РЕФЕРАТ

Магистерская диссертация содержит 59 страниц, 11 рисунков, 3 таблицы, 13 использованных источников.

РАСПОЗНАВАНИЕ ЛИЦ, АТАКУЮЩИЕ ПРИМЕРЫ, ПРИВАТНОСТЬ, МАШИННОЕ ОБУЧЕНИЕ, АУГМЕНТАЦИИ ИЗОБРАЖЕНИЙ, АНАЛИЗ ДАННЫХ, ПРЕДСТАВЛЕНИЕ ИЗОБРАЖЕНИЙ, ГРАДИЕНТНЫЙ СПУСК С ИМПУЛЬСОМ.

Магистерская диссертация посвящена изучению методов генерации атакующих примеров, в частности для систем распознавания лиц на основе глубоких нейронных сетей. Анализируется специфика работы систем распознавания лиц, проводится обзор методов и алгоритмов по созданию атакующих примеров. На основе анализа собирается знание из нескольких областей и применяется к решению поставленной задачи. Предлагается улучшение переносимости атакующих примеров при помощи вероятностной аугментации изображений.

Реализован и исследован алгоритм, способный успешно генерировать атакующие примеры. Изучены его качество и переносимость. Даются рекомендации по подбору гиперпараметров.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

DNN – глубокая нейронная сеть.

FGSM – метод знака быстрого градиента.

MTCNN – multi-task cascaded convolutional networks.

BIM – базовый итеративный метод.

ВВЕДЕНИЕ

Мы живём во время, когда информационные технологии окружают человека практически во всех сферах его жизни. Каждый день через нас проходит огромное количество информации. Ещё десять лет назад было сложно представить, что у человека за день в поле зрения будет появляться такое большое количество данных. Всё это стало настолько привычно, что большинство даже не задумывается о том, сколько информации за сутки поглощается ими, обрабатывается и передаётся дальше. Появление новых коммуникационных технологий оказало огромное влияние на рост этой тенденции.

К сожалению, технологический прогресс сочетается с новыми сложными методами для информационных атак и мошенничества. Все чаще вокруг всплывают термины «машинное обучение» или «глубокие нейронные сети». Несомненно, такие прорывные технологии вывели решение задач основанных на данных на качественно новый уровень. Но как и в любой быстро растущей технологии при таком бешеном темпе тяжело думать о безопасности используемого инструмента. Как будет раскрыто далее в этой работе, оказывается, надежность нейронных сетей находится под большим вопросом. А вместе с этим находится под угрозой приватность людей, использующих эти технологии. Вопросы приватности и безопасности являются основополагающими для данной работы и подчеркивают её актуальность.

Давайте сначала вкратце рассмотрим термин «приватность». Термин «приватность» широко используется в технологическом секторе. Текущие подходы к «приватности» и «машинному обучению» можно просто описать в основном в двух разделах: «контроль пользователя» и «защита данных». Таким образом, в разделе «контроль пользователя» права пользователя могут быть прописаны путем понимания того, кто собирает данные, с какой целью и как долго они будут храниться. С другой стороны, переходя к разделу «защита данных», применяется шифрование данных, и удаление идентифицирующей

информации, чтобы сделать данные анонимными. Второй подход применяется повсеместно.

Но в настоящее время, что касается машинного обучения и применения данных в нем, в обоих этих компонентах есть пробелы, которые нам необходимо устранить. Кроме того, само по себе машинное обучение подразумевает взаимодействие с данными, а это значит, что с этими данными нужно работать – где-то они должны быть расшифрованы, создавая новую уязвимость. Поэтому важно иметь дополнительную защиту для обоих вышеупомянутых направлений.

Несколько лет назад вступил в силу генеральный регламент о защите персональных данных. Это постановление Европейского Союза с помощью которого Европейский парламент, Совет Европейского Союза и Европейская комиссия усиливают и унифицируют защиту персональных данных всех лиц в Европейском Союзе. Постановление также направлено на экспорт данных из Европейском Союзе.

В машинном обучении применяются различные техники для решения этой задачи:

- Federated Learning;
- Multi Party Computation;
- Homomorphic Encryption;
- Differential Privacy.

В данной же работе для достижения приватности применяется близкое но немного другое направление машинного обучения – состязательные атаки (adversarial attack) или генерация «атакующих» примеров. В нем не пытаются решать проблему приватности напрямую, а вместо этого исследуют уязвимости алгоритмов машинного обучения, в частности – глубоких нейронных сетей. Задача приватности в данной работе, которую будет решать генерация атакующих примеров рассматривается в домене систем по распознаванию лиц.

В настоящее время системы распознавания лиц используются повсеместно. С помощью них решается огромное количество задач и существует множество применений данной технологии. К ним относятся, например:

- система «умный город»;
- видео-аналитика;
- 3D сканер лица;
- идентификация пользователей;
- оплата услуг;

Но вместе с широким использованием технологии приходят и проблемы безопасности. Например, существует множество веб-сервисов, которые по одной лишь фотографии могут выдать целый список информации о пользователе в интернете. И эта проблема поднимает остро вопрос приватности, очень важный в наши дни.

Данная работа посвящена исследованию и созданию алгоритма, способного «анонимизировать» изображение человека для систем распознавания лиц. Как было показано ранее [7], существует возможность «скрыться» от таких систем в физическом мире при помощи наклеек на части лица или детали одежды, такие как очки. Но очевидно, что данное решение весьма не практично и может быть мало применимо. Поэтому в данной работе рассматривается создание такого фильтра для изображений, который позволит скрыть их содержимое для систем распознавания лиц в сети интернет.

Так как в современных системах распознавания лиц зарекомендовали себя глубокие нейронные сети [8], то и решение должно основываться на работе этих алгоритмов. В 2014 году было показано [9], что глубокие нейронные сети подвержены состязательным атакам. Для входного изображения x и глубокой нейронной сети с функцией потерь $J(\theta, x, y)$ создавалась специальная маска, имеющая вид $\varepsilon\, sign(\nabla_x J(\theta, x, y))$, где ε – уровень шума, x – входное изображение, y – метка изображения . И если для входного изображения x модель могла однозначно определить класс, то для

изображения с маской $x + \varepsilon\, sign(\nabla_x J(\theta, x, y))$ модель уже ошибалась, выдавая не правильный вердикт. При этом визуально изменения, наложенные на изображение, не были заметны человеческому глазу. Данный подход можно применить и для систем распознавания лиц на основе нейронных сетей.

В работе рассмотрены особенности моделей глубокого обучения для распознаванию лиц, рассмотрены передовые модели из данной области. Было замечено, на основе публикации [10], что строить атакующие примеры стоит не от метки класса изображения, а от его внутреннего представления в сети, так называемого – вложения. Также, так как неизвестно какая именно глубокая нейронная сеть применяется в той или иной системе распознавания лиц, а примеры строятся на основе весов конкретной модели, было проведено исследование по возможности переносимости [10] таких атакующих примеров на другие архитектуры нейронных сетей. Для улучшения переносимости были использованы аугментации изображений при построении атакующего примера. Подобная идея уже применялась ранее и показывала хорошие возможности для роста переносимости [11]. Аугментации в данной работе носят вероятностный характер, так как на каждой итерации создания атакующего примера происходят случайные изменения изображения, с параметрами из равномерного распределения U(135, 160). Границы распределения выбраны исходя из специфики работы с изображением. Рассмотрение таким вероятностных аугментаций открывает новое направление для исследований в данной области.

Таким образом, главными целями данной работы являются:

- исследование генерации атакующих примеров для глубоких нейронных сетей в задаче распознавания лиц;
- исследование переносимости атакующих примеров между различными глубокими нейронными сетями в задаче распознавания лиц;
- разработка алгоритма генерации атакующих примеров для глубоких нейронных сетей в задаче распознавания лиц;
- программная реализация предложенного алгоритма;

- анализ эффективности предложенного алгоритма.

Полученное решение позволило добиться алгоритма, способного «анонимизировать» изображение для систем распознавания лиц. В качестве заключительного эксперимента производились тесты с изображениями на веб-сервисах в интернете, которые по фотографии находят данные о человеке в сети. И не имея знания об архитектуре сетей, используемых в данных сервисах, удалось с применением фильтра сбить детектирующую способность моделей.

ОСНОВНАЯ ЧАСТЬ

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Применение машинного обучения

В этом подразделе рассматривается подход машинного обучения, а именно задача классификации. Так как в поставленной проблеме представлено количество классов равное количеству уникальных людей, известных системе, то решаемая задача является задачей мульти-классификации. Описан принцип работы глубоких нейронных сетей. Приведено описание и сравнение метрик качества.

1.1.1 Постановка задачи классификации

В машинном обучении существует несколько типов задач, как правило их делят на несколько групп: обучение с учителем, обучение без учителя, обучение с частичным привлечением учителя. Задача классификации относится к первому – обучение с учителем. Это означает, что предоставлена выборка с заранее известными ответами для неё. В случае задачи регрессии – это вещественные числа, в случае задача классификации – дискретное количество значений. Самый простой случай – это бинарная классификация. По мимо неё часто встречается задача мульти-классификации, которая будет рассмотрена позже, и задача нечеткой классификации, когда одному примеру может соответствовать несколько классов.

Пусть задано пространство объектов $\boldsymbol{U}$ и пространство классов $\boldsymbol{C}$, и существует некоторое неизвестное отображение $\boldsymbol{f}: \boldsymbol{U} \to \boldsymbol{C}$. Дана обучающая выборка $\{(\boldsymbol{u_1}, \boldsymbol{c_1}), \dots, (\boldsymbol{u_n}, \boldsymbol{c_m})\}$, где $\boldsymbol{n}$ – число объектов выборки, $\boldsymbol{m}$ – число классов, $\boldsymbol{u_i} \in \boldsymbol{U}$ – объект выборки, $\boldsymbol{c_i} \in \boldsymbol{C}$ – класс. Для этой обучающей выборки известны значения отображения $\boldsymbol{f}$. Тогда задача классификации заключается в поиске такого отображения $\boldsymbol{f^*}: \boldsymbol{U} \to \boldsymbol{C}$, способного классифицировать $\forall \boldsymbol{u} \in \boldsymbol{U}$.

В качестве объектов классификации может выступать множество различных данных. Так на вход классификатору могут подаваться, например, текст, фотография, видео, аудио или материальный объект, описанный с

помощью признаков. Самый частый подход это представление объектов в виде матрицы «объекты-признаки» $\boldsymbol{X}$. В таком матрице строки – это объекты, а столбцы – это определенные $\boldsymbol{k}$ признаков.

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{u_1^1} & \cdots & \boldsymbol{u_1^k} \\ \vdots & \ddots & \vdots \\ \boldsymbol{u_n^1} & \cdots & \boldsymbol{u_n^k} \end{pmatrix}$$

Так, например, решая задачу классификации животных за строки можно принять животных, а столбцы будут соответствовать размеру, цвету, количеству зубов и прочим признакам. В таком случае класс каждого объекта будет в соответствующей строке вектора $\boldsymbol{Y}$. В случае бинарной классификации в векторе $\boldsymbol{Y}$ будут содержаться значения $\{\boldsymbol{-1};\ \boldsymbol{+1}\}$, в случае задачи мульти-классификации $\boldsymbol{Y} = \{\boldsymbol{1}, \ldots, \boldsymbol{n}\}$, где $\boldsymbol{n}$ – количество классов.

Признаки могут быть следующих типов:

- бинарные;
- номинальные;
- порядковые;
- количественные.

Формальное определение признака: признак – это отображение $\boldsymbol{f}: \boldsymbol{U} \rightarrow \boldsymbol{D_f}$, где $\boldsymbol{D_f}$ – это множество допустимых значений признака. Соответственно каждый объект может быть представлен вектором таких отображений $\boldsymbol{f}$, иными словами – вектором признаков.

В практических задачах часто встречаются все вышеперечисленные типы признаков. Из-за этого некоторые методы решения задачи классификации могут не подойти.

Процесс решения задачи классификации состоит нескольких важных этапов. Каждый из них играет значительную роль в качестве построенной модели. Для построения классификатора необходимы данные. И первый важный этап – это сбор данных для обучения модели. Считается, что самый верный способ улучшить качество модели – это добавить больше данных. Выборка должна быть репрезентативной и отображать действительное

распределение. Так, если наша модель обучена классифицировать на данных за июнь, а в июле что-то случилось и распределение изменилось, то классификатор не сможет перестроиться для ситуации, которую он никогда не встречал.

Выборка может быть линейно разделима или линейно неразделима. Ниже представлены соответствующие примеры на рис. 1.1 и 1.2.

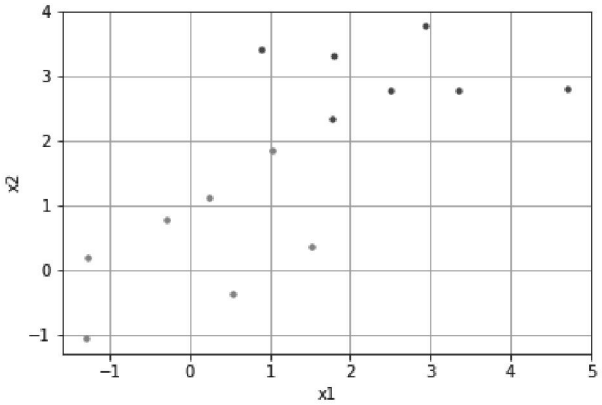

Рис. 1.1 Линейно разделимые классы

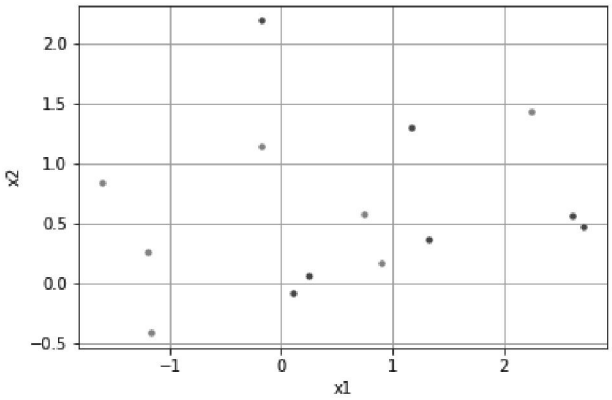

Рис. 1.2 Линейно неразделимые классы

На рис. 1.1 и 1.2 изображены примеры объектов двух классов. Синие точки – первый класс, красные точки – второй класс. Каждая точка является объектом. Оси абсцисс и ординат, в этом случае $\boldsymbol{x_1}$ и $\boldsymbol{x_2}$, соответствуют признакам. Координаты каждой точки – это определенные значения соответствующего признака. В данном случае они вещественные. Но если бы признаки были категориальными, то каждая точка лежала бы в узлах сетки. В первом случае между точками двух классов можно провести линию, так, что

два класса будут однозначно разделены. Заметим, что таких линий можно провести бесконечное множество. В таком, когда объекты можно разделить гиперплоскостью, классы называются линейно разделимыми. В случае, когда такой линии провести нельзя, классы называются линейно не разделимыми.

Таким образом, задачу классификации можно представить как построение такой гиперплоскости в признаковом пространстве размерности $\boldsymbol{n}$, что два класса будут разделены с наименьшим количеством ошибок.

Стоит отметить такие явления, возникающие при создании моделей машинного обучения, как переобучение и недообучение. Недообучение означает, что наша модель по каким-то причинам не смогла найти зависимостей в данных и обучиться. В таких ситуациях качество модели низкое на обучающей выборке. Это может происходить по разным причинам. Возможно неправильно подобран алгоритм машинного обучения, плохие данные, неверная конфигурация модели. В качестве борьбы с этим эффектом можно увеличить количество данных, поменять конфигурацию модели. Возможно стоит произвести отбор признаков или извлечение более удачных новых из имеющихся данных. Вторая ситуация – это переобучение. Оно гораздо чаще встречается на практике. Переобучение возникает, когда модель начинает слишком сильно подстраиваться под обучающую выборку, и начинает плохо работать на незнакомых примерах, которые не участвовали в её обучении. Такое явление может возникать, когда в данных обучающей выборке обнаруживаются скрытые зависимости. Способы борьбы с переобучением для разных видов моделей разные. Так для линейных моделей – это регуляризация, для деревьев – это обрезание веток, а для нейронных сетей, к примеру, ранняя остановка.

1.1.2 Методы решения задачи классификации

После того, как данные подготовлены, следует выбрать обучаемую модель. Существует большое множество различных алгоритмов, среди которых:

- линейные методы;
- байесовские методы;
- метрические методы;
- нейронные сети;
- логические методы.

В задачах связанных со звуком, изображениями, текстом хорошо зарекомендовали себя нейронные сети. Это очень мощный инструмент, и прежде чем использовать его стоит действительно подумать, нужно ли это в данной задаче. Для их использования требуется много данных и мощное оборудование. Время обучения может занимать от нескольких минут до нескольких дней. Сам подход заключается в композиции нелинейных функций от входного сигнала. Глубокие нейронные сети буду рассмотрены дальше в пункте 1.1.4.

Альтернативой нейронным сетям, служат алгоритмы основанные на деревьях решений. Деревья решений в чистом виде используются редко, так как малоэффективны. В практических задачах их объединяют в ансамбли и используют в виде случайного леса или градиентного бустинга. Оба этих подхода могут решать как задачу классификации, так и задачу регрессии. Случайный лес строит заданное количество деревьев решений, каждое из которых делает предсказание, после чего их результат усредняется. Градиентный бустинг строит деревья решений последовательно так, что каждое новое дерево учитывает ошибки предыдущего. Эти два сильных метода часто применяются в анализе данных.

К линейным методам классификации относятся такие как логистическая регрессия, метод опорных векторов и другие. Эти алгоритмы строят разделяющую гиперплоскость между двумя классами.

Метрические методы основаны на поиске расстояний между объектами обучающей выборки. Их популярный представитель – это алгоритм $\boldsymbol{k}$ ближайших соседей. Данный метод не требует обучения. Ему достаточно знать матрицу расстояний $\boldsymbol{D}$, для поиска $\boldsymbol{k}$ ближайших соседей по заданной

метрике $\boldsymbol{\rho}$, после чего соседи голосуют – примеров какого класса среди них больше, тот и выбирается моделью. Минусами такого подхода являются: необходимость хранить всю матрицу расстояний, при большом количестве признаков возникает проклятье размерности.

1.1.3 Метрики качества

После построения модели возникает необходимость узнать качество полученного классификатора. Выбор метрики, подходящей для конкретной задачи, является одной из обязанностей специалиста по анализу данных. Для описания метрик понадобится ввести понятие матрицы ошибок. Пусть имеется построенная модель $\boldsymbol{a(u)}$ и известна разметка классов для предоставленной выборки. Тогда матрица ошибок принимает вид, как показано в таблице 1.1.

Таблица 1.1 Матрица ошибок

	$\boldsymbol{c = +1}$	$\boldsymbol{c = -1}$
$\boldsymbol{a(u) = +1}$	TP	FP
$\boldsymbol{a(u) = -1}$	FN	TN

- TP – это количество объектов класса +1, предсказанных классификатором верно;
- TN – это количество объектов класса -1, предсказанных классификатором верно;
- FP – это ошибка первого рода, означающая ложное срабатывание классификатора;
- FN – это ошибка второго рода, означающая ложный пропуск объекта классификатором.

Ошибка второго рода страшнее ошибки первого рода по последствиям. Так, например, если модель скажет больному человеку, что он здоров, то результат может оказаться фатальным.

Самой понятной метрикой, является доля правильных ответов:

$$\frac{TP + FN}{TP + FN + FP + FN}.$$

Проблема этой метрики заключается в том, что при большом дисбалансе классов, она плохо отражает действительность. Так, классификатор выдавая в качестве предсказания всегда значение наибольшего класса будет иметь очень высокую долю правильных ответов.

Для борьбы с этим вводятся две метрики: точность и полнота.

$$\text{Точность} = \frac{TP}{TP + FP},$$

$$\text{Полнота} = \frac{TP}{TP + FN}.$$

Точность и полнота позволяют различать ложные срабатывания и ложные пропуски.

Точность отвечает за то: какая доля объектов классифицированных как +1, действительно являются положительными. Полнота отвечает за то: какую долю объектов класса +1 модель смогла определить верно.

Оптимизировать две метрики неудобно, поэтому существует метрика объединяющая в себе точность и полноту, как среднее гармоническое между ними – это F_1-мера.

$$F_1 = \frac{2 * \text{точность} * \text{полнота}}{\text{точность} + \text{полнота}}.$$

В случае задачи мульти-классификации переход от бинарного классификатора может быть осуществлен с помощью подхода «один против всех». Подход заключается в построении m классификаторов, где m – количество классов, способных выдавать оценку вероятности $a_i(u) \in \mathbb{R}$ принадлежности объекта к классу $i = \overline{1, \dots, m}$. Например, это может быть логистическая регрессия. После этого класс определяется по формуле:

$$c = \arg\max_{i=1,\dots,m} a_i(u).$$

Метрики для задачи бинарной классификации могут быть использованы в мульти-классификации, например, с помощью микро или макро усреднения. В обоих случаях строится матрица ошибок, но уже большего размера с учётом

всех классов. В случае микро усреднения, TP, FP, FN, TN находятся для каждой бинарной задачи отдельно, после усредняются и рассчитывается необходимая метрика. В ситуации макро усреднения, метрика для каждой бинарной задачи вычисляется сразу, после чего усредняется. В первом случае вклад каждого класса зависит от его размера. Во втором случае все классы вносят равный вклад.

1.1.4 Глубокие нейронные сети

Глубокие нейронные сети зарекомендовали себя в работе с неструктурированными данными, к ним относятся: звук, изображения, текст. Конечно, их также можно применять и к табличным данным, правда на практике со структурированными данными лучшие результаты показывают модели основанные на гридиентном бустинге.

Глубокие нейронные сети состоят из базовой абстракции, называемой – персептрон. Первая конструкция линейного персептрона была представлена Фрэнком Розенблаттом в 1950-х годах.

На самом деле, персептрон Розенблатта является линейной моделью, которая в оригинале решала поставленную задачу классификации. В пункте 1.1.1 мы уже обсуждали задачи бинарной и мульти классификации достаточно подробно. В качестве примера, предлагается взять самую простую задачу классификации – бинарную. Напомни, это когда все объекты в обучающей выборке помечены одной из двух меток (скажем +1 или -1) и задача состоит в том, чтобы научиться расставлять эти метки и новых, заранее не известных объектов из тестовой выборки.

Пусть $\boldsymbol{U} = \{(\boldsymbol{u_1}, \boldsymbol{c_1}), \dots, (\boldsymbol{u_k}, \boldsymbol{c_k})\}, \boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}, \boldsymbol{c_i} \in \{-\mathbf{1}, +\mathbf{1}\}$ – обучающая выборка. Тогда линейная модель классификации:

$$\boldsymbol{a(u, w) = sign < u, w >.}$$

Если знак скалярного произведения положительный то предсказываемый класс будет +1, если отрицательный, то -1.Требуется найти вектор весов $\boldsymbol{w}$ так,

чтобы число ошибок на обучающей выборке было минимально. Это сводится к задаче оптимизации.

Определение 1.1. [3]. Функция потерь $\mathcal{L}(a, c)$ характеризующая величину отклонения ответа алгоритма $a(u, w)$ от правильного ответа c, $\forall u \in U$.

Определение 1.2. [3]. Эмпирический риск $Q(w)$ — это функционал качества, характеризующий среднюю ошибку алгоритма с весами w на обучающей выборке.

Выпишем бинарную функцию потерь:

$$\mathcal{L}(a, c_i) = [a(u, w)c_i < 0].$$

Хотелось бы минимизировать число неверно классифицированных примеров, но к сожалению это не выйдет. Минимизировать такой функционал неудобно, так как он кусочно-постоянный. Поэтому вместо бинарной функции потерь в персептроне используется иная функция потерь, которая выглядит следующим образом:

$$Q(w) = -\sum_{x \in M} c(x)(w^T x).$$

где M обозначает множество тех примеров, которые персептрон с весами w классифицирует неверно.

Простым языком, решается задача минимизации суммарного отклонения ответов модели от правильных меток, но только в неверную сторону; правильный ответ ничего не вносит в функцию потерь. Знак производной всегда получается отрицательным за счет умножения на c(x): если правильный ответ −1, значит, персептрон выдал положительное число (иначе бы ответ был верным), и наоборот. В результате получается кусочно-линейная функция, дифференцируемая почти везде.

Теперь для процесса оптимизации применяется градиентный спуск, выразить градиентный шаг можно следующим образом:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w Q(w).$$

Последовательно проходим примеры из обучающего множества $x_1, \dots, x_n$, и для каждого примера x_i выполняем следующее:

- классифицирован правильно – не трогаем веса;
- классифицирован неправильно – меняем вес.

Для каждого объекта нашей выборки $\boldsymbol{x_i}$ ошибка будет уменьшаться, так как будет происходить подстройка веса под нее. Но не существует гарантий, что это не приведет к появлению ошибки на других примерах обучающего множества. Этот алгоритм был предложен Розенблаттом в его оригинальной работе, и получил название – правило обучения персептрона.

Но и это еще не все. Чтобы двигаться дальше, нам нужно добавить в персептрон еще один компонент — так называемую функцию активации. Дело в том, что в реальности персептроны не могут быть линейными, как мы их определили сейчас: если они останутся линейными, то из них невозможно будет составить содержательную сеть. На выходе персептрона обязательно присутствует нелинейная функция активации, которая принимает на вход все ту же линейную комбинацию. Функции активации бывают разными, например:

- логистический сигмоид;
- гиперболический тангенс;
- SoftSign;
- функция Хевисайда
- SoftPlus;
- ReLU;
- Leaky ReLU;
- ELU.

Но самая классическая, наиболее популярная исторически и до сих пор часто используемая функция активации — это логистический сигмоид (рис. 1.3):

$$\boldsymbol{\sigma(x) = \frac{1}{1 + exp^{-x}}}.$$

Это монотонно неубывающая функция, которая при $\boldsymbol{x \to -\infty}$ стремится к нулю, а при $\boldsymbol{x \to \infty}$ стремится к единице. Неформально говоря, это значит,

что если на вход подают большое отрицательное число, то нейрон совсем не активируется, а если большое положительное, то активируется почти наверняка.

Обучать один такой персептрон несложно: можно применить все тот же градиентный спуск. Разница только в том, что теперь мы рассматриваем задачу бинарной классификации, и данные c(x) представляют собой метки 0 и 1, а функция ошибки выглядит как перекрестная энтропия:

$$\boldsymbol{Q(w) = -\frac{1}{N}\sum_{i=1}^{N}(c_i \log \sigma(w^T x_i) + (1 - c_i) \log\left(1 - \sigma(w^T x_i)\right)).}$$

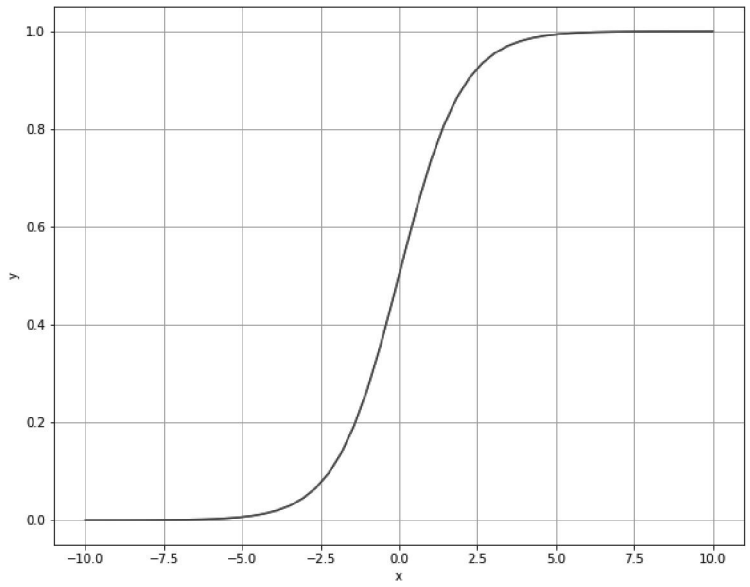

Рис. 1.3 Функции сигмоида

От этой функции по-прежнему несложно взять производную. В итоге персептрон с логистическим сигмоидом в качестве функции активации фактически реализует логистическую регрессию и строит при этом линейные разделяющие поверхности.

Теперь, когда мы разобрались с конструкцией одного персептрона, мы можем собрать из них целую сеть. Она будет представлять собой граф вычислений. Один персептрон может служить одним узлом в этом графе, выступая в роли элементарной функции: нам для этого требуется только уметь

считать частные производные по всем переменным, что для персептрона сделать совсем не сложно (рис. 1.4).

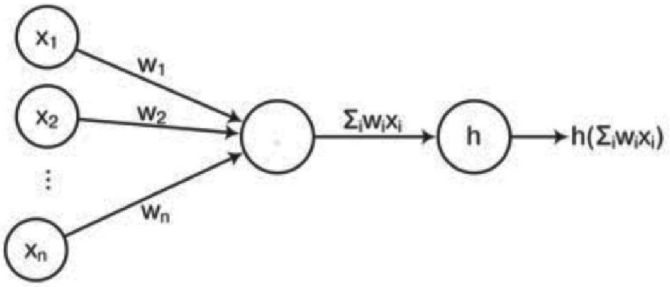

Рис. 1.4 Структура персептрона

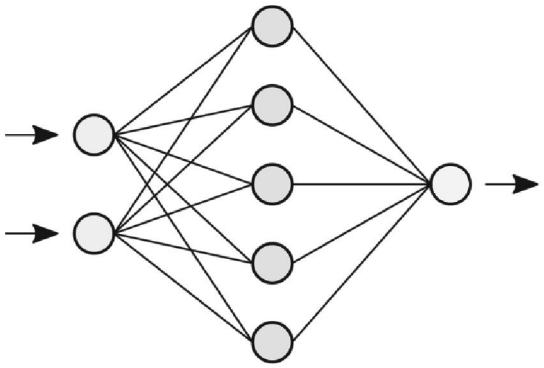

Рис. 1.5 Полносвязная нейронная сеть

Таким образом в глубоких нейронных сетях используются следующие компонент:

- нейронов (персептрон Розенблатта);
- функций активации;
- слоев.

Нейронные сети состоят из отдельных нейронов, которые объединяются между собой в слои. Входной сигнал подается входному слою, такой слой может состоять из нескольких нейронов. У каждого из них есть собственные веса. Далее выходной сигнал входного слоя попадает в следующий слой и рассматривается уже, как входной сигнал, для следующего уровня. И так далее. В какой-то момент мы получаем выходной сигнал на выходном слое.

Нейронная сеть состоит из следующих слоев: входного, выходного и скрытых.

Нейронная сеть называется глубокой, если она содержит один и более скрытых слоев (рис. 1.5).

При решении многих задач вы можете начать с единственного скрытого слоя и получить хорошие результаты. В исследованиях ранее было показано, что даже многослойный персептрон с одним скрытым слоем способен аппроксимировать функции различной сложности, если ввести достаточное количество нейронов на этот слой.

На практике данные часто структурированы в соответствии с какой-то иерархией. Глубокие нейронные сети самостоятельно могут получать преимущество из этого факта. Скрытые слои, расположенные в самом низу, моделируют низкоуровневые структуры, например линейные сегменты разнообразных форм и ориентаций. Промежуточные скрытые слои объединяют такие низкоуровневые структуры для моделирования структур промежуточных уровней, например квадратов или окружностей. Высокоуровневые скрытые слои объединяют промежуточные структуры для моделирования высокоуровневых структур, например лиц.

Иерархическая структура глубоких нейронных сетей не только помогает им быстрее сходиться в поиск наилучшего решения, но также улучшает способность сетей к обобщению на новые, ранее невидимые наборы данных.

Стоит отметить, что нейронные сети также часто используются в задачах машинного обучения без учителя. Примеры таких архитектур:

- сети Хопфилда;
- ограниченные машины Больцмана;
- авто-энкодеры.

В основном они решают задачи генерации, а не распознавания. Генерация может быть в различных доменах: изображения, тексты, видео, аудио.

Также стоит отметить различное многообразие методов обучения глубоких нейронных сетей. Как правило они основаны на градиенте, так как

существует мощный математический аппарат для его вычисления – алгоритм обратного распространения ошибки.

Приведем список возможных модификаций градиентного спуска для решения задачи обучения глубоких нейронных сетей:

- градиентный спуск;
- стохастический градиентный спуск;
- моментум оптимизация;
- ускоренный градиент Нестерова;
- AdaGrad;
- RMSProp;
- Adam.

Метод моментум или как его еще называют – импульс, будет рассмотрен подробнее далее для улучшения генерации атакующих примеров для систем по распознаванию лиц.

1.1.5 Работа с изображениями

Как уже говорилось ранее – глубокие нейронные сети хорошо подходят для работы с неструктурированными данными. К ним относятся: видео, аудио данные, текст и, конечно, изображения.

Изображения это вид данных, лежащий в основе данной научной работы. Они достаточно подробно изучены и огромное количество задач с ними уже решается в мире машинного обучения. Например:

- классификация изображения в различных отраслях и доменах;
- обнаружение и локализация объектов на изображении;
- семантическая сегментация;
- генерация новых изображений;
- и многие другие.

Так как в данной работе мы будем производить состязательные атаки на изображения лиц людей, то давайте подробнее остановимся на этом типе данных.

Цифровые изображения представляют из себя матрицы и состоят из пикселей. Пиксели – это дискретные значения, отражающие величину аналогового светового сигнала в соответствующих точках. Изображения бывают монохромными и цветными.

В монохромных изображениях каждый пиксель находится в числовом диапазоне от 0 до 255. Крайние значения отображают следующее – 0 соответствует черному цвету (минимальной интенсивности), 255 соответствует белому цвету (максимальной интенсивности). Каждое такое значение исчисляется размером в 1 байт памяти. Так как 256 различных значений можно представить с помощью 8 бит.

Один из основных подходов к представлению цветных изображений сводится к тому, чтобы представить 256 цветов при помощи того же объема памяти. Для этого сначала определяется «цветовая палитра». Иными словами вводится отображения между 256 значениями и цветами. Несмотря на довольно большие выразительные возможности, данный подход все же не обеспечивает достаточное разнообразие и широту диапазона цветов для адекватного отображения реалистичных изображений. Поэтому на фотографиях каждый пиксель обычно представляется тремя значениями – по одному для красной, зеленой и синей составляющих цвета. Пример представлен на рис. 1.6.

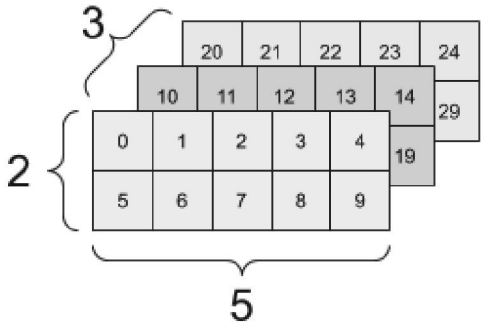

Рис. 1.6 Представление цветного изображения

Для каждого из этих RGB-значений обычно выделяется 1 байт памяти, что обеспечивает диапазон от 0 до 255. При этом цвет отдельного пикселя определяется как комбинация соответствующих RGB-значений. Такой способ представления отличается большой гибкостью – RGB-значения в диапазоне от 0 до 255 обеспечивают палитру из 16.777.216 различных цветов.

Точность изображения можно оценить двумя параметрами. Первый из них – это глубина цвета. Под этим понимается количество бит, используемых для значения каждого пикселя. Второй параметр – количество пикселей.

Общая точность изображения в каждом конкретном случае может отличаться. Это зависит от того: как оно снято (на это могут влиять настройки фотоаппарата), целей его использования и существующих ограничений по памяти. Для обработки изображений можно использовать большое количество распространенных графических форматов, например:

- jpeg (joint photographic experts group);
- png (portable network graphics);
- bmp (bitmap picture);
- gif (graphics interchange format);
- eps (encapsulated PostScript);
- RAW image files;
- tiff (tagged image file format).

Таким образом, изображения легко можно представлять в виде трехмерных тензоров, при этом сохраняя широкий диапазон значений цветов. Такое представление очень удобно, так как операции над матрицами: сложение, умножение, очень хорошо поддаются параллельным вычислениям при помощи графических процессоров. Также существуют качественные фреймворки для работы с тензорами на различных язык программирования. Поэтому такой способ представления цифровых изображений не только сохраняет высокую точность, но и позволяет добиться отличной вычислительной эффективности.

Также обратим внимание, что матричное представление изображений позволяет проводить над ними достаточно легко аффинные преобразования путем простого перемножения. Что также используется в данной научной работе. Ниже в таблице 1.2 представлены некоторые из наиболее частых преобразований с соответствующими им аффинными матрицами.

Стоит отдельно отметить возможность наложения матричных фильтров. Например – гауссово размытие или пространственный фильтр нижних частот. Но в данной работе они не понадобятся.

Таблица 1.2 Матрица аффинных преобразований

Название преобразования	Аффинная матрица
отражение	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
изменение пропорций	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
вращение	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

1.1.6 Свёрточные нейронные сети

Данный раздел будет посвящен свёрточным нейронным сетям и их специфики работы с изображениями.

Принцип работы нейронных сетей, рассмотренный в предыдущих главах опирается на следующий принцип: у нас есть вектор данных на входе некоторого слоя, затем этот вектор умножается на матрицу весов, затем применяется нелинейная функция активации. И это повторяется в каждом слое полносвязной сети. Но у нас есть некоторое априорное знание о типе данных изображений. Например, мы понимаем, что они имеют некоторую структуру, контуры, текстуры. И было бы большим опущением не воспользоваться этой информацией. Таким образом свёрточные нейронные сети призваны учитывать топологию данных.

Определение 1.3. [3]. Карта признаков – это матрица с выделенными в ней признаками, полученная из слоя свёртки.

Свёрточные нейронные сети имеют в своей основе сильную математическую операцию свёртки. Свёртка является линейным преобразованием. Математически свёртку можно записать следующим образом (рис. 1.7):

$$\boldsymbol{y}_{\boldsymbol{i,j}}^{\boldsymbol{k}} = \sum_{-\boldsymbol{d}\leq \boldsymbol{a,b}\leq \boldsymbol{d}} \boldsymbol{W}_{\boldsymbol{a,b}} \boldsymbol{x}_{\boldsymbol{i+a,j+b}}^{\boldsymbol{k}}.$$

где:

$\boldsymbol{y}_{\boldsymbol{i,j}}^{\boldsymbol{k}}$ – результат свёртки на слое k;

W – матрица весов (ядро) размером $(\boldsymbol{2d}+\boldsymbol{1})\times(\boldsymbol{2d}+\boldsymbol{1})$;

$\boldsymbol{x}^{\boldsymbol{k}}$– карта признаков на слое k.

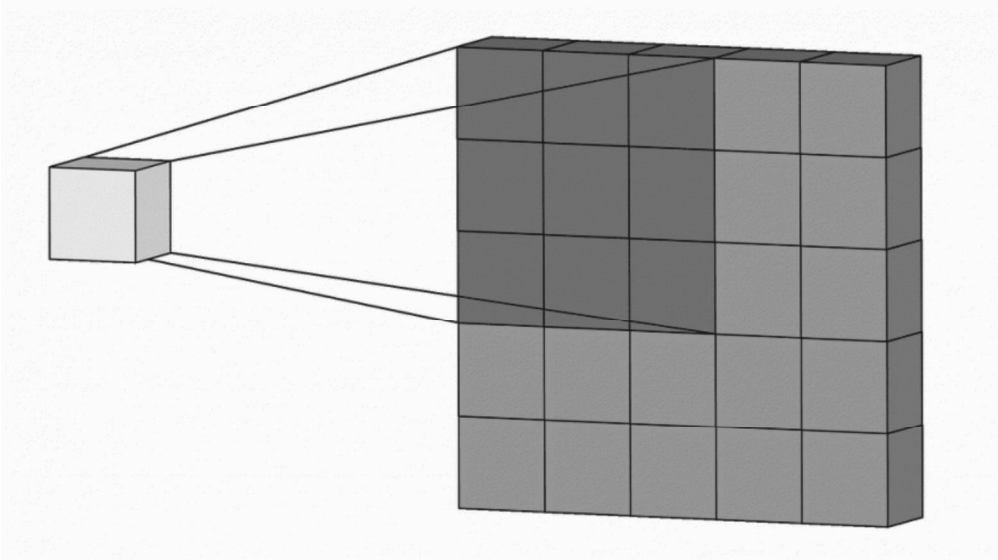

Рис. 1.7 Операция свёртки

Попробуем описать идею свёртки более понятно. По своей сути она состоит из следующих шагов:

1. Ядро представленное квадратной матрицей весов скользит по изображению. Это сделано для того, чтобы искать локальные признаки, независимо от того, где они расположены на изображении. Каждое такое ядро является по своей сути извлекателем признаков из изображения. Матрица ядра скалярно умножается на соответствующий квадрат (подматрицу) изображения;
2. После работы каждого ядра (свёрточного нейрона) на выходе получается так называемая карта признаков (feature map). Эти карты признаков представляют собой соответствующие каналы выхода слоя, которые отправляются на вход последующему;
3. Далее к картам признаков применяется нелинейная функция активации, как правило ReLU;
4. Затем происходит субдискретизация. Так как нам важнее факт самого наличия признака, нежели его конкретное расположение на изображении.

И также кажется важным подсветить в свёрточных нейронных сетях этап субдискретизации. Чаще всего в качестве инструмента операции «подвыборки» используется подход max-pooling (рис. 1.8). Также порой применяют avg-pooling, но исследования по сравнению этих двух вариантов показывают превосходство варианта с максимумом. Формализовать max-pooling можно так:

$$x_{i,j}^{k+1} = \max_{-d \le a \le d, -d \le b \le d} z_{i+a,j+b}^{k}.$$

где:

$\boldsymbol{d}$ – размер окна субдискретизации;

$z_{i,j}^{k} = a(y_{i,j}^{k})$ – результат после работы функции активации $\boldsymbol{a}$.

Последнее, на что бы хотелось обратить внимание в этой главе – работа с цветными изображениями, представленными 3 каналами RGB.

На практике у одного фильтра для каждого цветового канала есть свое ядро. Это позволяет уделить больше внимания какому-то конкретному каналу. После получения карт признаков по нескольким каналам от одного фильтра – они просто суммируются между собой в одну карту признаков и отдаются дальше на вход следующему слою, как новый канал.

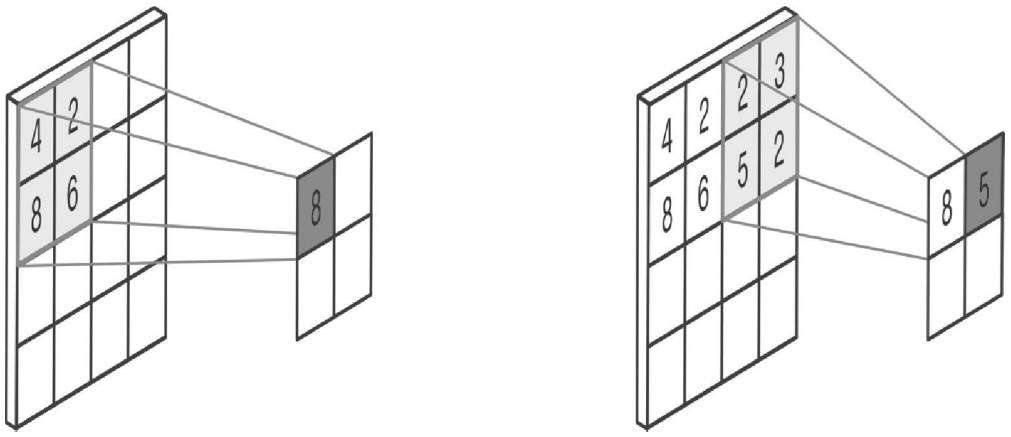

Рис. 1.8 Операция max-pooling

1.2 Системы распознавания лиц

В этом разделе описывается задача распознавания лиц. Рассматривается ее подзадача – обнаружение лиц. Проводится краткий обзор архитектур, являющихся передовыми в данной области. Определяется математическая постановка задачи. Также рассматриваются особенности применения таких моделей.

1.2.1 Задача обнаружения лиц

Прежде чем приступать к рассмотрению основной задачи – распознаванию лиц, нужно решить ее вспомогательную подзадачу – детектирование или обнаружение лиц на изображениях.

Обнаружение лиц часто используется в задачах связанных с биометрией, и выступает как часть систем по распознаванию лиц. Прежде чем изображение подается модели, обученной для распознавания лиц, осуществляется выделение области, так называемой рамки (box), соответствующей лицу на фотографии, для этого решается задача регрессии. И уже обрезанное изображение подается на вход основной сети.

Данная задача решается с помощью глубоких нейронных сетей уже рассмотренных в этой работе ранее. В последнее время свёрточные нейронные сети достигли значительного прогресс в различных задачах компьютерного зрения.

Существует множество архитектур для решения данной задачи и все они используют разные подходы и разные архитектуры, вот некоторые из них:

- MTCNN;
- TinaFace;
- AlnnoFace;
- DSFD;
- SRN;
- RetinaFace;

Рассмотрим задачу определения лиц на примере глубокой нейронной сети архитектуры MTCNN (рис 1.9).

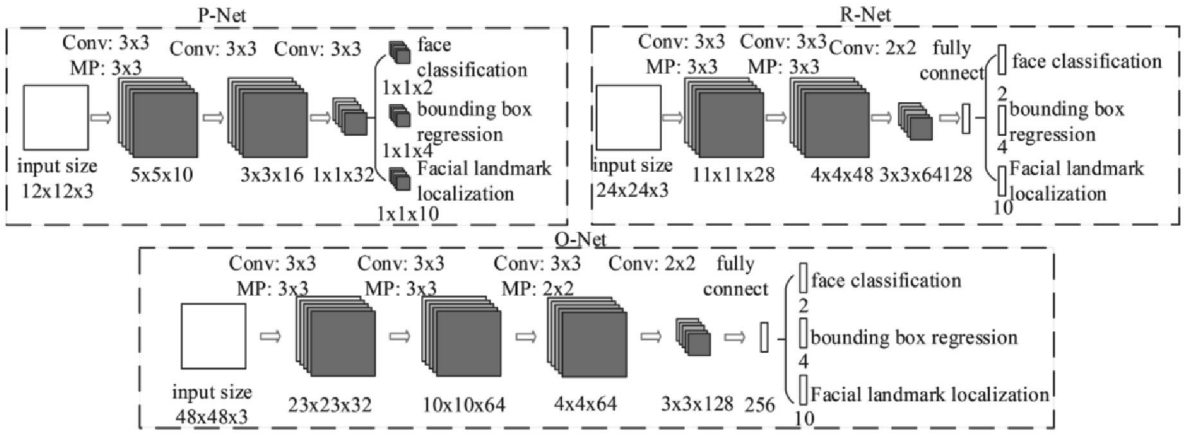

Рис. 1.9 Архитектура MTCNN

Сперва входное изображение многократно масштабируется, создавая тем самым набор одинаковых изображений с разным размером. Его называются пирамидой, так как последовательность уменьшающихся картинок напоминает эту форму. Далее полученная пирамида подается на вход трёх шаговому каскадному конвееру, состоящему из: P-net (proposal network), R-net (refinement network), O-net (output network).

P-net это полная свёрточная сеть, которая используется для получения рамок-кандидатов и их векторов регрессии рамок. Потом происходит оценка этих векторов регрессии и по ним делается калибровка рамок-кандидатов. После используется не максимальное подавление (NMS), чтобы объединить сильно перекрывающихся кандидатов. Иными словами на этом шаге среди большого количества потенциальных рамок происходит уточнение их краев и объединение рамок возле одного и того же лица.

Затем все кандидаты передаются в R-net. Её задача отклонить большое количество ложных кандидатов, откалибровать векторы-регрессии для этих рамок и слить одинаковых кандидатов опять с помощью не максимального подавления.

Далее следует этап O-net. Этот этап похож на второй, но на этом этапе мы стремимся более подробно описать лицо. В частности, сеть определяет позиции пяти ориентиров на лицах.

Таким образом решается сразу три задачи: классификация лица или не лица, регрессия координат лица, определение ориентиров лица.

Первая задача формализуется следующим образом:

$$\boldsymbol{L_i^{det} = -\left(y_i^{det} \log(p_i) + \left(1 - y_i^{det}\right)(1 - \log(p_i))\right).}$$

Вторая задача формализуется следующим образом:

$$\boldsymbol{L_i^{box} = \left\|\hat{y}_i^{box} - y_i^{box}\right\|_2^2.}$$

где:

$\boldsymbol{y_i^{box} \in R^4}$.

Третья задача формализуется следующим образом:

$$\boldsymbol{L_i^{landmark} = \left\|\hat{y}_i^{landmark} - y_i^{landmark}\right\|_2^2.}$$

где:

$\boldsymbol{y_i^{landmark} \in R^{10}}$.

Тогда функция потерь будет выглядеть следующим образом:

$$\boldsymbol{Q = \sum_{i=1}^{N} \sum_{j \in \{det, box, landmarks\}} \alpha_j \beta_i^j L_i^j.}$$

где:

N – количество примеров в обучающей выборке;

$\boldsymbol{\alpha_j}$ – важность задачи ($\boldsymbol{\alpha_{det} = 1, \alpha_{box} = 0.5, \alpha_{landmark} = 0.5}$ для P-net, R-net и $\boldsymbol{\alpha_{det} = 1, \alpha_{box} = 0.5, \alpha_{landmark} = 1}$ для O-net);

$\boldsymbol{\beta_i^j \in \{0, 1\}}$ – индикатор типа образца.

Теперь, когда была поставлена вспомогательная задача – обнаружение лица, можно перейти к следующей задаче – распознаванию лиц. И хотя в данной работе не затрагивается этап обнаружения лица, эта информация полезна для представления общей картины работы систем распознавания лиц.

1.2.2 Задача распознавания лиц

Задача распознавания лиц достигла больших успехов за последнее время. Такой прогресс вызван несколькими очень значительными факторами. К ним можно отнести следующие направления: появление большого количества наборов данных в высоком разрешении, разработка эффективных функций потерь для данной задачи, изобретение новых продвинутых архитектур глубоких нейронных сетей.

Наборы данных с высоким разрешением играют важную роль в задаче глубокого распознавания лиц. Например данные CAISA-WebFace это первый широко используемый набор данных для обучения глубоких нейронных сетей для распознавания лиц, он содержит 490000 изображений 10575 знаменитостей. MS-Celeb-1M является первых публичным набором данных их миллиона лиц. VGGFace2 имеет 3.31 миллиона изображений 9131 личности с большой вариацией положений, возраста, национальности и профессии. IMDb-Face набор данных из миллионов примеров, с низким уровнем шума в метках, содержит он 1.7 миллионов изображений 59000 личностей.

Следующей важной точкой роста качества моделей по распознаванию лиц являются эффективные функции потерь для обучения. Некоторые из них основываются на обучении с евклидовой метрикой. Некоторые функции потерь изменяют softmax с помощью добавления веса или нормализации признаков. Другой мощный вид функций потерь – это softmax с большим зазором: SphereFace, CosFace, ArcFace. Функции потерь такого вида значительно повышают производительность таких моделей. Поэтому они наиболее популярны и эффективны в задаче глубокого распознавания лиц.

Ожидаемо, что исследование в области архитектур нейронных сетей также вносит свой весомый вклад в данную задачу. К таким архитектурам, например, можно отнести:

- ResNet 64 слоя в SphereNet;
- Inception-ResNet в Adacos;

- ResNet и SENet в ArcFace;
- MobileNet.

Таким образом, многообразие моделей достигается путем различных вариацией этих 3 компонент: наборов данных, функций обучения и архитектур. Эксперименты по сочетанию этих компонент происходят активно и сейчас. Исследователи постоянно соревнуются между собой в данной задаче и поисках лучшей модели. [10]

Давайте теперь рассмотрим подробнее постановку задачи распознавания лиц.

Пусть $\boldsymbol{D} = \{\boldsymbol{x^i}, \boldsymbol{y^i}\}_{\boldsymbol{i=\overline{1,N}}}$ – это набор размеченных данных. Где $\boldsymbol{x^i}$ это изображение, $\boldsymbol{y^i} \in \{\boldsymbol{1, 2, 3, \dots, C}\}$ – это метка, $\boldsymbol{C}$ – это количество личностей. Во время обучения глубокая нейронная сеть с параметрами $\boldsymbol{\theta_N}$ и размером мини-батча N обучается на наборе данных $\boldsymbol{D}$, минимизируя функцию кросс-энтропии:

$$\boldsymbol{J(x^i, y^i) = -\frac{1}{N}\sum_{i=1}^{N} \log p(y^i | x^i, \theta_N)}.$$

Предсказание получается следующим образом:

$$\boldsymbol{p(y^i | x^i, \theta_N) = \frac{e^{W_{y^i}^T x_i + b_{y^i}}}{\sum_{j=1}^{C} e^{W_j^T x_i + b_j}}}.$$

где:

$\boldsymbol{x_i} \in \boldsymbol{R^d}$– векторное представление (вложение) примера $\boldsymbol{x^i}$;

$\boldsymbol{y^i}$– метка i-ого примера;

$\boldsymbol{W_j} \in \boldsymbol{R^d}$– j-ый столбец весов последнего полносвязного слоя;

$\boldsymbol{b_j} \in \boldsymbol{R^C}$– смещение.

В процессе обучения сеть получает входное изображение $\boldsymbol{x^i}$ и возвращает вероятность $\boldsymbol{p(y^i | x^i, \theta_N)}$ и предсказанную метку. В процессе использования модели (в тестовом процессе) модель используется как средство для извлечения признаков – представление изображения внутри сети, так

называемый embedding или вложение. Это делается очень просто. Последний слой сети softmax обрезается. На выходе работы сети извлекается нормализованное внутреннее представление изображения. Затем это вложение сравнивается с базой данных уже готовых и размеченных представлений лиц людей, например с помощью евклидовой метрики.

1.3 Генерация атакующих примеров

В этой главе будет рассмотрен процесс создания атакующих примеров для моделей машинного обучения, в частности глубоких нейронных сетей. Сначала вводится понятие состязательных атак, подсвечивается их актуальность. Затем понятие атакующего примера формализуется. Проводится обзор алгоритмов создания атакующих примеров. Рассматривается специфика создания примеров для систем по распознаванию лиц. Все это необходимо для демонстрации алгоритма в практической части. Так как там будет предлагаться улучшение генерации примеров на основе теоритической базы, описанной в этой главе.

1.3.1 Атакующие примеры

Главным предметом исследования данной работы являются атакующие примеры. Ещё в 2014 году было показано [9], что глубокие нейронные сети не так надежны как кажется на первый взгляд. Оказывает, их можно обманывать. Идея заключается в создании таких зашумлённых входных данных, которые будут способны заставлять модель свой вердикт в сравнении с предсказанием на чистых данных.

Изображения, способные «обманывать» модель машинного обучения с помощью контролируемого шума мы и будем называть – атакующими примерами. Важно, что на шум накладывается ограничение – он должен быть мало заметен для человеческого восприятия.

Направление изучающее вопрос создания атакующих примеров и защиты от них называется – состязательные атаки (adversarial attack).

Давайте формализуем понятие нетаргетированного атакующего примера.

Пусть дан классификатор $\boldsymbol{f(x)}: \boldsymbol{x \in X \rightarrow y \in C}$, который предсказывает для входных данных $\boldsymbol{x}$ выходную метку $\boldsymbol{y}$. Требуется найти пример $\boldsymbol{x^*}$ в окрестности примера $\boldsymbol{x}$, такой что для пары $(\boldsymbol{x}, \boldsymbol{y})$ выполняется следующее: $\boldsymbol{f(x^*) \neq y, \|x^* - x\|_p \leq \varepsilon}$, где $\boldsymbol{p}$ может быть $\boldsymbol{0, 1, 2, \infty}$.

В наше время технологии на основе глубоких нейронных сетей используются повсеместно. Множество применений данных технологиях в различных сферах человеческой деятельности делает их потенциально опасными. Голосовые помощники, машины с автопилотом, контроль сетевого взаимодействия и множество других приложений могут быть потенциально уязвимы к атакующим примерам. Поскольку данные часто приходят не из надежных источников, то любая компьютерная система уязвима к таким атакам. Таким образом актуальность изучения данного вопроса говорит сама за себя. Один из вариантов классификации такого вида атак следующий: уклонение, влияние, дезориентация.

Под уклонением понимается сокрытие контента от автоматического цифрового анализа. Например это обход веб-фильтров, детектирующих технологий в сетевых потоках, видеонаблюдение, конфиденциальность и приватность. Данный вид атак и изучается в рамках данной работы и систем по распознаванию лиц.

Следующий тип атак – это влияние. Воздействие на автоматизированные решения для получения личной, коммерческой или организационной выгоды. Например это можно использовать для продвижения сайтов в интернете, улучшения репутации и бренда. Более опасные потенциальные атаки – выдача себя за другого человека. Например, наложения шума на запись голоса так, чтобы система по распознаванию человека в банке приняла самозванца за атакуемую цель.

И третий тип атаки – это дезориентация. Создание хаоса с целью дискредитировать или нарушить работу организации. Например был проведен ряд исследований с наклейками на автомобильных знаках [12]. Удалось

показать, что машины с автопилотом могут быть «сведены с ума» при помощи специально нанесенных меток. Или другим примером может послужить создание такого звукового сигнала, который будет вводить в заблуждение голосового помощника, тем самым создавая хаос.

Таким образом надежность моделей машинного обучения находится под большим вопросом. Изучая данный вопрос и придумывая все более новые и новые атаки, исследователи смогут понимать с чем они борются и создавать соответствующие варианты защиты. Пути защиты от атакующих примеров мы рассмотрим обзорно немного позже. Например, в данной работе изучается создание атакующих примеров, но используется оно в качестве защитного механизма. Поэтому важно понимать, что «атака» это не всегда означает что-то плохое.

Так например атака на системы по распознаванию лиц поможет сохранить приватность и конфиденциальность человека. Или размещение атакованных примеров на сайте продавца позволит ему защититься от потенциального воровства объявлений другими его конкурентами. Последний вариант уже имел место в практике и был успешно применен в реальных «боевых» условиях.

Атакующие примеры классически между собой делятся на атаки белого ящика и атаки чёрного ящика.

Вариант белого ящика подразумевает абсолютное знание об атакуемой системе. Это знание может включать в себя: архитектуру глубокой нейронной сети, веса сети, обучающую выборку, методы защиты сети и так далее. Очевидно, что такое знание на практике очень тяжело получить. Но работать с атаками белого ящика гораздо проще.

Второй вариант – это атаки чёрного ящика. Данный типа атаки подразумевает, что знание об атакуемой модели отсутствует. Но зато существует доступ к ней. То есть атакующий может получить доступ к выходным данным модели (метка или показатель достоверности, например вероятность класса). Таким образом этот подход основывается на запросных

методах. Тут можно либо численно оценивать градиент, либо строить новую модель по полученным предсказаниям, пытаясь аппроксимировать атакуемую модель. К сожалению для создания одного примера требуется много запросов, что может вызвать подозрение со стороны сервиса. Значит этот подход гораздо лучше применяется на практике, но имеет определенные ограничения.

Таким образом имеется два возможных пути создания атак и у каждого есть свои плюсы и минусы.

Неожиданным третьим решением выступает переносимость атакующих примеров. В работе [10] показано, что атакующие примеры имеют свойство переносимости на различные модели, решающие одни задачи. Так атакующий пример созданный под одну архитектуру глубокой нейронной сети могут быть перенесены на другую архитектуру. В этой же и другой работе [11] также предлагаются возможные варианты по улучшение переносимости таких атак.

И такой третий способ, можно называть его «серым ящиком», сочетает в себе плюсы двух рассмотренных ранее подходов, если выполняется предположение о переносимости. В данной работе будет рассматриваться как раз он, так как в работе [10] было показано, что переносимость атакующих примеров высока среди моделей распознавания лиц.

Также в данной работе будет предлагаться новый вероятностный метод аугментации изображений, улучшающий переносимость примеров. Подробнее об этом можно будет узнать в главе 2.4.

Часто атакующие примеры разделяют еще на 2 подмножества: таргетированные и не таргетированные атаки.

Тарегитрованные атаки, это когда шум, добавляемый к входным данным меняет предсказание модели на конкретное другое. Желаемое новое предсказание задает атакующий. Так, например, можно заставить систему по распознаванию лиц видеть в изображении человека совсем другую конкретную личность.

Не таргетированные атаки это соответственно, когда шум, добавляемый к входным данным меняет предсказание модели на любое другое. Какое

именно в данном случае не важно. Данный вид атак исследуется в данной работе. Так как нам важно сбить распознавательную способность системы и не важно в какую сторону, главное, чтобы она больше не могла определять человека на изображении.

Гипотеза о переносимости примеров опирается на то, что если вредоносные входные данные создаются с помощью некоторого набора замещающих моделей, то такой образ, срабатывающий против ряда замещающих моделей, должен быть достаточно гибким для переноса на другую, целевую модель.

Что интересно, разные обучающие наборы данных могут занимать примерно одинаковые области входного пространства даже при определенных различиях между ними, если они были созданы на основе одной и той же информации о физическом мире. При этом области входного пространства с недостаточным количеством репрезентативных данных (соответствующие данным вне распределения), как правило, выглядят неизменно и не зависят от используемого обучающего набора данных. Кроме того, разные обучающие наборы данных могут иметь между собой определенное сходство, например одинаковый наклон камеры или одинаковые характеристики голоса. То есть разные обучающие наборы данных могут иметь примерно одинаковое распределение характерстик, признаков, и, что самое главное, вредоносных подпространств.

Единообразие вредоносных подпространств делает возможным проведение универсальной атаки с переносом. Хотя, что не удивительно, осуществить такую атаку гораздо труднее, она дает злоумышленнику большие возможности. Когда входной сигнал эффективен против различных моделей, злоумышленник может провести атаку против целой группы глубоких нейронных сетей.

Таким образом идея, лежащая в основе генерации атакующих примеров очень проста – изменение не вредоносного образа со смещением его в другую

часть входного пространства признаков таким образом, чтобы предсказания моделей изменились до максимального желаемого эффекта.

Также хочется упомянуть возможные способы защиты от атакующих примеров. Хоть они и не изучаются в данной работе, стоит их привести для полноты понимания данной области:

- обучение на атакующих примерах;
- дистилляция градиента;
- поиск примеров вне обучающего распределения;
- анализ разброса предсказаний с помощью dropout.

1.3.2 Методы создания атакующих примеров

Существует множество методов создания атакующих примеров для различных целей: атаки белого ящика, атаки чёрного ящика, таргетированные, не таргетированные и другие. Так как в работе рассматривается именно переносимость примеров, то в данной главе будет представлен обзор методов атаки белого ящика. Методы чёрного ящика хорошо описаны в статье [12], вот некоторые из них:

- SimBA;
- Zoo;
- Query reduction using finite differences;
- Local substitute model;
- GenAttacks.

Первый метод, который хотелось бы рассмотреть называется fast gradient sign method (FGSM). Его предложил Goodfellow [9] и заключался он в следующем: атакующий пример строился по одношаговому обновлению градиента в направлении знака градиента каждого пикселя. Также есть модификация метода fast target gradient sign method, он соответственно расширяет метод до таргетированной атаки.

При этом алгоритм FGSM не рассматривается как наилучший способ поиска вредоносных входных данных, его изначальная задача была продемонстрировать уязвимость глубоких нейронных сетей.

Алгоритм FGSM рассчитывает то направление во входном пространстве, движение по которому из места размещения изображения является кратчайшим путем к ошибочной классификации. Это направление рассчитывается с использованием градиентного спуска и практически такой же функции потерь, которая используется при обучении сети. На концептуальном уровне такой расчет направления можно рассматриваться просто как косвенный способ оценки крутизны контуров на ландшафте предсказаний в месте размещения изображения.

Для входного изображения x и глубокой нейронной сети с функцией потерь $\boldsymbol{J(\theta, x, y)}$ создавалась специальная маска, имеющая вид $\boldsymbol{\varepsilon\, sign(\nabla_x J(\theta, x, y))}$, где $\boldsymbol{\varepsilon}$ – уровень шума, $\boldsymbol{x}$ – входное изображение, $\boldsymbol{y}$ – метка изображения. И если для входного изображения $\boldsymbol{x}$ модель могла однозначно определить класс, то для изображения с маской $\boldsymbol{x + \varepsilon\, sign\big(\nabla_x J(\theta, x, y)\big)}$ модель уже ошибалась, выдавая не правильный вердикт.

То есть FGSM выглядит так:

$$\boldsymbol{x_{adv} = x + \varepsilon\, sign\big(\nabla_x J(\theta, x, y)\big).}$$

Важно, что градиент функции потерь здесь рассчитывается по пикселям входного изображения, а не по весам самой сети.

Определив направление, метод FGSM вносит крошечное искажение в каждое входное значение, добавляя его при положительном направлении вредоносного изменения и вычитая его в противном случае. При этом можно надеяться на то, что из-за этих изменений изображение лишь немного выйдет за границу области, относящейся к его правильной категории, тем самым обеспечив вредоносный образ.

Базовый принцип метода FGSM сводится к тому, чтобы изменять все входные значения, но на очень небольшую величину. В основе такого подхода лежит тот факт, что множество крошечных изменений вдоль каждой

координатной оси многомерного пространства может давать значительное изменение в совокупности; изменение множества пикселей на крошечную величину обеспечивает значительное смещение во входном пространстве. По сути данный метод сводится к минимизации максимального изменения отдельного пикселя, то есть использует $\boldsymbol{L}^{\infty}$ норму.

Несмотря на то, что эта атака выглядит просто, ей удаётся обманывать огромное количество моделей. Более того, этот метод лежит в основе множества других.

Рассмотрим basic iterative method (BIM) (он же PGD – project gradient attack), он расширяет идею FGSM применяя его несколько раз с маленьким шагом.

$$\boldsymbol{x_{adv,0}} = \boldsymbol{x},$$

$$\boldsymbol{x_{adv,N+1}} = \boldsymbol{C}\left(\boldsymbol{x_{adv,N}} + \boldsymbol{\varepsilon sign}\left(\boldsymbol{\nabla_{x_{adv,N}} J(\theta, x_{adv,N}, y)}\right)\right).$$

где:

$\boldsymbol{C}$ – функция отсечения значений, так, чтобы они оставались в $\boldsymbol{L}^{\infty}$ окрестности от оригинального изображения.

Общая особенность всех методов белого ящика является то, что они стремятся оптимизировать поиск во входном пространстве за счет использования вычислительно осуществимого алгоритма. Как правило большинство методов основано на градиенте.

1.3.3 Атакующие примеры для систем распознавания лиц

Теперь когда было введено понятие атакующих примеров и рассмотрен алгоритм по их созданию, можно сузить область применения данных методов до конкретной, интересующей нас в данной работе – системы распознавания лиц.

Первая особенность в домене распознавания лиц это то, что данная задача является открытой (open-set) задачей. Это означает, что данные, которые поступают в тестовом режиме на вход модели могут сильно отличаться от

распределения данных в тестовой выборке. Например в задаче распознавания лиц очевидно стоит ожидать большое количество личностей, которых не было в обучающем наборе данных во время эксплуатации модели. Поэтому обученные классификаторы для распознавания лиц применяются другим образом – они выступают в качестве способа извлечения признаков. И определение класса уже происходит с помощью заданной метрики, например косинусного расстояния. Напомним что расстояние смотрится между полученным вложением и базой размеченных вложений.

Тут исходя из открытой специфики задачи мы переходим к следующей особенности работы систем распознавания лиц. Дело в том, что они являются лишь способом извлекать признаки. И мы не опираемся на класс, предсказываемый последним softmax слоем. А это значит, что строить атаки от метки бессмысленно. И тут предлагается [10] начать строить атаки немного хитрее, уже от непосредственно от признакового представления – вложения.

Дана пара изображений лиц одного и того же человека $\{\boldsymbol{x^s}, \boldsymbol{x^t}\}$, происходит сравнение их внутренних нормализованных представлений $\boldsymbol{F(x^s)}$ и $\boldsymbol{F(x^t)}$. $\boldsymbol{F\left(x^i\right)}$ – это вектор представления изображения внутри сети после нормализации. Теперь запишем нашу задачу поиска атакующего примера:

$$\Delta \boldsymbol{x} = \boldsymbol{arg}\max_{\Delta \boldsymbol{x}} \|\boldsymbol{F}(\boldsymbol{x^s} + \Delta \boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x^t})\|_2 , \|\Delta \boldsymbol{x}\|_\infty < \boldsymbol{\varepsilon}.$$

где:

$\boldsymbol{\varepsilon}$ – ограничения на вносимые возмущения.

Выпишем новую функцию потерь:

$$\boldsymbol{J}(\boldsymbol{x^s} + \Delta \boldsymbol{x}, \boldsymbol{x^t}) = \|\boldsymbol{F}(\boldsymbol{x^s} + \Delta \boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x^t})\|_2.$$

И теперь перепишем basic iterative method но уже с атакой на уровне признаков:

$$\Delta \boldsymbol{x_0} = \boldsymbol{0},$$
$$\boldsymbol{g_{N+1}} = \nabla_{\boldsymbol{x^{(s)}} + \Delta \boldsymbol{x_N}} \boldsymbol{J}(\boldsymbol{x^s} + \Delta \boldsymbol{x_N}, \boldsymbol{x^t}),$$
$$\boldsymbol{x^s} + \Delta \boldsymbol{x_{N+1}} = \boldsymbol{C_{x^s, \varepsilon}}\left(\boldsymbol{x^s} + \Delta \boldsymbol{x_N} + \boldsymbol{sign}(\boldsymbol{g_{N+1}})\right).$$

где:

$\boldsymbol{C_{x^s,\varepsilon}(x') = \min(255, x + \varepsilon, \max(0, x - \varepsilon, x'))}$.

Теперь полученный алгоритм генерации атакующих примеров учитывает особенности работы систем по распознаванию лиц. Именно этот подход и применяется в практической части данной работы.

1.3.4 Улучшение переносимости атакующих примеров

О переносимости атакующих примеров уже было сказано в пункте 1.3.1. Сейчас будут рассмотрены способы по улучшению переносимости таких примеров.

Как было показано ранее в методах генерации атакующих примеров используется градиентный спуск. Одним из напрашивающихся подходов является идея модификации алгоритма градиентного спуска [13].

Идея метода запоминать изменение приращения на каждой итерации, затем рассчитывая новое изменение как сумму текущего градиента и предыдущих изменений. Данная модификация позволит стабилизировать обновления параметра.

Далее будем рассуждать в терминах изображения.

Алгоритм следующий:

Вход:

$\boldsymbol{f}$ – классификатор;

$\boldsymbol{J}$ – функция потерь;

$\boldsymbol{x}$ – оригинальное изображение;

y – метка оригинального изображения;

$\boldsymbol{\varepsilon}$ – размер возмущений;

$\boldsymbol{T}$ – количество итераций;

$\boldsymbol{\mu}$ – параметр отставания в моментум.

Выход:

$\boldsymbol{x}^*$ – атакующее изображение, такое что:

$$\|x^* - x\|_\infty \leq \varepsilon.$$

Алгоритм:

1. $\alpha = \frac{\varepsilon}{T}$.
2. $g_0 = 0$;
   $x_0^* = x$.
3. $for\ t = 0\ to\ T - 1\ do$
4. Подаём x_t^* на вход f, и получаем градиент $\nabla_x J(x_t^*, y)$;
5. Обновляем g_{t+1} следующим образом:

$$g_{t+1} = \mu \cdot g_t + \frac{\nabla_x J(x_t^*, y)}{\|\nabla_x J(x_t^*, y)\|_1};$$

6. Обновляем x_{t+1}^*:

$$x_{t+1}^* = x_t^* + \alpha \cdot sign(g_{t+1});$$

7. end for;
8. return $x^* = x_T^*$.

Таким образом, метод на основе моментум будет лучше переносим, так как не будет сваливаться в локальные минимумы и переобучаться на сети, к которой применяется атака белого ящика.

Еще одним интересным вариантом улучшения переносимости атакующих примеров является разнообразие входа [11]. Идея следующая – добавить в итерационный процесс вычисления атакующего примера изменения изображения-кандидата на каждой итерации. Данный подход будет подробнее описан в практической части. В данном исследовании будут предложены вероятностные аугментации и рекомендации по подбору гиперпараметров.

ПРАКТИЧЕСКАЯ ЧАСТЬ

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

Практическим результатом работы является предложенный и реализованный алгоритм, способный создавать атакующие примеры для систем распознавания лиц. Удалось улучшить переносимость атакующих примеров при помощи добавления вероятностных аугментаций над изображением. Также, в качестве дополнительного результата, предоставляются рекомендации по подбору значений гиперпараметров алгоритма генерации атакующих примеров.

2.1 Постановка задачи

Необходимо предложить и реализовать алгоритм генерации атакующих примеров для систем распознавания лиц.

Пусть дана модель распознавания лиц $\boldsymbol{f(x)}: \boldsymbol{x \in X \rightarrow y \in R^d}$, где $\boldsymbol{X}$ – множество лиц, а $\boldsymbol{R^d}$ – представление лица внутри нейронной сети (вложение), которая определяет вложение заданного лица. Требуется найти пример $\boldsymbol{x^* = x + \Delta x}$ в окрестности примера $\boldsymbol{x}$, такой что:

$$\boldsymbol{\Delta x = arg \max_{\Delta x} \|F(x + \Delta x) - F(x)\|_2 , \|\Delta x\|_\infty < \varepsilon.}$$

Важно, что эту задачу также можно переписать для пары лиц. Выглядит это следующим образом.

Пусть дана модель распознавания лиц $\boldsymbol{f(x)}: \boldsymbol{x \in X \rightarrow y \in R^d}$, где $\boldsymbol{X}$ – множество лиц, а $\boldsymbol{R^d}$ – представление лица внутри нейронной сети (вложение), которая определяет вложение заданного лица. Дана пара изображений лиц одного и того же человека $\{\boldsymbol{x^s}, \boldsymbol{x^t}\}$. Требуется найти изменения изображения $\boldsymbol{x^s}$ так, чтобы внутреннее нормализованное представление максимально отличалось от $\boldsymbol{x^t}$.

$$\boldsymbol{\Delta x = arg \max_{\Delta x} \|F(x^s + \Delta x) - F(x^t)\|_2 , \|\Delta x\|_\infty < \varepsilon.}$$

Как уже было показано в пункте 1.3.3, функцию потерь для данной задачи можно записать в следующем виде:

$$\boldsymbol{J}(\boldsymbol{x}^{\boldsymbol{s}} + \Delta\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{t}}) = \|\boldsymbol{F}(\boldsymbol{x}^{\boldsymbol{s}} + \Delta\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x}^{\boldsymbol{t}})\|_{\boldsymbol{2}}.$$

В качестве экспериментальной выборки для исследования был выбран набор данных MS-Celeb-1M. Из него была сделана выборка пар изображений 500 знаменитых личностей. Пример того, как выглядят данные по одной персоне в MS-Celeb-1M представлен ниже на рис. 2.1.

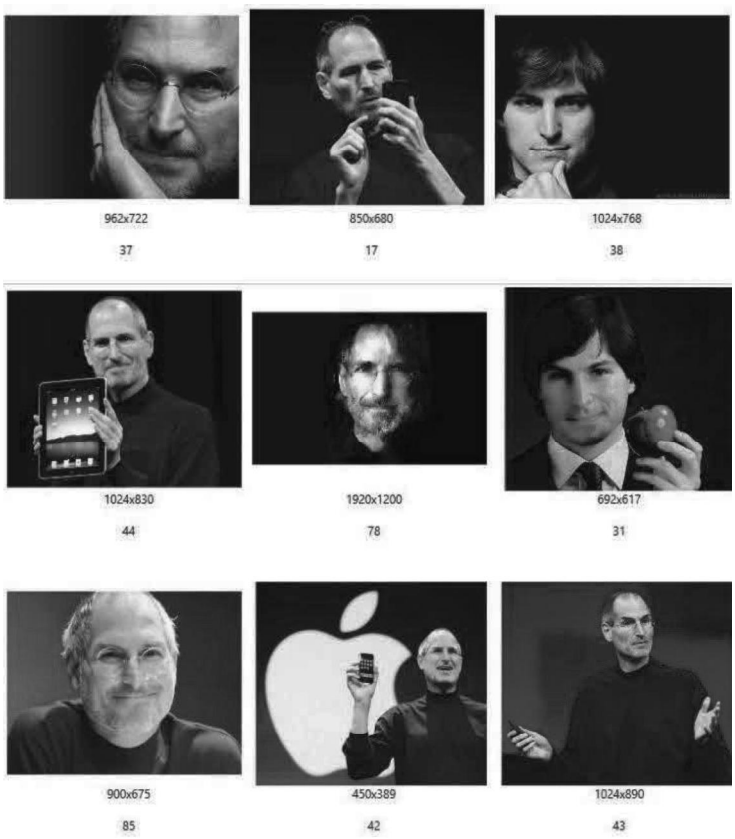

Рис. 2.1 Пример изображений человека из набора данных

Проверять работу алгоритма было решено на максимально репрезентативной основе: онлайн сервис по распознаванию лиц и другая модель. Для этого были выбраны следующие веб-сервис и модель соответственно:

- https://pimeyes.com/;
- Inception ResNet v1.

2.2 Модель распознавания лиц

Модель распознавания лиц, которая использовалась в качестве белого ящика – FaceNet модель. Она была уже предобучена и реализована на python3.5 с использованием фреймворка tensorflow 1.7. В качестве обучающей выборки для не выступал набор данных VGGFace2.

Отметим, что для сравнения бралась эта же архитектура, но обученная на другом наборе данных – CASIA-WebFace.

В качестве детектирующей технологии лиц используется сеть MTCNN. Она была рассмотрена ранее в пункте 1.2.1.

В основе FaceNet лежит идея сиамских нейронных сетей. То есть она состоит из двух одинаковых глубоких нейронных сетей. Важно отметить, что веса этих сетей совпадают. На вход подаются два изображения, они пропускаются через все слои FaceNet и в конце их векторное представление сравнивается друг с другом. То есть происходит вычисление расстояния между двумя выходами.

В качестве функции потерь в FaceNet используется триплет функция (Triplet Loss). Она максимизирует расстояние между разными изображением и якорем, и минимизирует для тех, кто содержит похожие лица.

$$\boldsymbol{TripletLoss} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \mathbf{max}\,(\mathbf{0}, \left\|\boldsymbol{f_i^a} - \boldsymbol{f_i^p}\right\|_{\mathbf{2}}^{\mathbf{2}} - \|\boldsymbol{f_i^a} - \boldsymbol{f_i^n}\|_{\mathbf{2}}^{\mathbf{2}} + \boldsymbol{\alpha}).$$

где:

$\boldsymbol{f_i^a}$ – вложение якоря;

$\boldsymbol{f_i^p}$ – вложение похожих лиц;

$\boldsymbol{f_i^n}$ – вложение разных лиц;

$\boldsymbol{\alpha}$ – константа для оптимизации.

Для генерации атакующих примеров последний слой сети нам не понадобится. Поэтому он просто отрезается, и на выходе сети получается внутреннее представление изображения – вложение.

2.3 Алгоритм генерации атакующих примеров

После того, как сеть для атаки белого ящика подготовлена можно приступать к созданию алгоритма генерации атакующих примеров.

За основу атаки берётся алгоритм, описанный в пункте 1.3.4. За исключением того, что теперь атака будет строиться не в классическом варианте от метки, а как было показано ранее наиболее эффективным способом для систем распознавания лиц – от внутреннего представления. Также стоит отметить, что сразу будет реализовываться подход с моментум. Это должно повысить переносимость созданных атакующих примеров.

Запишем новую функцию потерь:

$$\boldsymbol{J}(\boldsymbol{x^s} + \Delta\boldsymbol{x}, \boldsymbol{x^t}) = \|\boldsymbol{F}(\boldsymbol{x^s} + \Delta\boldsymbol{x}) - \boldsymbol{F}(\boldsymbol{x^t})\|_2$$

Полученный алгоритм представлен ниже:

Вход:

$\boldsymbol{f}$ – классификатор;

$\boldsymbol{J}$ – функция потерь;

$\boldsymbol{x}$ – оригинальное изображение;

$\boldsymbol{\varepsilon}$ – размер возмущений;

$\boldsymbol{T}$ – количество итераций;

$\boldsymbol{\mu}$ – параметр отставания в моментум.

Выход:

$\boldsymbol{x}^*$ – атакующее изображение, такое что:

$$\|\boldsymbol{x}^* - \boldsymbol{x}\|_\infty \leq \boldsymbol{\varepsilon}.$$

Алгоритм:

1. $\boldsymbol{\alpha} = \frac{\boldsymbol{\varepsilon}}{\boldsymbol{T}}$.
2. $\boldsymbol{g_0} = \boldsymbol{0}$;

   $\boldsymbol{x_0^*} = \boldsymbol{x}$.
3. $\boldsymbol{for\ t\ =\ 0\ to\ T\ -1\ do}$

4. Подаём $\boldsymbol{x_t^*}$ на вход $\boldsymbol{f}$, и получаем градиент $\boldsymbol{\nabla_x J(\, x_t^*, x)}$;
5. Обновляем $\boldsymbol{g_{t+1}}$ следующим образом:

$$\boldsymbol{g_{t+1} = \mu \cdot g_t + \frac{\nabla_x J(\, x_t^*, x)}{\|\nabla_x J(\, x_t^*, x)\|_1}};$$

6. Обновляем $\boldsymbol{x_{t+1}^*}$:

$$\boldsymbol{x_{t+1}^* = x_t^* + \Pi_{x+S}(\alpha \cdot sign(g_{t+1}))};$$

7. end for;
8. return $\boldsymbol{x^* = x_T^*}$.

Данный алгоритм был реализован на языке Python3.7 с применением фреймворка tensorflow1.7 и библиотеки numpy. Программа создающая атакующий пример представлена в виде python скрипта. Доступные ключи для запуска:

- --attack

  Путь до файла с изображением, который нужно «анонимизировать»;
- --output

  Путь до файла с полученным изображением;
- --eps

  Максимальное изменение пикселя.

Полученные результаты описаны в разделе 2.5.

2.4 Переносимость атакующих примеров

В данной работе предлагается способ улучшения переносимости атакующих примеров, а именно вероятностная аугментация.

Смысл этой аугментации состоит в том, чтобы избавиться от переобучения создания атакующего примера на суррогатной модели. Тем самым сделав его более генерализованным.

В пункте 1.1.5 уже описывались возможные аугментации над изображением. В данной работе предлагается рассмотреть изменение пропорций изображения и поворот на случайный угол.

Перепишем алгоритм в новом виде:

Вход:

$\boldsymbol{f}$ – классификатор;

$\boldsymbol{J}$ – функция потерь;

$\boldsymbol{x}$ – оригинальное изображение;

$\boldsymbol{\varepsilon}$ – размер возмущений;

$\boldsymbol{T}$ – количество итераций;

$\boldsymbol{\mu}$ – параметр отставания в моментум.

Выход:

$\boldsymbol{x}^*$ – атакующее изображение, такое что:

$$\|\boldsymbol{x}^* - \boldsymbol{x}\|_\infty \leq \boldsymbol{\varepsilon}.$$

Алгоритм:

9. $\boldsymbol{\alpha} = \frac{\boldsymbol{\varepsilon}}{\boldsymbol{T}}$.

10. $\boldsymbol{g_0} = \mathbf{0}$;

$\boldsymbol{x_0^*} = \boldsymbol{x}$.

11. $\boldsymbol{for\ t\ =\ 0\ to\ T\ -1\ do}$

12. $\boldsymbol{x_t^*} = \boldsymbol{T}(\boldsymbol{x_t^*})$

13. Подаём $\boldsymbol{x_t^*}$ на вход $\boldsymbol{f}$, и получаем градиент $\boldsymbol{\nabla_x J}(\boldsymbol{x_t^*}, \boldsymbol{x})$;

14. Обновляем $\boldsymbol{g_{t+1}}$ следующим образом:

$$\boldsymbol{g_{t+1}} = \boldsymbol{\mu} \cdot \boldsymbol{g_t} + \frac{\boldsymbol{\nabla_x J}(\boldsymbol{x_t^*}, \boldsymbol{x})}{\|\boldsymbol{\nabla_x J}(\boldsymbol{x_t^*}, \boldsymbol{x})\|_1};$$

15. Обновляем $\boldsymbol{x_{t+1}^*}$:

$$\boldsymbol{x_{t+1}^*} = \boldsymbol{x} + \Pi_{\boldsymbol{x}+\boldsymbol{S}}(\boldsymbol{\alpha} \cdot \boldsymbol{sign}(\boldsymbol{g_{t+1}}));$$

16. end for;

17. return $\boldsymbol{x}^* = \boldsymbol{x_T^*}$.

где:

$$\boldsymbol{T}(\boldsymbol{x}_t^*; \mathbf{p}; \mathbf{r}; \boldsymbol{\alpha}) = \begin{cases} \boldsymbol{Aug}(\boldsymbol{x}_t^*; \mathbf{r}; \alpha), \text{ с вероятностью } p \\ x_t^*, \text{с вероятностью } 1 - p \end{cases};$$

$$Aug(x_t^*; \mathrm{r}; \alpha) = \begin{cases} Resize(x_t^*; r), \text{ с вероятностью } 0.5 \\ Rotate(x_t^*; \alpha), \text{с вероятностью } 0.5 \end{cases};$$

$$r \sim U(135,\ 160);$$

$$\alpha \sim U(-15,\ 15).$$

Диапазон для равномерного распределения подобран эмпирически, исходя из специфики работы с изображением.

Данный алгоритм был также реализован как и предыдущий с использованием таких же инструментов и фреймворков. Поддерживает те же параметры скрипта, как и те, что были описаны ранее в пункте 2.3.

Полученные результаты описаны в разделе 2.5.

2.5 Описание результатов

В качестве метрики будем рассматривать долю успехов – это количество раз, когда атакующие примеры сработали и человека не удалось распознать.

Как говорилось ранее, тестирование проводилось на веб-сервисе по распознаванию лиц и на отдельной модели со схожей архитектурой, но другой обучающей выборкой.

В ситуации со сравнением с веб-сервисом узнать качество не так просто, так как это требует отдельной разметки – перепроверки, удалось ли сбить распознавательную способность сервиса. Но потратив время на ручную разметку удалось получить результат, представленный ниже в таблице.

При оценки в сравнении с отдельной моделью удалось проще оценить долю успехов. Зафиксировав степень уверенности (косинусное расстояние между вложениями) в опознанном лице на уровне 0.6 (ниже считаем, что распознавание сбилось), удалось получить результаты представленные ниже. Также удалось провести анализ вероятности случайной аугментации.

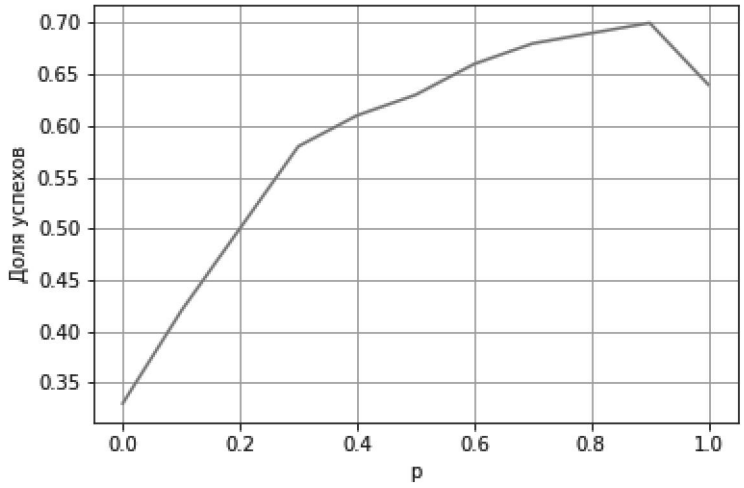

Рис. 2.2 Зависимость вероятности аугментации и доли успешных примеров

Из результатов (рис. 2.2) оценки вероятности случайной аугментации видно, что на уровне $p = 0.9$ удаётся добиться примерно 70% доли успехов в генерации атакующих примеров для отдельной модели. Видно, что при полном отсутствии вероятностных аугментаций ($p = 0.0$) доля успехов была бы в 2 раза ниже. Такая низкая доля успехов для базового подхода может быть вызвана тем, что наборы данных, на которых были обучены суррогатная и атакуемая модели сильно отличаются и формируют различное внутреннее пространство. Это может быть темой отдельного исследования в дальнейшем.

Таблица 2.1 Процент успешных атакующих примеров

	Веб-сервис	Другая модель
Базовый алгоритм	34.5	53.5
Алгоритм со случайной аугментацией	55.5	66.1
Алгоритм со случайной аугментацией и поворотом	59.4	70.0

Из результатов таблицы 2.1 можно сделать следующие выводы:

- случайные аугментации увеличивают долю успешных атакующих примеров;
- оптимальный гиперпараметр случайной аугментации $p = 0.9$;
- у веб-сервиса сложнее сбить способность распознавания.

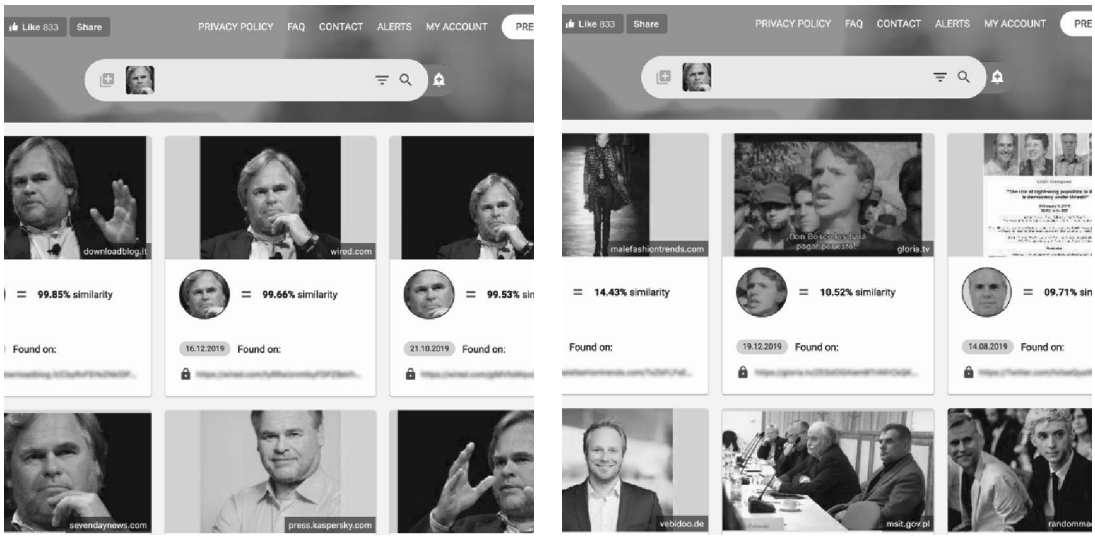

Рис. 2.3 Пример проверки изображения на веб-сервисе

Выше на рис. 2.3 представлен пример атакующего примера для веб-сервиса.

В качестве возможных дальнейших шагов можно предложить следующие направления:

- рассмотрение других аугментаций изображения;
- применение других методов генерации атакующих примеров;
- изучение влияния набора данных на переносимость атакующих примеров;
- реализация решения в виде продукта (например мобильное приложение).

ЗАКЛЮЧЕНИЕ

В представленной научной работе решены все поставленные задачи в полном объёме.

- исследована генерация атакующих примеров для глубоких нейронных сетей в задаче распознавания лиц;
- исследована переносимость атакующих примеров между различными глубокими нейронными сетями в задаче распознавания лиц;
- разработан алгоритм генерации атакующих примеров для глубоких нейронных сетей в задаче распознавания лиц;
- повышена эффективность атакующих примеров с помощью предложенного подхода вероятностных аугментаций с поворотом;
- даны рекомендации по подбору гипер-параметров для данного алгоритма;
- осуществлена компьютерная реализация предложенного алгоритма;
- улучшена переносимость атакующих примеров;
- предложены шаги для дальнейших исследований в данном направлении.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Айвазян С. А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика: классификация и снижение размерности. — М.: Финансы и статистика, 1989 – 607 с.
2. Мхитарян В.С., Архипова М.Ю., Миронкина Ю.Н., Сиротин В.П., Дуброва Т.А. Анализ данных. — М.: Юрайт, 2018 – 490 с.
3. Machinelearning.ru [Электронный ресурс]. URL: http://www.machinelearning.ru/wiki/ (дата обращения 21.03.2020).
4. Mitchell, T. M. Machine Learning. – McGraw-Hill, 1997. – P. 432 .
5. Bishop C. M. Pattern Recognition and Machine Learning. – Springer, 2006. – P. 749.
6. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. – Springer, 2001. – P 764.
7. M. Pautov, G. Melnikov, E. Kaziakhmedov, K. Kireev, A. Petiushko, On adversarial patches: real-world attack on ArcFace-100 face recognition system // arXiv preprint, arXiv: 1910.07067 — 2020. — 6 pp.
8. Mei Wang, Weihong Deng, Deep Face Recognition: A Survey // arXiv preprint, arXiv: 1804.06655 — 2020. — 31 pp.
9. I. J. Goodfellow, J. Shlens, and C. Szegedy, Explaining and harnessing adversarial examples, CoRR, vol. abs/1412.6572, 2014
10. Yaoyao Zhong, Weihong Deng, Towards Transferable Adversarial Attack against Deep Face Recognition // arXiv preprint, arXiv: 2004.05790 — 2020. — 15 pp.
11. Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, Alan Yuil, Improving Transferability of Adversarial Examples with Input Diversity // arXiv preprint, arXiv: 1803.06978 — 2019. — 10 pp.
12. Siddhant Bhambri, Sumanyu Muku, Avinash Tulasi, Arun Balaji Buduru, A Survey of Black-Box Adversarial Attacks on Computer Vision Models // arXiv preprint, arXiv: 1912.01667 — 2020. — 33 pp.

13. Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, Jianguo Li, Boosting Adversarial Attacks with Momentum // arXiv preprint, arXiv: 1710.06081 — 2018. — 12 pp.