сегодняшний день успешно решают задачи распознавания образов, прогнозирования и кластеризации.
2. **Рекуррентные нейронные сети (с обратными связями)**. Здесь сигнал двигается и в прямом, и в обратном направлении. В итоге результат выхода способен возвращаться на вход. Выход нейрона определяется весовыми характеристиками и входными сигналами, плюс дополняется предыдущими выходами, снова вернувшимися на вход. Этим нейросетям присуща функция кратковременной памяти, на основании чего сигналы восстанавливаются и дополняются во время их обработки.
3. **Радиально-базисные функции**.
4. **Самоорганизующиеся карты**.

**Свёрточные нейронные сети** *(convolutional neural networks, CNN)* и глубинные свёрточные нейронные сети *(deep convolutional neural networks, DCNN)* сильно отличаются от других видов сетей. Обычно они используются для обработки изображений, реже для аудио. Типичным способом применения CNN является классификация изображений: если на изображении есть кошка, сеть выдаст «кошка», если есть собака — «собака». Такие сети обычно используют «сканер», не парсящий все данные за один раз. Например, если у вас есть изображение 200×200, вы не будете сразу обрабатывать все 40 тысяч пикселей. Вместо это сеть считает квадрат размера 20 x 20 (обычно из левого верхнего угла), затем сдвинется на 1 пиксель и считает новый квадрат, и т.д. Эти входные данные затем передаются через свёрточные слои, в которых не все узлы соединены между собой. Эти слои имеют свойство сжиматься с глубиной, причём часто используются степени двойки: 32, 16, 8, 4, 2, 1. На практике к концу CNN прикрепляют FFNN для дальнейшей обработки данных. Такие сети называются глубинными (DCNN).

Методы распознавания

Системы распознавания реализуются как классификаторы, использующие различные методы: шаблонные (растровые); признаковые; структурные. В классификаторе *шаблонного* типа с помощью критерия сравнения определяется, какой из шаблонов выбрать из базы. Самый простой критерий - минимум точек, отличающих шаблон от исследуемого изображения. К достоинствам шаблонного классификатора относятся хорошее распознавание дефектных символов («разорванных» или «склеенных»), простота и высокая скорость распознавания. Недостатком является необходимость настройки системы на типы и размеры шрифтов.

В *признаковых* классификаторах анализ проводится только по набору чисел или признаков, вычисляемых по изображению. Этот метод позволяет распознавать различные начертания символов, т.е. различные подчерки шрифты и т.д. Этот метод неизбежно вызывает некоторую потерю информации, так как используется топологическое представление, отражающее информацию о взаимном расположении структурных элементов символа. Эти данные могут быть представлены в графовой форме. При этом данный метод обеспечивает инвариантность относительно типов и размеров шрифтов. Недостатками являются трудность распознавания дефектных символов и медленная работа.

Основой *структурно-пятенного* метода является структурно-пятенный эталон. Он имеет вид набора пятен с попарными отношениями между ними. Данное представление нечувствительно к различным начертаниям и дефектам символов. Алгоритм основан на сочетании шаблонного и структурного методов распознавания образов. При анализе образца выделяются ключевые точки объекта — так называемые «пятна». В качестве пятен, например, могут выступать: концы линий; узлы, где сходятся несколько линий; места изломов линий; места пересечения линий; крайние точки. После выделения характерных точек определяются связи между ними — отрезок, или дуга. Таким образом, итоговое описание представляет собой *граф*, который и служит объектом поиска в библиотеке «структурно-пятенных эталонов». При поиске устанавливается соответствие между ключевыми точками образца и эталона, после чего определяется степень деформации связей, необходимая, чтобы привести искомый объект к сравниваемому эталонному образцу. При этом, меньшая степень необходимой деформации предполагает большую вероятность правильного распознавания символа. Далее рассмотрим этапы обработки изображения.

1. Предобработка. На этом этапе выполняются следующие задачи: повышение качества изображения за счет фильтрации, шумоподавления и других операций, имеющих своей целью повысить качество изображения. На этом этапе происходит очистка изображения от дефектов сканирования. В частности, в самом начале работы к изображению в целях шумоподавления часто применяется фильтр Гаусса. Важную роль играет пороговая бинаризация, то есть перевод изображения в чёрно-белый формат из цветного или оттенков серого. Это позволяет резко разделить текст и фон, упрощает в дальнейшем применение многих алгоритмов, а также избавляет от некоторых шумов на изображении. При этом используется гистограмма яркости изображения текста, на котором наблюдается два пика: высокий пик, соответствующий белому фону, то есть цвету бумаги, и пик в области тёмных пикселей, соответствующих яркости символов текста.
2. Выделение региона интереса. На этом этапе на бинаризованном изображении выделяется непосредственно область, на которой находится распознаваемый текст, и отбрасываются элементы, текстом не являющиеся. К ним относятся такие объекты, как кляксы, пятна на бумаге, не удалённые в процессе бинаризации, картинки и др. Для их удаления можно, например, выделять компоненты связности на изображении, вычислять геометрические признаки и на их основе классифицировать компоненту связности как часть текста или дефект, используя методы машинного обучения или эвристики.
3. Сегментация и нормализация текста. На этом этапе текст разделяется, или сегментируется, на удобные для анализа составные части. Наиболее естественными действиями на данном этапе является разделение текста на отдельные строки (сегментация строк) и разделение строк на слова (сегментация слов), а также, теоретически, разделение слов на элементарные составные части. Кроме того, на данном этапе проводится нормализация текста приведение выделенных составных частей к некоторому стандартному виду для снижения вариативности и упрощения распознавания.

*Сегментация строк.* Задача сегментации (разделения) строк в машинопечатных документах на сегодняшний день считается полностью решённой. Но в задачах при разделении строк в общем случае возникают сложности, не позволяющие напрямую применять алгоритмы, пригодные для машинопечатных текстов:

- строки не только могут не являться параллельными, но и могут изгибаться;

- различные строки могут быть слишком близки, а элементы текста, принадлежащего различным строкам, могут налагаться друг на друга.

Например, если коэффициент формы области (отношения квадрата её периметра к площади) меньше некоторого значения, а площадь больше некоторого значения, то это с большой вероятностью дефект (т.к. рукописный текст обычно является некоторой кривой)

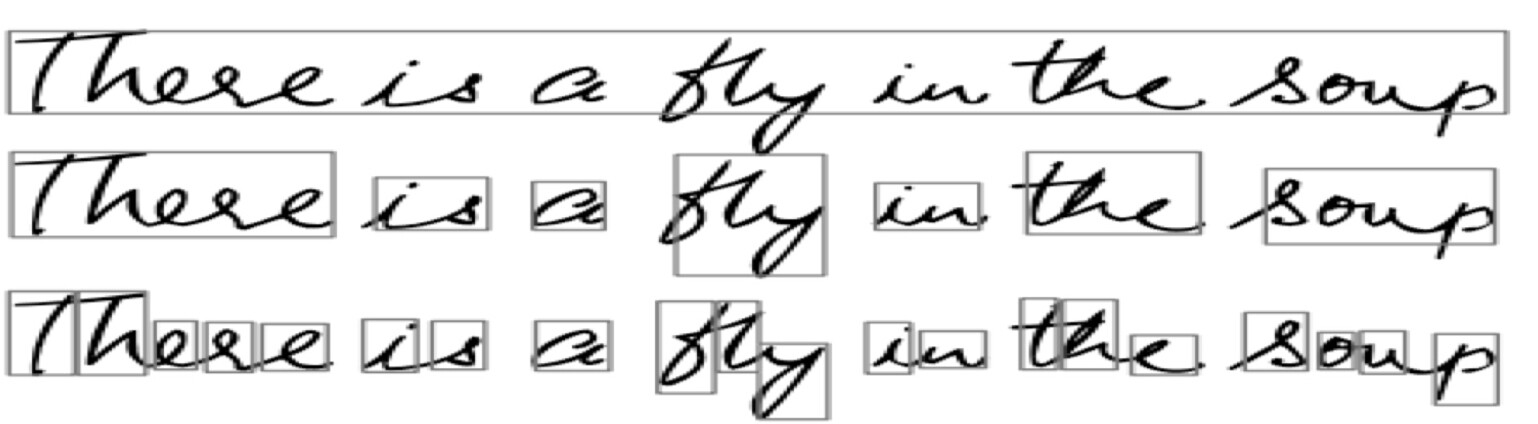


Рисунок 6 - сегментация текста: сначала выделяется строка, затем слова и буквы

*Выделение базовых линий*. Эти методы основаны на идее, что человек пишет либо по, либо поверх некоторой воображаемой линии. Данные методы пытаются аппроксимировать эту линию, а затем восстановить по ней строку. В преобразовании Хафа выделяются прямые, если они не слишком искривлены. Преобразование Хафа применяется к центрам компонент связности пикселей текста. Такой подход требует, чтобы строки текста были близки к прямым, но зато позволяет выделять строки, расположенные в произвольном месте и идущие под произвольными углами.

*Пересечение элементов различных строк* представляет собой проблему не только сегментации строк, но и распознавания текста, так как отнесение элемента к неправильной строке очевидно ухудшает его распознаваемость. Пересекающиеся компоненты являются проблемой для методов горизонтальной проекции (так как они увеличивают значение профиля проекции в тех местах, где должен быть его минимум) группировочных методов (так как они используют связные компоненты пикселей текста для построения строк), но слабо влияют на некоторые методы выделения базовых линий. Для поиска пересекающихся элементов из различных строк можно использовать такие признаки, как размер компонент связности текста, факт отнесения одной компоненты к нескольким строкам или, напротив, не относящимся ни к какой строке. После нахождения таких сомнительных компонент нужно определить, относятся ли они к какой-то строке или же их нужно декомпозировать на элементы, относящиеся к разным строкам. Такая вертикальная декомпозиция компонент сложная задача. Простое решение заключается в разрезании компоненты на части горизонтальными линиями, но можно применить и более тонкие подходы, например, выделение отдельных штрихов.

1. Сегментация слов. На этом этапе работы системы распознавания выделенные строки текста разделяются на отдельные слова. В отличие от машинописного текста, в котором расстояние между словами более-менее постоянно, а интервалы между символами внутри слова гораздо меньше, чем интервалы между словами, в рукописном тексте размер интервалов между словами может варьироваться в очень широких пределах. Компоненты связности текста, отнесённые к одной строке на предыдущем этапе работы системы распознавания, объединяются в слова на этом этапе.

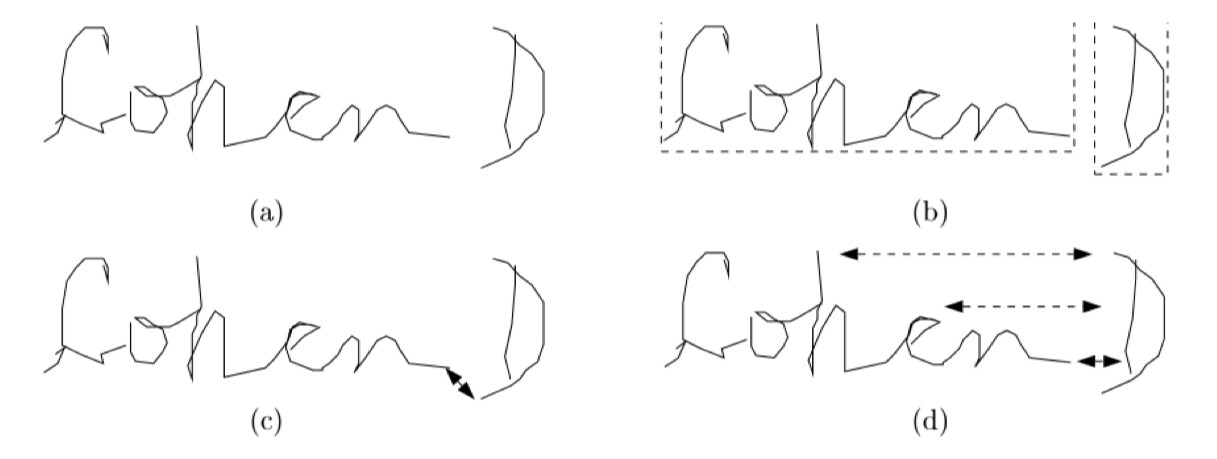
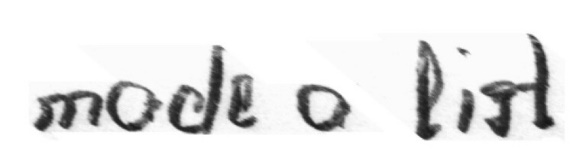
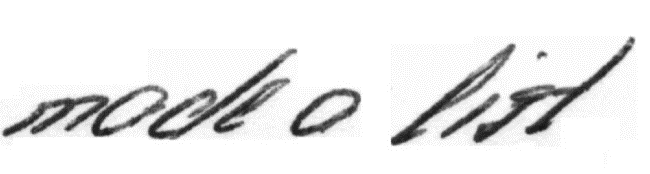


Рисунок 7 – а) исходный кусок текста, графическое изображение методов нахождения расстояния между словами: b) ​ограничивающий прямоугольник,

с) Евклидово расстояние, d) горизонтальное расстояние

1. Нормализация. В силу высокой вариативности начертания слов их распознавание является очень сложным процессом. Нормализация служит для приведения слова к некоторому стандартному виду без значительной потери информации, необходимой для распознавания. Одними из наиболее часто используемых методов нормализации является метод коррекции наклона слова от горизонтальной и вертикальной линии. Простейший метод коррекции по горизонтали состоит в выполнении максимизации его на некотором диапазоне. Существуют и другие методы нормализации, например, коррекция размера и выделение скелета текста, но они применяются реже.



а) б)

Рисунок 8 – а) слова до коррекции наклона, б) слова после коррекции наклона

1. Языковая модель. помогает алгоритму распознавания улучшить результат с помощью известных языковых словосочетаний. Языковая модель может предсказывать следующее слово на основании предыдущих и предлагать варианты с различной степенью вероятности. Например, более вероятной считается словосочетание «There are» по сравнению с «Their are». Такую модель можно тренировать на основании большого количества текста с расчетом повторений одного слова за другими. Конечно, нет сильной уверенности, что языковая модель сможет во всех случаях предсказывать следующее слово для любого текста, поэтому модель тренируют на тексте той же тематики, что и распознаваемый. Использование языковой модели также опционально.
2. Префиксное дерево. Алгоритм поиска слова по префиксу достаточно быстрый и простой для понимания. Как можно догадаться, используется структура дерево, где рёбра — буквы, а вершина содержит признак слова. Таким образом, слово представляется в виде пути от корневого узла (первой буквы слова) через рёбра — составные буквы слова к вершине с признаком слова.

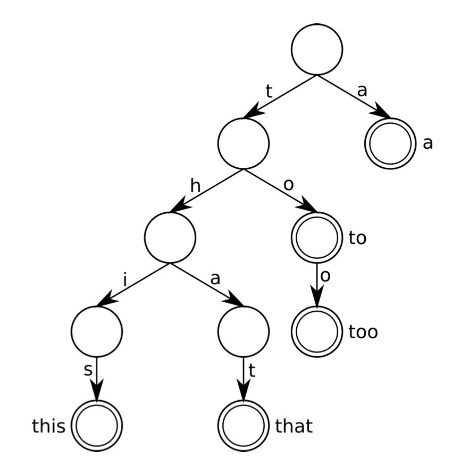


Рисунок 9 - представление слов to, too, a, this, that в префиксном дереве

# РАЗРАБОТКА ПРОГРАММНОЙ ЧАСТИ

Для реализации нейронной сети будут использоваться следующие инструменты: Python 3, TensorFlow 1.3, numpy и OpenCV.

Обзор модели

Предлагаемая система использует исскуственную нейронную сеть (Artifical Neural Network - ANN), при этом для извлечения объектов используются многочисленные слои сверточных слоев нейросети (Convolutional NN - CNN) с входной фотографии. Затем выход этих слоев подается в рекуррентные слой нейросети (Recurrent NN - RNN). RNN распространяет информацию через последовательность. Вывод RNN содержит вероятности для каждого символа в последовательности. Для прогнозирования конечного текста реализуются алгоритмы декодирования в выход RNN. Функции коннекционистской временной классификации (Connectionist Temporal Classification - CTC) отвечают за декодирование вероятностей в окончательный текст. Для повышения точности распознавания декодирование может также использовать языковую модель. CTC используется для получения знаний; выход RNN представляет собой матрицу, содержащую вероятности символов для каждого временного шага. Алгоритм декодирования CTC преобразует эти символические вероятности в окончательный текст. Затем, чтобы повысить точность, используется алгоритм, который продолжает поиск слов в словаре. Однако время, необходимое для поиска фраз, зависит от размеров словаря, и он не может декодировать произвольные символьные строки, включая числа. На рис. 10 показан обзор нашей системы распознавания рукописного текста (Handwritten Text Recognition – HTR).

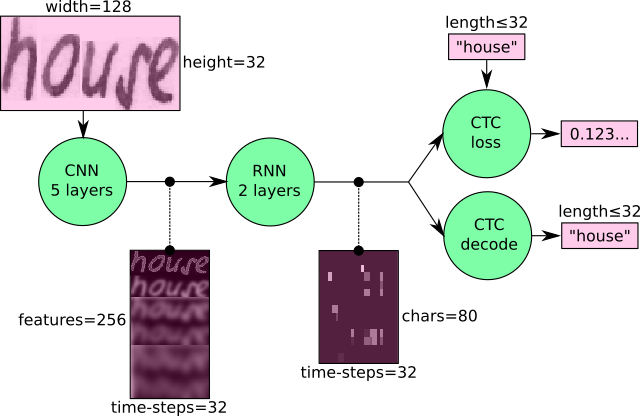


Рисунок 10 - обзор операций НС (зеленый) и потока данных через НС (розовый).

Мы можем рассматривать NN более формальным образом как функцию (см. рисунок 11), которая отображает изображение (или матрицу) M размера W × H в последовательность символов (c1, c2, …) длиной от 0 и L. Как видите, текст распознается на уровне символов, поэтому слова или тексты, не содержащиеся в обучающих данных, также могут быть распознаны (если отдельные символы правильно классифицированы).

Image for post

Рисунок 11 - NN, записанный как математическая функция, которая отображает изображение M в последовательность символов (c1, c2, …)

Операции

**CNN**: входное изображение подается в слои CNN. Эти слои отвечают за извлечение объектов. Есть 5х5 фильтры в первом и втором слоях и фильтры 3х3 в последних трех слоях. Они также содержат нелинейную функцию RELU и максимальный объединяющий слой, который суммирует изображения и делает их меньше, чем входные данные. Хотя высота изображения уменьшается в 2 раза в каждом слое, карты объектов (каналы) добавляются таким образом, чтобы получить выходную карту объектов (или последовательность) размером от 32x256. 5 слоев CNN сопоставляют входное изображение с последовательностью признаков размером 32x256

**RNN**: последовательность признаков содержит 256 признаков или симптомов на каждом временном шаге. Соответствующая информация распространяется РНН через эти серии. LSTM — это один из известных алгоритмов RNN, который переносит информацию на большие расстояния и более эффективное обучение, чем типичные РНН. Выходная последовательность RNN сопоставляется с матрицей 32х80. 2 слоя LSTM с 256 элементами распространяют информацию по последовательности и отображают последовательность в матрицу размером 32x80. Каждый элемент матрицы представляет собой оценку для одного из 80 символов на одном из 32 временных шагов.

**CTC**: получает выходную матрицу RNN и прогнозируемый текст в процессе обучения нейронной сети, а также определяет величину потерь. CTC получает только матрицу после обработки и декодирует ее в окончательный текст. Длина основного текста и известного текста не должна превышать 32 символов. Уровень CTC либо вычисляет значение потерь, заданное матрицей и текстом основной истины (при обучении), либо декодирует матрицу до конечного текста с декодированием наилучшего пути или декодированием поиска луча (при выводе)  
Размер партии устанавливается в 50.

**Данные**: Входные данные: это изображение в оттенках серого размером от 128x32. Изображения в наборе данных обычно не имеют точно такого размера, поэтому их исходный размер изменяется (без искажений) до тех пор, пока они не будут иметь ширину 128 и высоту 32. Затем изображение копируется в целевое (белое) изображение размером от 128x32 дюймов. Этот процесс показан на рисунке 12. Наконец, мы нормализуем оттенки серого изображения, что упрощает задачу для NN. Увеличение данных может быть легко интегрировано путем копирования изображения в случайные позиции вместо выравнивания его по левому краю или путем случайного изменения размера изображения.

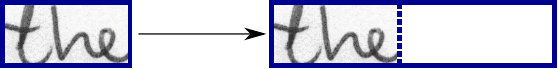


Рисунок 12 – слева: изображение из набора данных произвольного размера. Оно масштабируется под размер целевого изображения 128×32, пустая часть целевого изображения заполняется белым цветом.

Реализация с использованием TensorFlow

Реализация состоит из 4 модулей:

1. SamplePreprocessor.py: подготавливает изображения из набора данных IAM для NN (см. исходный код в приложении 1);
2. DataLoader.py: считывает образцы, помещает их в пакеты и предоставляет интерфейс-итератор для просмотра данных (см. исходный код в приложении 2);
3. Model.py: создает модель, как описано выше, загружает и сохраняет модели, управляет сеансами TF и предоставляет интерфейс для обучения и вывода (см. исходный код в приложении 3);
4. main.py: объединяет все ранее упомянутые модули (см. исходный код в приложении 4);
5. analyze.py: анализ файла изображения. Вычисление на релевантность пикселей для правильного прогноза, проверка на то, инвариантна ли модель к горизонтальным переводам текста (см. исходный код в приложении 5).

Мы смотрим только на Model.py, так как другие исходные файлы связаны с базовым вводом-выводом файла (DataLoader.py) и обработкой изображений (SamplePreprocessor.py).

**CNN**:

Для каждого слоя CNN создаётся ядро размером k×k, которое будет использоваться в операции свертки.



Затем результат свертки передаётся в операцию RELU, а затем снова в слой объединения с размером px×py и размером шага sx×sy.



Эти шаги повторяются для всех слоев в цикле for.

**RNN**:

Создаются и складываются два слоя RNN по 256 единиц каждый.



Затем из него создаётся двунаправленный RNN, так что входная последовательность будет проходить спереди назад и наоборот. В результате мы получаем две выходные последовательности fw и bw размером 32×256, которые мы позже объединяем вдоль оси признаков, чтобы сформировать последовательность размером 32×512. Наконец, он сопоставляется с выходной последовательностью (или матрицей) размером 32×80, которая подается в слой CTC.





**CTC**:

Для расчета потерь мы вводим в операцию как текст основной истины, так и матрицу. Текст основной истины кодируется как разреженный тензор. Длина входных последовательностей должна быть передана обеим операциям CTC.







Теперь у нас есть все входные данные для создания операции потери и операции декодирования.







**Тренировка**:

Среднее значение потерь элементов пакета используется для обучения NN: оно подается в оптимизатор, такой как RMSProp.



**Выходные данные CNN**: на рисунке 13 показаны выходные данные уровней CNN, которые представляют собой последовательность длиной 32. Каждая запись содержит 256 характеристик. Конечно, эти характеристики дополнительно обрабатываются уровнями RNN, однако некоторые из них уже показывают высокую корреляцию с определенными высокоуровневыми свойствами входного изображения: есть характеристики, которые имеют высокую корреляцию с символами (например, «e»), или с повторяющимися символами (например, «tt»), или со свойствами символов, такими как петли (как содержащиеся в написанных от руки «l» или «e»).

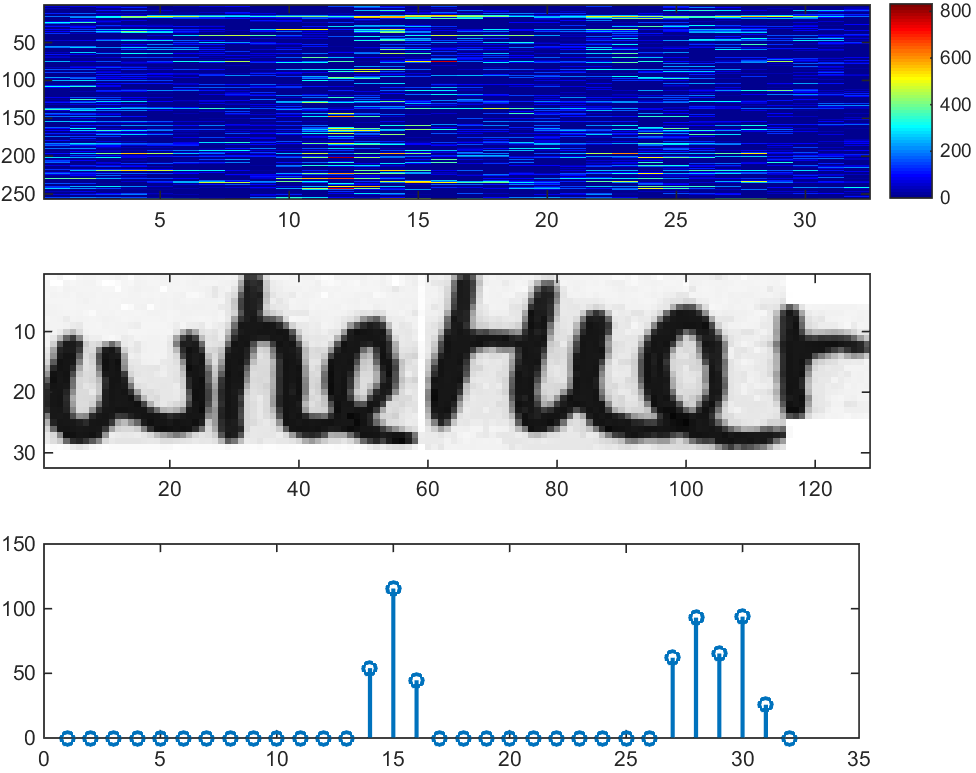


Рисунок 13 - вверху: 256 характеристик на временной шаг вычисляются слоями CNN. В центре: входное изображение. Внизу: график 32-го объекта, который имеет высокую корреляцию с появлением символа «e» на изображении.

**Выходные данные RNN**: на рис. 14 показана визуализация выходной матрицы RNN для изображения, содержащего текст «little». Матрица, показанная на самом верхнем графике, содержит оценки для символов, включая пустую метку CTC в качестве последней (80-й) записи. Другие элементы матрицы, сверху вниз, соответствуют следующим символам: «“!”#&’()\*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgh ijklmnopqrstuvwxyz». Можно увидеть, что в большинстве случаев символы предсказываются точно в том положении, в котором они появляются на изображении (например, сравните положение «i» на изображении и на графике). Не выравнивается только последний символ «е». Но это нормально, поскольку операция CTC не требует сегментации и не заботится об абсолютных позициях. Из самого нижнего графика, показывающего оценки для символов «l», «i», «t», «e» и пустой метки CTC, текст можно легко декодировать: мы просто берем наиболее вероятный символ из каждого временного шага, это формирует так называемый лучший путь «best path», затем мы отбрасываем повторяющиеся символы и, наконец, все пробелы: “l---ii-tt-l-…-e”→”l---i--t-t--l-…-e”→ “little”.

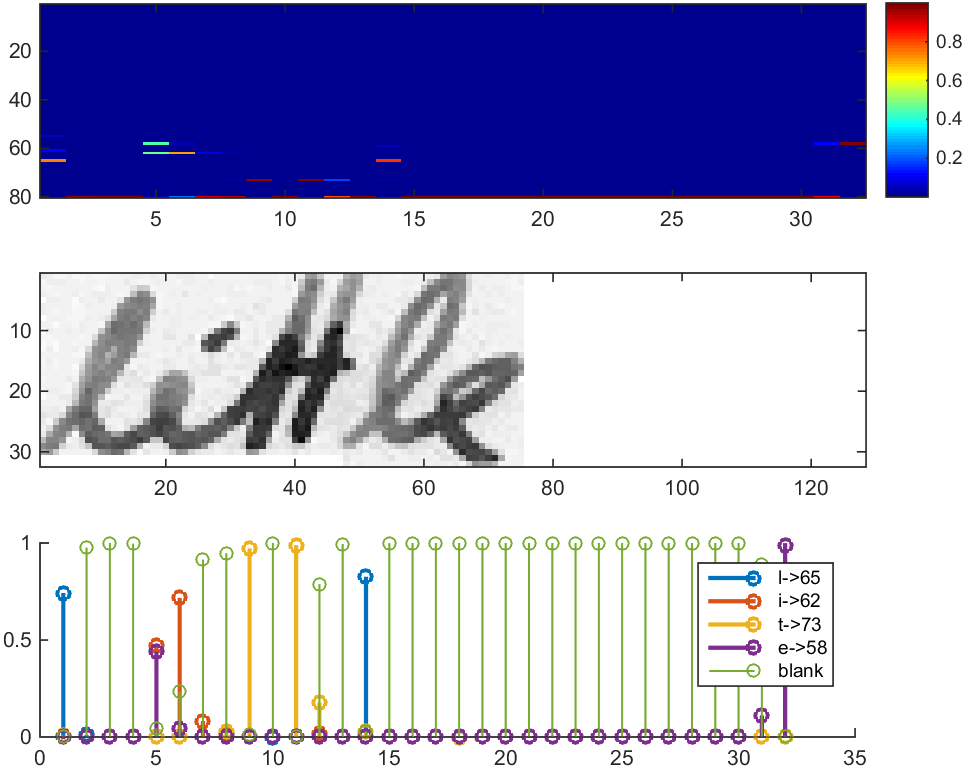


Рисунок 14 - вверху: выходная матрица слоев RNN. В центре: входное изображение. Внизу: вероятности символов «l», «i», «t», «e» и пустой метки CTC.

# МНОГОФАКТОРНЫЙ ЭКСПЕРИМЕНТ

Разработка метрики измерения точности распознавания рукописного текста

В следствие того, что рукописный текст не имеет каких-то характерных черт, как, например вид шрифта (у печатных текстов), предполагается измерять точность измерения в конкретном случае посредством сравнения распознанного программой текста с оригиналом. Допустим мы пытаемся распознать слово «отлично». В данном слове 7 букв, значит 100-процентная точность его распознавания будет достигнута при условии, что слово распознано полностью, без допуска орфографических ошибок. Если 3 буквы распознаны не, верно, тогда точность распознавания будет равна 100 – 3\*(100/7) = 57,14%, т. е. процентная точность высчитывается путём вычитания процента неправильно распознанных букв в слове из 100%.

Также потенциально могут возникнуть ситуации, когда программа не считывает некоторые символы или обнаруживает лишний символ. В случае не считывания некоторых символов, например вместо «отлично» программа считывает «отичн», предполагается так же, как и в случае неправильного распознавания символа. Если же распознан лишний символ, то считывание слова примем за неудачное, с точностью 0%.

Обучение нейронной сети

Загрузчик данных ожидает набор данных IAM в каталоге data /. Для обучения были загружены файлы words/words.tgz. и ascii/words.txt с сайта IAM Handwriting Database. Общее время обучения составило 3 часа 1 минуту.

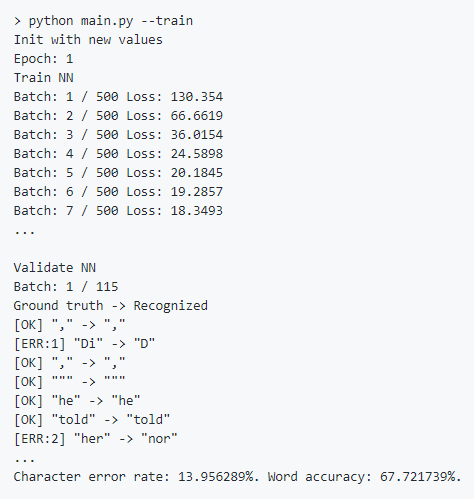


Рисунок 15 – в конце обучения точность распознавания слов составляет 67,72%,

а коэффициент 13,96%

Анализ

Запускаем analysis.py со следующими аргументами, чтобы проанализировать файл изображения data/analysis.png с достоверным текстом "are":

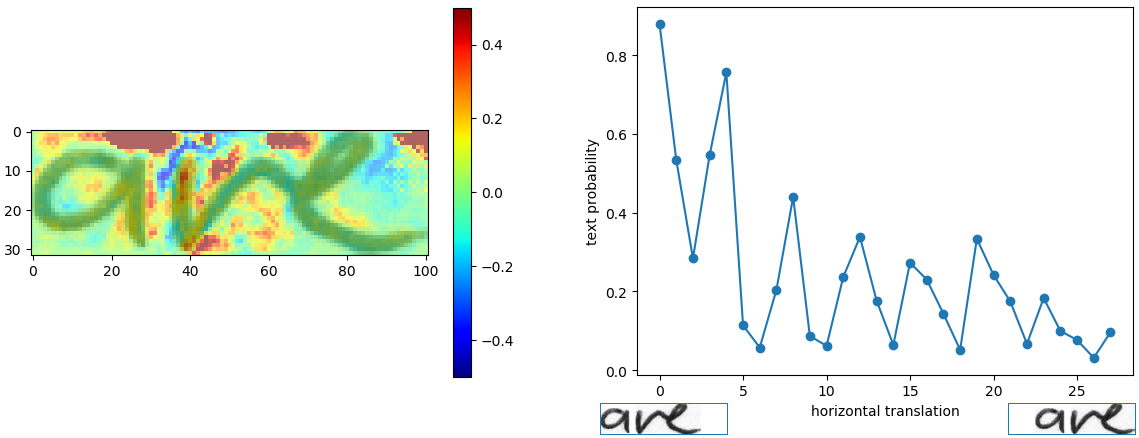
--relevance: вычислить релевантность пикселей для правильного прогноза.

--invariance: проверить, инвариантна ли модель к горизонтальным переводам текста.

Аргумент не предоставлен: покажите результаты.

Результаты показаны на графиках ниже. Релевантность пикселей (слева) показывает, как пиксель влияет на оценку для правильного класса. Красные пиксели голосуют за правильный класс, а синие пиксели голосуют против правильного класса. Можно увидеть, что пустое пространство над вертикальными линиями на изображениях важно для классификатора, чтобы решить, не использовать ли символ «i» с его точкой в верхнем индексе. Если точку над «a» (красная область на графике), и вы получите «aive» вместо «are».

Второй график (справа) показывает, как изменяется вероятность достоверного текста, когда текст смещается вправо. Как можно видеть, модель не является инвариантной к трансляции, поскольку все обучающие изображения из IAM выровнены по левому краю. Добавление увеличения данных, которое использует случайное выравнивание текста, может улучшить инвариантность перевода модели. Более подробную информацию можно найти в этой статье.



а) б)

Рисунок 16 – а) график релевантности пикселей,

б) график изменения вероятности достоверного распознавания текста

SimpleHTR эксперименты

SimpleHTR модель – это обучение, валидация и тестирование на двух различных датасетах. Набор данных был разделен на два подмножества: 90% для обучения и 10% для проверки обученной модели.

**Первый Набор Данных:** для обучения на собранных данных была обработана модель SimpleHTR, в которой есть 42 названия стран и городов с различными узорами почерка. Такие данные были увеличены в 10 раз. Были проведены два теста: с выравниванием курсивных слов и без выравнивания. После изучения были получены значения по валидации данных, представленных в Таблице ниже.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | выравнивание скорописи | | нет выравнивания | |
| CER | WAR | CER | WAR |
| bestpath | 19.13 | 52.55 | 17.97 | 57.11 |
| beamsearch | 18.99 | 53.33 | 17.73 | 58.33 |
| wordbeamsearch | 16.38 | 73.55 | 15.78 | 75.11 |

Эта таблица показывает точность распознавания SimpleHTR для раличных методов декодирования (bestpath, beamsearch, wordbeamsearch). Декодирование наилучшего пути использует только выход NN и вычисляет оценку, принимая наиболее вероятный символ в каждой позиции. Поиск луча также использует только выход NN, но он использует больше данных из него и, следовательно, обеспечивает более детальный результат. Поиск луча с character-LM также забивает символьные последовательности, которые еще больше повышают исход.

Результаты обучения можно посмотреть на рисунке ниже:

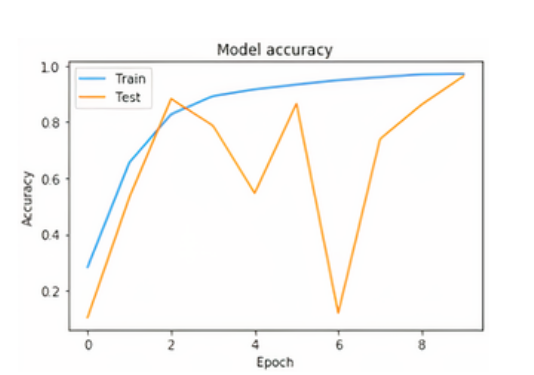


Рисунок 17 - результаты эксперимента с использованием SimpleHTR (lr=0,01): точность модели

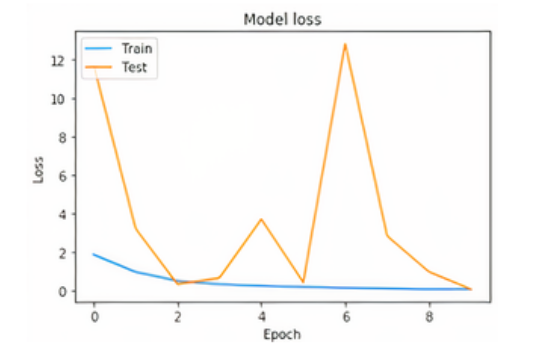
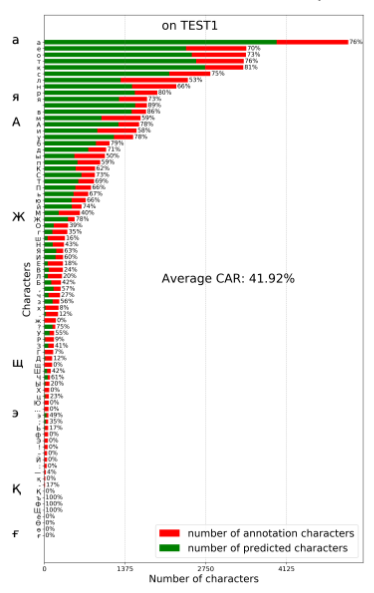
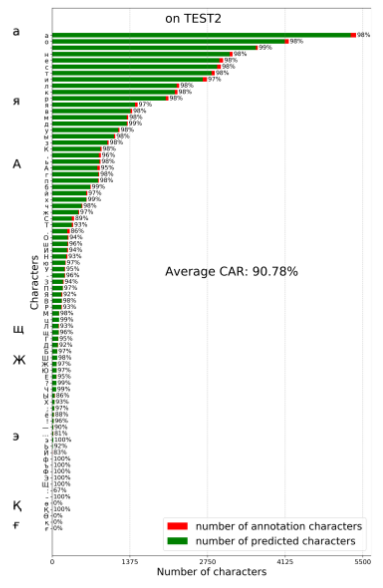


Рисунок 18 - результаты эксперимента с использованием SimpleHTR (lr=0,01): погрешность модели.

**Второй набор данных (HKR Dataset):** Модель SimpleHTR показала в первом тесте набора данных 20,13% символьной ошибки (CER) и второго набора данных 1,55% CER. Мы также оценили модель SimpleHTR по каждому показателю точности символов (рисунок ниже). Частота ошибок в словах (WER) составил 58,97% для теста 1 и 11,09% для теста 2. Результат, например TEST2 показывает, что модель может распознавать слова, которые существуют в обучающем наборе данных, но имеют полностью различные стили почерка. Набор данных TEST1 показывает, что результат не является хорошим, когда модель распознает слова, которые не существуют в обучении и наборы данных проверки.



а)



б)

Рисунок 19 – коэффициент точности символов а) в первом тесте, б) во втором тесте

Следующий эксперимент проводился с моделью LineHTR, обученной на данных за 100 эпох. Эта модель продемонстрировала производительность со средним CAR 29,86% и 86,71% для наборов данных TEST1 и TEST2 соответственно (рисунок ниже). Здесь также наблюдается аналогичная тенденция переобучения обучающих данных.

# ЗАКЛЮЧЕНИЕ

Мною была изучена тема разработки метрики и исследования точности распознавания рукописного текста, её основные методы.

По данным, полученным во время исследования, можно сказать, что использование распознавания рукописного текста может использоваться во многих сферах. Например, для оцифровки рукописных текстов, имеющих историческую ценность, для сохранения их в электронных библиотеках.

# ПРИЛОЖЕНИЕ

Код на Python SamplePreprocessor.py

|  |
| --- |
| from \_\_future\_\_ import division |
|  | from \_\_future\_\_ import print\_function |
|  |  |
|  | import random |
|  |  |
|  | import cv2 |
|  | import numpy as np |
|  |  |
|  |  |
|  | def preprocess(img, imgSize, dataAugmentation=False): |
|  | "put img into target img of size imgSize, transpose for TF and normalize gray-values" |
|  |  |
|  | # there are damaged files in IAM dataset - just use black image instead |
|  | if img is None: |
|  | img = np.zeros([imgSize[1], imgSize[0]]) |
|  |  |
|  | # increase dataset size by applying random stretches to the images |
|  | if dataAugmentation: |
|  | stretch = (random.random() - 0.5) # -0.5 .. +0.5 |
|  | wStretched = max(int(img.shape[1] \* (1 + stretch)), 1) # random width, but at least 1 |
|  | img = cv2.resize(img, (wStretched, img.shape[0])) # stretch horizontally by factor 0.5 .. 1.5 |
|  |  |
|  | # create target image and copy sample image into it |
|  | (wt, ht) = imgSize |
|  | (h, w) = img.shape |
|  | fx = w / wt |
|  | fy = h / ht |
|  | f = max(fx, fy) |
|  | newSize = (max(min(wt, int(w / f)), 1), |
|  | max(min(ht, int(h / f)), 1)) # scale according to f (result at least 1 and at most wt or ht) |
|  | img = cv2.resize(img, newSize) |
|  | target = np.ones([ht, wt]) \* 255 |
|  | target[0:newSize[1], 0:newSize[0]] = img |
|  |  |
|  | # transpose for TF |
|  | img = cv2.transpose(target) |
|  |  |
|  | # normalize |
|  | (m, s) = cv2.meanStdDev(img) |
|  | m = m[0][0] |
|  | s = s[0][0] |
|  | img = img - m |
|  | img = img / s if s > 0 else img |
|  | return img |

Код на Python DataLoader.py

|  |
| --- |
| from \_\_future\_\_ import division |
|  | from \_\_future\_\_ import print\_function |
|  |  |
|  | import os |
|  | import random |
|  |  |
|  | import cv2 |
|  | import numpy as np |
|  |  |
|  | from SamplePreprocessor import preprocess |
|  |  |
|  |  |
|  | class Sample: |
|  | "sample from the dataset" |
|  |  |
|  | def \_\_init\_\_(self, gtText, filePath): |
|  | self.gtText = gtText |
|  | self.filePath = filePath |
|  |  |
|  |  |
|  | class Batch: |
|  | "batch containing images and ground truth texts" |
|  |  |
|  | def \_\_init\_\_(self, gtTexts, imgs): |
|  | self.imgs = np.stack(imgs, axis=0) |
|  | self.gtTexts = gtTexts |
|  |  |
|  |  |
|  | class DataLoader: |
|  | "loads data which corresponds to IAM format, see: http://www.fki.inf.unibe.ch/databases/iam-handwriting-database" |
|  |  |
|  | def \_\_init\_\_(self, filePath, batchSize, imgSize, maxTextLen): |
|  | "loader for dataset at given location, preprocess images and text according to parameters" |
|  |  |
|  | assert filePath[-1] == '/' |
|  |  |
|  | self.dataAugmentation = False |
|  | self.currIdx = 0 |
|  | self.batchSize = batchSize |
|  | self.imgSize = imgSize |
|  | self.samples = [] |
|  |  |
|  | f = open(filePath + 'words.txt') |
|  | chars = set() |
|  | bad\_samples = [] |
|  | bad\_samples\_reference = ['a01-117-05-02.png', 'r06-022-03-05.png'] |
|  | for line in f: |
|  | # ignore comment line |
|  | if not line or line[0] == '#': |
|  | continue |
|  |  |
|  | lineSplit = line.strip().split(' ') |
|  | assert len(lineSplit) >= 9 |
|  |  |
|  | # filename: part1-part2-part3 --> part1/part1-part2/part1-part2-part3.png |
|  | fileNameSplit = lineSplit[0].split('-') |
|  | fileName = filePath + 'words/' + fileNameSplit[0] + '/' + fileNameSplit[0] + '-' + fileNameSplit[1] + '/' + \ |
|  | lineSplit[0] + '.png' |
|  |  |
|  | # GT text are columns starting at 9 |
|  | gtText = self.truncateLabel(' '.join(lineSplit[8:]), maxTextLen) |
|  | chars = chars.union(set(list(gtText))) |
|  |  |
|  | # check if image is not empty |
|  | if not os.path.getsize(fileName): |
|  | bad\_samples.append(lineSplit[0] + '.png') |
|  | continue |
|  |  |
|  | # put sample into list |
|  | self.samples.append(Sample(gtText, fileName)) |
|  |  |
|  | # some images in the IAM dataset are known to be damaged, don't show warning for them |
|  | if set(bad\_samples) != set(bad\_samples\_reference): |
|  | print("Warning, damaged images found:", bad\_samples) |
|  | print("Damaged images expected:", bad\_samples\_reference) |
|  |  |
|  | # split into training and validation set: 95% - 5% |
|  | splitIdx = int(0.95 \* len(self.samples)) |
|  | self.trainSamples = self.samples[:splitIdx] |
|  | self.validationSamples = self.samples[splitIdx:] |
|  |  |
|  | # put words into lists |
|  | self.trainWords = [x.gtText for x in self.trainSamples] |
|  | self.validationWords = [x.gtText for x in self.validationSamples] |
|  |  |
|  | # number of randomly chosen samples per epoch for training |
|  | self.numTrainSamplesPerEpoch = 25000 |
|  |  |
|  | # start with train set |
|  | self.trainSet() |
|  |  |
|  | # list of all chars in dataset |
|  | self.charList = sorted(list(chars)) |
|  |  |
|  | def truncateLabel(self, text, maxTextLen): |
|  | # ctc\_loss can't compute loss if it cannot find a mapping between text label and input |
|  | # labels. Repeat letters cost double because of the blank symbol needing to be inserted. |
|  | # If a too-long label is provided, ctc\_loss returns an infinite gradient |
|  | cost = 0 |
|  | for i in range(len(text)): |
|  | if i != 0 and text[i] == text[i - 1]: |
|  | cost += 2 |
|  | else: |
|  | cost += 1 |
|  | if cost > maxTextLen: |
|  | return text[:i] |
|  | return text |
|  |  |
|  | def trainSet(self): |
|  | "switch to randomly chosen subset of training set" |
|  | self.dataAugmentation = True |
|  | self.currIdx = 0 |
|  | random.shuffle(self.trainSamples) |
|  | self.samples = self.trainSamples[:self.numTrainSamplesPerEpoch] |
|  |  |
|  | def validationSet(self): |
|  | "switch to validation set" |
|  | self.dataAugmentation = False |
|  | self.currIdx = 0 |
|  | self.samples = self.validationSamples |
|  |  |
|  | def getIteratorInfo(self): |
|  | "current batch index and overall number of batches" |
|  | return (self.currIdx // self.batchSize + 1, len(self.samples) // self.batchSize) |
|  |  |
|  | def hasNext(self): |
|  | "iterator" |
|  | return self.currIdx + self.batchSize <= len(self.samples) |
|  |  |
|  | def getNext(self): |
|  | "iterator" |
|  | batchRange = range(self.currIdx, self.currIdx + self.batchSize) |
|  | gtTexts = [self.samples[i].gtText for i in batchRange] |
|  | imgs = [ |
|  | preprocess(cv2.imread(self.samples[i].filePath, cv2.IMREAD\_GRAYSCALE), self.imgSize, self.dataAugmentation) |
|  | for i in batchRange] |
|  | self.currIdx += self.batchSize |
|  | return Batch(gtTexts, imgs) |

Код на Python Model.py

|  |
| --- |
| from \_\_future\_\_ import division |
|  | from \_\_future\_\_ import print\_function |
|  |  |
|  | import numpy as np |
|  | import os |
|  | import sys |
|  | import tensorflow as tf |
|  |  |
|  | # Disable eager mode |
|  | tf.compat.v1.disable\_eager\_execution() |
|  |  |
|  |  |
|  | class DecoderType: |
|  | BestPath = 0 |
|  | BeamSearch = 1 |
|  | WordBeamSearch = 2 |
|  |  |
|  |  |
|  | class Model: |
|  | "minimalistic TF model for HTR" |
|  |  |
|  | # model constants |
|  | batchSize = 50 |
|  | imgSize = (128, 32) |
|  | maxTextLen = 32 |
|  |  |
|  | def \_\_init\_\_(self, charList, decoderType=DecoderType.BestPath, mustRestore=False, dump=False): |
|  | "init model: add CNN, RNN and CTC and initialize TF" |
|  | self.dump = dump |
|  | self.charList = charList |
|  | self.decoderType = decoderType |
|  | self.mustRestore = mustRestore |
|  | self.snapID = 0 |
|  |  |
|  | # Whether to use normalization over a batch or a population |
|  | self.is\_train = tf.compat.v1.placeholder(tf.bool, name='is\_train') |
|  |  |
|  | # input image batch |
|  | self.inputImgs = tf.compat.v1.placeholder(tf.float32, shape=(None, Model.imgSize[0], Model.imgSize[1])) |
|  |  |
|  | # setup CNN, RNN and CTC |
|  | self.setupCNN() |
|  | self.setupRNN() |
|  | self.setupCTC() |
|  |  |
|  | # setup optimizer to train NN |
|  | self.batchesTrained = 0 |
|  | self.learningRate = tf.compat.v1.placeholder(tf.float32, shape=[]) |
|  | self.update\_ops = tf.compat.v1.get\_collection(tf.compat.v1.GraphKeys.UPDATE\_OPS) |
|  | with tf.control\_dependencies(self.update\_ops): |
|  | self.optimizer = tf.compat.v1.train.RMSPropOptimizer(self.learningRate).minimize(self.loss) |
|  |  |
|  | # initialize TF |
|  | (self.sess, self.saver) = self.setupTF() |
|  |  |
|  | def setupCNN(self): |
|  | "create CNN layers and return output of these layers" |
|  | cnnIn4d = tf.expand\_dims(input=self.inputImgs, axis=3) |
|  |  |
|  | # list of parameters for the layers |
|  | kernelVals = [5, 5, 3, 3, 3] |
|  | featureVals = [1, 32, 64, 128, 128, 256] |
|  | strideVals = poolVals = [(2, 2), (2, 2), (1, 2), (1, 2), (1, 2)] |
|  | numLayers = len(strideVals) |
|  |  |
|  | # create layers |
|  | pool = cnnIn4d # input to first CNN layer |
|  | for i in range(numLayers): |
|  | kernel = tf.Variable( |
|  | tf.random.truncated\_normal([kernelVals[i], kernelVals[i], featureVals[i], featureVals[i + 1]], |
|  | stddev=0.1)) |
|  | conv = tf.nn.conv2d(input=pool, filters=kernel, padding='SAME', strides=(1, 1, 1, 1)) |
|  | conv\_norm = tf.compat.v1.layers.batch\_normalization(conv, training=self.is\_train) |
|  | relu = tf.nn.relu(conv\_norm) |
|  | pool = tf.nn.max\_pool2d(input=relu, ksize=(1, poolVals[i][0], poolVals[i][1], 1), |
|  | strides=(1, strideVals[i][0], strideVals[i][1], 1), padding='VALID') |
|  |  |
|  | self.cnnOut4d = pool |
|  |  |
|  | def setupRNN(self): |
|  | "create RNN layers and return output of these layers" |
|  | rnnIn3d = tf.squeeze(self.cnnOut4d, axis=[2]) |
|  |  |
|  | # basic cells which is used to build RNN |
|  | numHidden = 256 |
|  | cells = [tf.compat.v1.nn.rnn\_cell.LSTMCell(num\_units=numHidden, state\_is\_tuple=True) for \_ in |
|  | range(2)] # 2 layers |
|  |  |
|  | # stack basic cells |
|  | stacked = tf.compat.v1.nn.rnn\_cell.MultiRNNCell(cells, state\_is\_tuple=True) |
|  |  |
|  | # bidirectional RNN |
|  | # BxTxF -> BxTx2H |
|  | ((fw, bw), \_) = tf.compat.v1.nn.bidirectional\_dynamic\_rnn(cell\_fw=stacked, cell\_bw=stacked, inputs=rnnIn3d, |
|  | dtype=rnnIn3d.dtype) |
|  |  |
|  | # BxTxH + BxTxH -> BxTx2H -> BxTx1X2H |
|  | concat = tf.expand\_dims(tf.concat([fw, bw], 2), 2) |
|  |  |
|  | # project output to chars (including blank): BxTx1x2H -> BxTx1xC -> BxTxC |
|  | kernel = tf.Variable(tf.random.truncated\_normal([1, 1, numHidden \* 2, len(self.charList) + 1], stddev=0.1)) |
|  | self.rnnOut3d = tf.squeeze(tf.nn.atrous\_conv2d(value=concat, filters=kernel, rate=1, padding='SAME'), axis=[2]) |
|  |  |
|  | def setupCTC(self): |
|  | "create CTC loss and decoder and return them" |
|  | # BxTxC -> TxBxC |
|  | self.ctcIn3dTBC = tf.transpose(a=self.rnnOut3d, perm=[1, 0, 2]) |
|  | # ground truth text as sparse tensor |
|  | self.gtTexts = tf.SparseTensor(tf.compat.v1.placeholder(tf.int64, shape=[None, 2]), |
|  | tf.compat.v1.placeholder(tf.int32, [None]), |
|  | tf.compat.v1.placeholder(tf.int64, [2])) |
|  |  |
|  | # calc loss for batch |
|  | self.seqLen = tf.compat.v1.placeholder(tf.int32, [None]) |
|  | self.loss = tf.reduce\_mean(input\_tensor=tf.compat.v1.nn.ctc\_loss(labels=self.gtTexts, inputs=self.ctcIn3dTBC, |
|  | sequence\_length=self.seqLen, |
|  | ctc\_merge\_repeated=True)) |
|  |  |
|  | # calc loss for each element to compute label probability |
|  | self.savedCtcInput = tf.compat.v1.placeholder(tf.float32, |
|  | shape=[Model.maxTextLen, None, len(self.charList) + 1]) |
|  | self.lossPerElement = tf.compat.v1.nn.ctc\_loss(labels=self.gtTexts, inputs=self.savedCtcInput, |
|  | sequence\_length=self.seqLen, ctc\_merge\_repeated=True) |
|  |  |
|  | # decoder: either best path decoding or beam search decoding |
|  | if self.decoderType == DecoderType.BestPath: |
|  | self.decoder = tf.nn.ctc\_greedy\_decoder(inputs=self.ctcIn3dTBC, sequence\_length=self.seqLen) |
|  | elif self.decoderType == DecoderType.BeamSearch: |
|  | self.decoder = tf.nn.ctc\_beam\_search\_decoder(inputs=self.ctcIn3dTBC, sequence\_length=self.seqLen, |
|  | beam\_width=50) |
|  | elif self.decoderType == DecoderType.WordBeamSearch: |
|  | # import compiled word beam search operation (see https://github.com/githubharald/CTCWordBeamSearch) |
|  | word\_beam\_search\_module = tf.load\_op\_library('TFWordBeamSearch.so') |
|  |  |
|  | # prepare information about language (dictionary, characters in dataset, characters forming words) |
|  | chars = str().join(self.charList) |
|  | wordChars = open('../model/wordCharList.txt').read().splitlines()[0] |
|  | corpus = open('../data/corpus.txt').read() |
|  |  |
|  | # decode using the "Words" mode of word beam search |
|  | self.decoder = word\_beam\_search\_module.word\_beam\_search(tf.nn.softmax(self.ctcIn3dTBC, axis=2), 50, 'Words', |
|  | 0.0, corpus.encode('utf8'), chars.encode('utf8'), |
|  | wordChars.encode('utf8')) |
|  |  |
|  | def setupTF(self): |
|  | "initialize TF" |
|  | print('Python: ' + sys.version) |
|  | print('Tensorflow: ' + tf.\_\_version\_\_) |
|  |  |
|  | sess = tf.compat.v1.Session() # TF session |
|  |  |
|  | saver = tf.compat.v1.train.Saver(max\_to\_keep=1) # saver saves model to file |
|  | modelDir = '../model/' |
|  | latestSnapshot = tf.train.latest\_checkpoint(modelDir) # is there a saved model? |
|  |  |
|  | # if model must be restored (for inference), there must be a snapshot |
|  | if self.mustRestore and not latestSnapshot: |
|  | raise Exception('No saved model found in: ' + modelDir) |
|  |  |
|  | # load saved model if available |
|  | if latestSnapshot: |
|  | print('Init with stored values from ' + latestSnapshot) |
|  | saver.restore(sess, latestSnapshot) |
|  | else: |
|  | print('Init with new values') |
|  | sess.run(tf.compat.v1.global\_variables\_initializer()) |
|  |  |
|  | return (sess, saver) |
|  |  |
|  | def toSparse(self, texts): |
|  | "put ground truth texts into sparse tensor for ctc\_loss" |
|  | indices = [] |
|  | values = [] |
|  | shape = [len(texts), 0] # last entry must be max(labelList[i]) |
|  |  |
|  | # go over all texts |
|  | for (batchElement, text) in enumerate(texts): |
|  | # convert to string of label (i.e. class-ids) |
|  | labelStr = [self.charList.index(c) for c in text] |
|  | # sparse tensor must have size of max. label-string |
|  | if len(labelStr) > shape[1]: |
|  | shape[1] = len(labelStr) |
|  | # put each label into sparse tensor |
|  | for (i, label) in enumerate(labelStr): |
|  | indices.append([batchElement, i]) |
|  | values.append(label) |
|  |  |
|  | return (indices, values, shape) |
|  |  |
|  | def decoderOutputToText(self, ctcOutput, batchSize): |
|  | "extract texts from output of CTC decoder" |
|  |  |
|  | # contains string of labels for each batch element |
|  | encodedLabelStrs = [[] for i in range(batchSize)] |
|  |  |
|  | # word beam search: label strings terminated by blank |
|  | if self.decoderType == DecoderType.WordBeamSearch: |
|  | blank = len(self.charList) |
|  | for b in range(batchSize): |
|  | for label in ctcOutput[b]: |
|  | if label == blank: |
|  | break |
|  | encodedLabelStrs[b].append(label) |
|  |  |
|  | # TF decoders: label strings are contained in sparse tensor |
|  | else: |
|  | # ctc returns tuple, first element is SparseTensor |
|  | decoded = ctcOutput[0][0] |
|  |  |
|  | # go over all indices and save mapping: batch -> values |
|  | idxDict = {b: [] for b in range(batchSize)} |
|  | for (idx, idx2d) in enumerate(decoded.indices): |
|  | label = decoded.values[idx] |
|  | batchElement = idx2d[0] # index according to [b,t] |
|  | encodedLabelStrs[batchElement].append(label) |
|  |  |
|  | # map labels to chars for all batch elements |
|  | return [str().join([self.charList[c] for c in labelStr]) for labelStr in encodedLabelStrs] |
|  |  |
|  | def trainBatch(self, batch): |
|  | "feed a batch into the NN to train it" |
|  | numBatchElements = len(batch.imgs) |
|  | sparse = self.toSparse(batch.gtTexts) |
|  | rate = 0.01 if self.batchesTrained < 10 else ( |
|  | 0.001 if self.batchesTrained < 10000 else 0.0001) # decay learning rate |
|  | evalList = [self.optimizer, self.loss] |
|  | feedDict = {self.inputImgs: batch.imgs, self.gtTexts: sparse, |
|  | self.seqLen: [Model.maxTextLen] \* numBatchElements, self.learningRate: rate, self.is\_train: True} |
|  | (\_, lossVal) = self.sess.run(evalList, feedDict) |
|  | self.batchesTrained += 1 |
|  | return lossVal |
|  |  |
|  | def dumpNNOutput(self, rnnOutput): |
|  | "dump the output of the NN to CSV file(s)" |
|  | dumpDir = '../dump/' |
|  | if not os.path.isdir(dumpDir): |
|  | os.mkdir(dumpDir) |
|  |  |
|  | # iterate over all batch elements and create a CSV file for each one |
|  | maxT, maxB, maxC = rnnOutput.shape |
|  | for b in range(maxB): |
|  | csv = '' |
|  | for t in range(maxT): |
|  | for c in range(maxC): |
|  | csv += str(rnnOutput[t, b, c]) + ';' |
|  | csv += '\n' |
|  | fn = dumpDir + 'rnnOutput\_' + str(b) + '.csv' |
|  | print('Write dump of NN to file: ' + fn) |
|  | with open(fn, 'w') as f: |
|  | f.write(csv) |
|  |  |
|  | def inferBatch(self, batch, calcProbability=False, probabilityOfGT=False): |
|  | "feed a batch into the NN to recognize the texts" |
|  |  |
|  | # decode, optionally save RNN output |
|  | numBatchElements = len(batch.imgs) |
|  | evalRnnOutput = self.dump or calcProbability |
|  | evalList = [self.decoder] + ([self.ctcIn3dTBC] if evalRnnOutput else []) |
|  | feedDict = {self.inputImgs: batch.imgs, self.seqLen: [Model.maxTextLen] \* numBatchElements, |
|  | self.is\_train: False} |
|  | evalRes = self.sess.run(evalList, feedDict) |
|  | decoded = evalRes[0] |
|  | texts = self.decoderOutputToText(decoded, numBatchElements) |
|  |  |
|  | # feed RNN output and recognized text into CTC loss to compute labeling probability |
|  | probs = None |
|  | if calcProbability: |
|  | sparse = self.toSparse(batch.gtTexts) if probabilityOfGT else self.toSparse(texts) |
|  | ctcInput = evalRes[1] |
|  | evalList = self.lossPerElement |
|  | feedDict = {self.savedCtcInput: ctcInput, self.gtTexts: sparse, |
|  | self.seqLen: [Model.maxTextLen] \* numBatchElements, self.is\_train: False} |
|  | lossVals = self.sess.run(evalList, feedDict) |
|  | probs = np.exp(-lossVals) |
|  |  |
|  | # dump the output of the NN to CSV file(s) |
|  | if self.dump: |
|  | self.dumpNNOutput(evalRes[1]) |
|  |  |
|  | return (texts, probs) |
|  |  |
|  | def save(self): |
|  | "save model to file" |
|  | self.snapID += 1 |
|  | self.saver.save(self.sess, '../model/snapshot', global\_step=self.snapID) |

Код на Python main.py

|  |
| --- |
| from \_\_future\_\_ import division |
|  | from \_\_future\_\_ import print\_function |
|  |  |
|  | import argparse |
|  |  |
|  | import cv2 |
|  | import editdistance |
|  |  |
|  | from DataLoader import DataLoader, Batch |
|  | from Model import Model, DecoderType |
|  | from SamplePreprocessor import preprocess |
|  |  |
|  |  |
|  | class FilePaths: |
|  | "filenames and paths to data" |
|  | fnCharList = '../model/charList.txt' |
|  | fnAccuracy = '../model/accuracy.txt' |
|  | fnTrain = '../data/' |
|  | fnInfer = '../data/test.png' |
|  | fnCorpus = '../data/corpus.txt' |
|  |  |
|  |  |
|  | def train(model, loader): |
|  | "train NN" |
|  | epoch = 0 # number of training epochs since start |
|  | bestCharErrorRate = float('inf') # best valdiation character error rate |
|  | noImprovementSince = 0 # number of epochs no improvement of character error rate occured |
|  | earlyStopping = 5 # stop training after this number of epochs without improvement |
|  | while True: |
|  | epoch += 1 |
|  | print('Epoch:', epoch) |
|  |  |
|  | # train |
|  | print('Train NN') |
|  | loader.trainSet() |
|  | while loader.hasNext(): |
|  | iterInfo = loader.getIteratorInfo() |
|  | batch = loader.getNext() |
|  | loss = model.trainBatch(batch) |
|  | print('Batch:', iterInfo[0], '/', iterInfo[1], 'Loss:', loss) |
|  |  |
|  | # validate |
|  | charErrorRate = validate(model, loader) |
|  |  |
|  | # if best validation accuracy so far, save model parameters |
|  | if charErrorRate < bestCharErrorRate: |
|  | print('Character error rate improved, save model') |
|  | bestCharErrorRate = charErrorRate |
|  | noImprovementSince = 0 |
|  | model.save() |
|  | open(FilePaths.fnAccuracy, 'w').write( |
|  | 'Validation character error rate of saved model: %f%%' % (charErrorRate \* 100.0)) |
|  | else: |
|  | print('Character error rate not improved') |
|  | noImprovementSince += 1 |
|  |  |
|  | # stop training if no more improvement in the last x epochs |
|  | if noImprovementSince >= earlyStopping: |
|  | print('No more improvement since %d epochs. Training stopped.' % earlyStopping) |
|  | break |
|  |  |
|  |  |
|  | def validate(model, loader): |
|  | "validate NN" |
|  | print('Validate NN') |
|  | loader.validationSet() |
|  | numCharErr = 0 |
|  | numCharTotal = 0 |
|  | numWordOK = 0 |
|  | numWordTotal = 0 |
|  | while loader.hasNext(): |
|  | iterInfo = loader.getIteratorInfo() |
|  | print('Batch:', iterInfo[0], '/', iterInfo[1]) |
|  | batch = loader.getNext() |
|  | (recognized, \_) = model.inferBatch(batch) |
|  |  |
|  | print('Ground truth -> Recognized') |
|  | for i in range(len(recognized)): |
|  | numWordOK += 1 if batch.gtTexts[i] == recognized[i] else 0 |
|  | numWordTotal += 1 |
|  | dist = editdistance.eval(recognized[i], batch.gtTexts[i]) |
|  | numCharErr += dist |
|  | numCharTotal += len(batch.gtTexts[i]) |
|  | print('[OK]' if dist == 0 else '[ERR:%d]' % dist, '"' + batch.gtTexts[i] + '"', '->', |
|  | '"' + recognized[i] + '"') |
|  |  |
|  | # print validation result |
|  | charErrorRate = numCharErr / numCharTotal |
|  | wordAccuracy = numWordOK / numWordTotal |
|  | print('Character error rate: %f%%. Word accuracy: %f%%.' % (charErrorRate \* 100.0, wordAccuracy \* 100.0)) |
|  | return charErrorRate |
|  |  |
|  |  |
|  | def infer(model, fnImg): |
|  | "recognize text in image provided by file path" |
|  | img = preprocess(cv2.imread(fnImg, cv2.IMREAD\_GRAYSCALE), Model.imgSize) |
|  | batch = Batch(None, [img]) |
|  | (recognized, probability) = model.inferBatch(batch, True) |
|  | print('Recognized:', '"' + recognized[0] + '"') |
|  | print('Probability:', probability[0]) |
|  |  |
|  |  |
|  | def main(): |
|  | "main function" |
|  | # optional command line args |
|  | parser = argparse.ArgumentParser() |
|  | parser.add\_argument('--train', help='train the NN', action='store\_true') |
|  | parser.add\_argument('--validate', help='validate the NN', action='store\_true') |
|  | parser.add\_argument('--beamsearch', help='use beam search instead of best path decoding', action='store\_true') |
|  | parser.add\_argument('--wordbeamsearch', help='use word beam search instead of best path decoding', |
|  | action='store\_true') |
|  | parser.add\_argument('--dump', help='dump output of NN to CSV file(s)', action='store\_true') |
|  |  |
|  | args = parser.parse\_args() |
|  |  |
|  | decoderType = DecoderType.BestPath |
|  | if args.beamsearch: |
|  | decoderType = DecoderType.BeamSearch |
|  | elif args.wordbeamsearch: |
|  | decoderType = DecoderType.WordBeamSearch |
|  |  |
|  | # train or validate on IAM dataset |
|  | if args.train or args.validate: |
|  | # load training data, create TF model |
|  | loader = DataLoader(FilePaths.fnTrain, Model.batchSize, Model.imgSize, Model.maxTextLen) |
|  |  |
|  | # save characters of model for inference mode |
|  | open(FilePaths.fnCharList, 'w').write(str().join(loader.charList)) |
|  |  |
|  | # save words contained in dataset into file |
|  | open(FilePaths.fnCorpus, 'w').write(str(' ').join(loader.trainWords + loader.validationWords)) |
|  |  |
|  | # execute training or validation |
|  | if args.train: |
|  | model = Model(loader.charList, decoderType) |
|  | train(model, loader) |
|  | elif args.validate: |
|  | model = Model(loader.charList, decoderType, mustRestore=True) |
|  | validate(model, loader) |
|  |  |
|  | # infer text on test image |
|  | else: |
|  | print(open(FilePaths.fnAccuracy).read()) |
|  | model = Model(open(FilePaths.fnCharList).read(), decoderType, mustRestore=True, dump=args.dump) |
|  | infer(model, FilePaths.fnInfer) |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |

Код на Python analyze.py

|  |
| --- |
| from \_\_future\_\_ import division |
|  | from \_\_future\_\_ import print\_function |
|  |  |
|  | import copy |
|  | import math |
|  | import pickle |
|  | import sys |
|  |  |
|  | import cv2 |
|  | import matplotlib.pyplot as plt |
|  | import numpy as np |
|  |  |
|  | from DataLoader import Batch |
|  | from Model import Model, DecoderType |
|  | from SamplePreprocessor import preprocess |
|  |  |
|  |  |
|  | # constants like filepaths |
|  | class Constants: |
|  | "filenames and paths to data" |
|  | fnCharList = '../model/charList.txt' |
|  | fnAnalyze = '../data/analyze.png' |
|  | fnPixelRelevance = '../data/pixelRelevance.npy' |
|  | fnTranslationInvariance = '../data/translationInvariance.npy' |
|  | fnTranslationInvarianceTexts = '../data/translationInvarianceTexts.pickle' |
|  | gtText = 'are' |
|  | distribution = 'histogram' # 'histogram' or 'uniform' |
|  |  |
|  |  |
|  | def odds(val): |
|  | return val / (1 - val) |
|  |  |
|  |  |
|  | def weightOfEvidence(origProb, margProb): |
|  | return math.log2(odds(origProb)) - math.log2(odds(margProb)) |
|  |  |
|  |  |
|  | def analyzePixelRelevance(): |
|  | "simplified implementation of paper: Zintgraf et al - Visualizing Deep Neural Network Decisions: Prediction Difference Analysis" |
|  |  |
|  | # setup model |
|  | model = Model(open(Constants.fnCharList).read(), DecoderType.BestPath, mustRestore=True) |
|  |  |
|  | # read image and specify ground-truth text |
|  | img = cv2.imread(Constants.fnAnalyze, cv2.IMREAD\_GRAYSCALE) |
|  | (w, h) = img.shape |
|  | assert Model.imgSize[1] == w |
|  |  |
|  | # compute probability of gt text in original image |
|  | batch = Batch([Constants.gtText], [preprocess(img, Model.imgSize)]) |
|  | (\_, probs) = model.inferBatch(batch, calcProbability=True, probabilityOfGT=True) |
|  | origProb = probs[0] |
|  |  |
|  | grayValues = [0, 63, 127, 191, 255] |
|  | if Constants.distribution == 'histogram': |
|  | bins = [0, 31, 95, 159, 223, 255] |
|  | (hist, \_) = np.histogram(img, bins=bins) |
|  | pixelProb = hist / sum(hist) |
|  | elif Constants.distribution == 'uniform': |
|  | pixelProb = [1.0 / len(grayValues) for \_ in grayValues] |
|  | else: |
|  | raise Exception('unknown value for Constants.distribution') |
|  |  |
|  | # iterate over all pixels in image |
|  | pixelRelevance = np.zeros(img.shape, np.float32) |
|  | for x in range(w): |
|  | for y in range(h): |
|  |  |
|  | # try a subset of possible grayvalues of pixel (x,y) |
|  | imgsMarginalized = [] |
|  | for g in grayValues: |
|  | imgChanged = copy.deepcopy(img) |
|  | imgChanged[x, y] = g |
|  | imgsMarginalized.append(preprocess(imgChanged, Model.imgSize)) |
|  |  |
|  | # put them all into one batch |
|  | batch = Batch([Constants.gtText] \* len(imgsMarginalized), imgsMarginalized) |
|  |  |
|  | # compute probabilities |
|  | (\_, probs) = model.inferBatch(batch, calcProbability=True, probabilityOfGT=True) |
|  |  |
|  | # marginalize over pixel value (assume uniform distribution) |
|  | margProb = sum([probs[i] \* pixelProb[i] for i in range(len(grayValues))]) |
|  |  |
|  | pixelRelevance[x, y] = weightOfEvidence(origProb, margProb) |
|  |  |
|  | print(x, y, pixelRelevance[x, y], origProb, margProb) |
|  |  |
|  | np.save(Constants.fnPixelRelevance, pixelRelevance) |
|  |  |
|  |  |
|  | def analyzeTranslationInvariance(): |
|  | # setup model |
|  | model = Model(open(Constants.fnCharList).read(), DecoderType.BestPath, mustRestore=True) |
|  |  |
|  | # read image and specify ground-truth text |
|  | img = cv2.imread(Constants.fnAnalyze, cv2.IMREAD\_GRAYSCALE) |
|  | (w, h) = img.shape |
|  | assert Model.imgSize[1] == w |
|  |  |
|  | imgList = [] |
|  | for dy in range(Model.imgSize[0] - h + 1): |
|  | targetImg = np.ones((Model.imgSize[1], Model.imgSize[0])) \* 255 |
|  | targetImg[:, dy:h + dy] = img |
|  | imgList.append(preprocess(targetImg, Model.imgSize)) |
|  |  |
|  | # put images and gt texts into batch |
|  | batch = Batch([Constants.gtText] \* len(imgList), imgList) |
|  |  |
|  | # compute probabilities |
|  | (texts, probs) = model.inferBatch(batch, calcProbability=True, probabilityOfGT=True) |
|  |  |
|  | # save results to file |
|  | f = open(Constants.fnTranslationInvarianceTexts, 'wb') |
|  | pickle.dump(texts, f) |
|  | f.close() |
|  | np.save(Constants.fnTranslationInvariance, probs) |
|  |  |
|  |  |
|  | def showResults(): |
|  | # 1. pixel relevance |
|  | pixelRelevance = np.load(Constants.fnPixelRelevance) |
|  | plt.figure('Pixel relevance') |
|  |  |
|  | plt.imshow(pixelRelevance, cmap=plt.cm.jet, vmin=-0.25, vmax=0.25) |
|  | plt.colorbar() |
|  |  |
|  | img = cv2.imread(Constants.fnAnalyze, cv2.IMREAD\_GRAYSCALE) |
|  | plt.imshow(img, cmap=plt.cm.gray, alpha=.4) |
|  |  |
|  | # 2. translation invariance |
|  | probs = np.load(Constants.fnTranslationInvariance) |
|  | f = open(Constants.fnTranslationInvarianceTexts, 'rb') |
|  | texts = pickle.load(f) |
|  | texts = ['%d:' % i + texts[i] for i in range(len(texts))] |
|  | f.close() |
|  |  |
|  | plt.figure('Translation invariance') |
|  |  |
|  | plt.plot(probs, 'o-') |
|  | plt.xticks(np.arange(len(texts)), texts, rotation='vertical') |
|  | plt.xlabel('horizontal translation and best path') |
|  | plt.ylabel('text probability of "%s"' % Constants.gtText) |
|  |  |
|  | # show both plots |
|  | plt.show() |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | if len(sys.argv) > 1: |
|  | if sys.argv[1] == '--relevance': |
|  | print('Analyze pixel relevance') |
|  | analyzePixelRelevance() |
|  | elif sys.argv[1] == '--invariance': |
|  | print('Analyze translation invariance') |
|  | analyzeTranslationInvariance() |
|  | else: |
|  | print('Show results') |
|  | showResults() |

# СПИСОК ЛИТЕРАТУРЫ

1. https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5/ Build a Handwritten Text Recognition System using TensorFlow
2. https://vc.ru/ml/96273-kak-rabotaet-raspoznavanie-rukopisnogo-teksta/ Как работает распознавание рукописного текста
3. https://scienceforum.ru/2019/article/2018017330/ МЕТОДЫ И ЭТАПЫ РАСПОЗНАВАНИЯ РУКОПИСНОГО ТЕКСТА
4. https://fki.tic.heia-fr.ch/databases/iam-handwriting-database/ IAM Handwriting Database
5. https://habr.com/ru/post/312450/ Нейронные сети для начинающих. Часть 1