НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«МЭИ»

УДК: 621.398

Институт автоматики и вычислительной техники
Кафедра прикладной математики
Направление
01.04.02 «Прикладная математика и информатика»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Программа: Математическое и программное обеспечение вычислительных машин и компьютерных сетей

Тема:
Разработка и программная реализация искусственной нейронной сети для идентификации человека по изображению лица

Студент А-13м-16 Апарнев А. Н.
группа подпись фамилия, и., о.,

Научный руководитель доцент, к.т.н., Бартеньев О. В.
должность звание подпись фамилия, и., о.,

Консультант_____
должность звание подпись фамилия, и., о.,

Консультант_____
должность звание подпись фамилия, и., о.,

Магистерская диссертация допущена к защите

Зав.кафедрой ПМ: д.т.н., профессор А.П.Еремеев

Дата_____

МОСКВА 2018

АННОТАЦИЯ

Автор: Апарнев А. Н.

Тема: Разработка и программная реализация искусственной нейронной сети для идентификации человека по изображению лица.

Объём: 108 стр., 14 рис., 5 табл., 43 источника.

Ключевые слова: БИОМЕТРИЧЕСКАЯ ИДЕНТИФИКАЦИЯ, ИДЕНТИФИКАЦИЯ, ВЕРИФИКАЦИЯ, РАСПОЗНАВАНИЕ ЛИЦА, ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, ГЛУБОКИЕ СВЁРТОЧНЫЕ ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, ФУНКЦИЯ ОШИБКИ.

В работе исследованы способы идентификации человека по лицу. Обосновано использование для этого глубоких свёрточных ИНС. По результатам анализа существующих ИНС разработана новая функция ошибки. Экспериментально доказана эффективность применения новой функции ошибки для обучения ИНС, выделяющих идентифицирующие признаков. Была разработана, обучена и протестирована ИНС для идентификации людей по лицу, а также предложены меры по дальнейшему улучшению показателей точности и возможные приложения.

The methods of person identification by face have been investigated in this work. The usage of deep convolutional ANNs for this purpose was justified. Based on the analysis of the existing ANNs, a new loss function for training was developed. The effectiveness of the application of the proposed loss function for the training ANNs for identifying features extraction have been experimentally proven. The ANN for person identification was developed, trained with proposed loss function and tested. Actions for further improvement accuracy and possible applications were proposed.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Идентификация человека по изображению его лица (ИЛ) – одна из актуальных задач сразу в нескольких областях, в том числе в построении систем безопасности (средства авторизации, поиск правонарушителей) и машинном обучении (распознавание многопараметрических объектов). ИЛ позволяет производить поиск правонарушителей, потерянных людей и людей в социальных сетях, производить авторизацию в информационных системах и в системах контроля и управления доступом.

Ввиду прорывных успехов в области машинного обучения одним из самых точных способов идентификации по лицу, как и многих других задач обработки изображений, стали глубокие свёрточные искусственные нейронные сети (ГСИНС).

Идентификация человека по лицу – это один из видов биометрической идентификации. Согласно ГОСТ [1], существуют две разновидности этой задачи:

а) Верификация – это определение, что представленный пользователем образец (изображение лица) соответствует зарегистрированному шаблону.
б) Идентификация – это определение, которому из зарегистрированных шаблонов соответствует представленный пользователем образец.

Задачи идентификации и верификации определены на открытом или закрытом множествах. Множество называется закрытым, если все пользователи зарегистрированы в системе, и открытым, если некоторые могут быть не зарегистрированными. С точки зрения машинного обучения [2], при идентификации и верификации на закрытом множестве обучающий набор

данных содержит классы (персоны) из тестового множества, в то время как на открытом обучающий набор данных не содержит классов из тестового.

На закрытом множестве верификация сводится к предсказанию идентификатора класса, к которому принадлежит пользовательский образец, и сравнению его с идентификатором зарегистрированного класса, а задача идентификации сводится к классификации. Работа ИНС как подсистемы обработки сигналов сводится к предсказанию идентификатора класса, к которому принадлежит пользовательский образец.

На открытом множестве работа ИНС как подсистемы обработки сигналов заключается в извлечении характерных численных признаков из пользовательского образца. При верификации эти признаки сравниваются с признаками зарегистрированного шаблона для определения их идентичности. При идентификации производится поиск наиболее похожего зарегистрированного шаблона.

В этой работе исследуются способы разработки и обучения ИНС для извлечения из изображения лица идентифицирующих человека признаков.

Одним из ключевых факторов при обучении ИНС является определение функции ошибки. На данный момент существует множество определений функции ошибки для обучения ГСИНС, извлекающих идентифицирующие признаки из изображений лиц, но многие из этих определений обладают различными недостатками: использование при обучении массивных матриц, которые для практического применения удаляются, медленный прогресс обучения, сложные процедуры отбора обучающих примеров, чувствительность к распределению примеров в обучающей выборке.

Цель этой работы – разработка и обучение свёрточной ИНС, извлекающей из изображений лиц признаки, пригодные для решения задач верификации и идентификации лиц на открытом множестве. Для этого, выделенные признаки должны отвечать следующим условиям:

а) Вариативность признаков в пределах одного класса должна быть мала для всего множества образцов этого класса.

б) Признаки, извлекаемые из образцов двух разных классов, должны быть различимы, или, иными словами, разница между ними должна быть не меньше некоторого порогового значения.

Для выполнения этих требований, предлагается и экспериментально проверяется новое определение функции ошибки.

1 Обзор задачи идентификации по лицу и методов её решения

1.1 Идентификация по изображению лица как вид биометрической идентификации

Идентификация по лицу – один из видов биометрической идентификации. В отличие от других, она не требует непосредственного взаимодействия пользователя с сенсором (как, например, при анализе отпечатков пальцев, сетчатки или распознавания по голосу). Существует несколько разновидностей идентификации по лицу: идентификация по форме, по тепловому снимку и по изображению лица.

При идентификации по форме лица последняя может быть представлена облаком точек, полигональной поверхностью (с текстурой лица или без неё) или изображением с дополнительным каналом, кодирующим глубину. Такие системы достигают высокой точности (97% – 99%). Их основным недостатком является необходимость использования дорогостоящих систем стереосъёмки или других методов получения информации о форме лица. В работе [3] предлагается подход, при котором 3-мерная полигональная модель лица используется только при регистрации, а для верификации достаточно обычного изображения. Так же возможно использовать ИНС для реконструкции формы лица по изображению [4].

При идентификации по тепловому снимку входными данными системы является снимок лица в одном или нескольких диапазонах инфракрасного излучения: ближнем (NIR, длина волны от 0,75 до 1,4 мкм), коротковолновом (SWIR, 1,4 – 3 мкм), средневолновом (MWIR, 3 – 8 мкм) или длинноволновом (LWIR, 8 – 15 мкм). Этот подход имеет ряд преимуществ: он не требует источника освещения и не чувствителен к освещению (в случае LWIR и MWIR, в которых уровень собственного излучения лица максимален), инфракрасное излучение значительно меньше рассеивается и поглощается пылью и дымом по

сравнению с излучением видимого спектра. Инфракрасный снимок может содержать не только внешние, но и анатомические признаки лица, например рисунок сети капилляров. Кроме того, инфракрасный снимок менее чувствителен к выражению лица. К недостаткам этого метода относится необходимость использования специальных инфракрасных камер, чувствительность к температуре окружающей среды (в диапазонах MWIR и LWIR), возможное наличие на субъекте непрозрачных в инфракрасном диапазоне очков, влияние на инфракрасное излучение лица таких факторов, как эмоциональное и физическое здоровье субъекта и наличие у него в крови алкоголя [5].

Идентификация по изображению лица не требует, в отличие от остальных описанных методов, использования специальных камер, систем камер, 3D сканеров или иного специального оборудования. Благодаря этому её можно производить при обработке любых фотографий и видеозаписей вне зависимости от средств и условий их создания (неконтролируемые условия).

Существует ряд факторов, представляющих сложность при идентификации по лицу в неконтролируемых условиях. К таким факторам относятся большое разнообразие возможных положений лица на изображении как физического тела (3 угла поворота в пространстве, различное расстояние от камеры), изменчивость лица во времени (старение, разнообразие возможных выражений лица, возможное наличие различных шрамов, травм, татуировок, бород, усов и причёсок, различных аксессуаров (очки, пирсинг, средства изменения внешности - косметика, камуфляж, пластический грим)), влияние на качество изображения факторов внешней среды (уровень освещения, задний план), характеристики камер (расфокусировка, размытие при движении или вибрации, низкое разрешение снимка), артефакты сжатия фото или видео.

Системы биометрической идентификации имеют два приложения: верификация и идентификация. Если все пользователи зарегистрированы в регистрационной БД, система решает задачу идентификации на закрытом

множестве. Если предполагается, что некоторые пользователи могут быть не зарегистрированы, то система решает задачу идентификации на открытом множестве. Из этого следует, что система идентификации, предназначенная для работы на открытом множестве, так же будет работать и на закрытом.

Согласно [1], точность верификации и идентификации, оцениваются следующими показателями (помимо специфических показателей для каждой из этих задач, которые описаны ниже):

а) вероятность ложного совпадения (FMR) – доля образцов (изображений), ошибочно признанных совпадающими с шаблоном другого пользователя;
б) вероятность ложного несовпадения (FNMR) – доля образцов подлинных лиц, ошибочно признанных несовпадающими с их шаблонами.

1.1.1 Задача верификации

Верификация – это проверка, является ли пользователь источником определённого зарегистрированного в системе шаблона путём сравнения этого шаблона и пользовательского образца. Процесс верификации состоит из следующих шагов (транзакция верификации):

а) Получение пользовательского образца.
б) Выделение признаков.
в) Проверка качества и возможное отклонение непригодного образца.
г) Сравнение выделенных признаков с признаками, выделенными из зарегистрированного шаблона и определение меры их схожести.
д) Если мера схожести больше порога принятия решения, верификация успешна, в противном случае она не успешна.
е) Возвращение результата верификации.

Для оценки качества системы верификации используются следующие характеристики:

а) вероятность ложного допуска (FAR) – доля ошибочно принятых транзакций верификации неподлинного лица (самозванца);
б) вероятность ложного недопуска (FRR) – доля ошибочно отвергнутых транзакций верификации подлинного лица.

Эти характеристики зависят от порога принятия решений, вследствие чего отображаются в виде кривых РХ (рабочей характеристики, ROC) или КОО (компромиссного определения ошибки). Также важным показателем является вероятность истинно-положительных допусков при фиксированной (ограниченной) вероятности ложных допусков.

Кривая рабочей характеристики (РХ) – график параметрической функции порога принятия решения, в котором по оси X откладываются вероятности ложноположительных решений, а по оси Y – вероятности истинно-положительных решений.

Кривая компромиссного определения ошибки (КОО) – модифицированный график кривой РХ, в котором по оси X откладываются вероятности ложноположительных решений, а по оси Y - вероятности ложноотрицательных.

1.1.2 Задача идентификации

Идентификация – это поиск шаблонов в регистрационной базе данных (регистрационной БД), источником которых может быть пользователь. Результатом идентификации является список идентификаторов пользователей, которые могут соответствовать идентифицируемому субъекту - список кандидатов. Этот список может содержать 0, 1 или более идентификаторов. Ограничение длины списка кандидатов называется рангом. Идентификация

является истинно положительной, если субъект зарегистрирован и его идентификатор есть в списке кандидатов. В противном случае идентификация ошибочна. Процесс идентификации содержит следующие шаги (транзакция идентификации):

а) Получение пользовательского образца.
б) Выделение признаков.
в) Проверка качества и возможное отклонение непригодного образца.
г) Сравнение выделенных признаков с шаблонами из базы данных.
д) Выбор идентификаторов, для которых степень схожести превышает порог принятия решения.
е) Возвращение списка кандидатов.

Качество идентификации определяется следующими показателями:

а) вероятность истинно положительной идентификации ранга r (вероятность идентификации) – доля транзакций идентификации зарегистрированных пользователей, результат которых содержит истинный идентификатор в первых r возвращаемых соответствиях;
б) вероятность ложноотрицательной идентификации (ВЛОИ) – доля транзакций зарегистрированных пользователей, результат которых не содержит правильного идентификатора;
в) вероятность ложноположительной идентификации (ВЛПИ) – доля транзакций незарегистрированных пользователей с непустым результатом (определена только для идентификации на открытом множестве).

Эти характеристики зависят от порога принятия решения, размера регистрационной БД и ранга. Поэтому они отображаются в виде графика зависимости истинно-положительной идентификации ранга 1 от размера регистрационной БД при постоянном значении ВЛПИ и кривых КОО (или РХ),

отображающих зависимость ВЛПИ и ВЛОИ для различных размеров регистрационной БД и ранга.

При идентификации на закрытом множестве характеристики изображаются в виде кривой ХСС (характеристики совокупной схожести) и графика зависимости вероятности идентификации ранга 1 от размера регистрационной БД.

Кривая характеристики совокупной схожести (ХСС) – график результатов испытаний идентификации, в котором по оси X откладываются значения ранга, а по оси Y – вероятность верной идентификации при данном или меньшем ранге.

1.2 Решение задачи идентификации по изображению лица

Задача идентификации по лицу в процессе решения разбивается на несколько подзадач:

а) Детектирование лица.
б) Выравнивание.
в) Нормализация изображения.
г) Извлечение признаков.
д) Идентификация или верификация по выделенным признакам.

Детектирование лица – поиск на изображении участков, содержащих лица. Задача усложняется разнообразием возможного положения лица на изображении, разнообразием лиц, возможным перекрытием частей лица другими объектами и условиями съёмки изображения.

Выравнивание – это преобразование изображения, снижающее влияние факторов, мешающих корректной идентификации и связанных с природой лица как физического тела. Обычно включает в себя аффинные преобразования изображения, в результате которых особые точки лица (например, глаза, нос,

уголки рта) принимают определённое положение на изображении. Также может включать в себя алгоритмы обработки изображения, реконструирующие фронтальное изображение лица (например, наложение изображения как текстуры на усреднённую 3d-модель лица и визуализация фронтального изображения модели [6]). Недостаток таких алгоритмов заключается в возможном внесении искажений в распознаваемый образ.

Нормализация – преобразование изображения, в результате которого интенсивности цветов пикселей получают одинаковые статистические показатели. Заключается в вычитании из них среднего значения и делении на стандартное отклонение. Среднее и стандартное отклонение рассчитываются по изображению или по обучающей БД (в этом случае для каждого пикселя среднее и стандартное отклонение рассчитываются отдельно).

Извлечение признаков может выполняться различными алгоритмами компьютерного зрения и машинного обучения. Для обработки изображений лиц часто применяются следующие:

a) Eigenfaces (1987) – анализ главных компонент обучающей БД путём поиска собственных чисел и векторов (собственных лиц) матрицы ковариации этой БД. Среди собственных векторов выбираются наиболее значимые по суммарному значению соответствующих собственных чисел. Эти векторы нормируются и определяют ортонормированный базис такого подпространства в пространстве изображений, что его размерность значительно меньше размерности пространства изображений, а вектор в нём приближает любое лицо в обучающей БД с малой погрешностью. Координаты изображения в этом подпространстве получаются путём скалярного произведения вектора, представляющего изображение, на базисные векторы, и используются как вектор признаков. Недостатком этого метода является высокая чувствительность к помехам (например, к

наличию очков), выражению лица, изменчивости положения лица и к освещению, что делает его практически неприменимым без соблюдения идеальных условий съёмки [7].

б) Fisherfaces (1996) – метод, аналогичный Eigenfaces в том плане, что в пространстве изображений производится поиск базиса подпространства со значительно меньшей размерностью. При этом искомый базис должен удовлетворять следующему свойству: вариативность координат внутри классов минимальна, а между классами максимальна. Для оценки вариативности внутри и между классами вычисляются матрицы разброса: межклассовая S_b и внутриклассовая S_w. Искомым базисом являются столбцы матрицы V, такой, что она максимизирует дискриминант Фишера (1.1).

$$\mathrm{V} = \mathrm{argmax}_{\mathrm{V}} \frac{\det(\mathrm{V}^{\mathrm{T}} \mathrm{S_b} \mathrm{V})}{\det(\mathrm{V}^{\mathrm{T}} \mathrm{S_w} \mathrm{V})} \qquad (1.1)$$

Эти векторы вычисляются как собственные векторы матрицы $S_w^{-1} S_b$, соответствующие ненулевым собственным числам. Они называются лицами Фишера (FisherFaces) и составляют базис, оптимальный для различения классов. Этот метод более устойчив к изменениям освещённости и выражения лица (если эти вариативности отображены в обучающей БД), чем EigenFaces, но также чувствителен к изменению положения головы на изображении.

в) Искусственные нейронные сети (ИНС). ИНС – математическая модель, по принципу построения и функционирования подобная естественным нейронным сетям. ИНС представляется как функция, значение которой зависит от двух аргументов – вектора входных данных и вектора весов (параметров, весовых коэффициентов). Для поиска весов, обеспечивающих наилучшее решение требуемой задачи (в данном случае – выделение идентифицирующих признаков) производится обучение – поиск значений параметров,

минимизирующих функционал качества (целевую функцию, функцию потерь, или функцию ошибки) на обучающем наборе данных. Недостатком этого метода является сложность при подборе архитектуры сети, параметров алгоритма обучения и функции ошибки, позволяющих получить наилучший для решения задачи набор весовых коэффициентов за конечное время. Методы EigenFace и FisherFace по методу извлечения признаков можно рассматривать как частный случай ИНС – однослойный перцептрон с линейной функцией активации и со специфическими алгоритмами обучения. Существует множество видов ИНС. Наиболее высокие результаты в задачах распознавания изображений на данный момент достигаются за счёт использования глубоких свёрточных ИНС [8, 9].

При идентификации и верификации вычисляется мера схожести между признаками пользовательского образца и признаками шаблона из регистрационной БД. Примеры возможных определений мер схожести (или расстояния, при уменьшении расстояния схожесть возрастает):

а) Евклидово расстояние в пространстве признаков (1.2).

$$D(A, B) = \|B - A\|_2 \quad (1.2)$$

б) Косинусное сходство (1.3). Чем меньше угол AOB, тем более схожи векторы A и B.

$$S(A, B) = \frac{A \cdot B}{\|A\|\|B\|} \quad (1.3)$$

в) Угловое расстояние, или угол между векторами (1.4).

$$D(A, B) = \cos^{-1} \frac{A \cdot B}{\|A\|\|B\|} \quad (1.4)$$

В терминах машинного обучения, формулировки задач верификации и идентификации на открытом и на закрытом множествах отличаются следующим образом: при обучении ИНС для идентификации на открытом

множестве тестовое множество не содержит классы, содержащиеся в обучающем множестве, в то время как при обучении ИНС для идентификации на закрытом множестве перечни персон в обучающем и в тестовом множествах совпадают. Цель верификации – определить, принадлежат лица на двух изображениях одному человеку или разным [2].

1.2.1 Преимущества использования глубоких свёрточных ИНС

Использование ГСИНС для извлечения признаков на данный момент является одним из самых точных подходов ввиду множества преимуществ:

а) ИНС позволяют обнаруживать сложные зависимости и закономерности во входных данных. Например, если в алгоритмах EigenFace и FisherFace выделяемым признаком является мера соответствия изображения тому или иному шаблону, в большей или меньшей степени определяющему признаки на всём изображении сразу, то ГСИНС позволяют выделять признаки, проявляющиеся на изображении локально. Например, если сеть обучена определять цвет глаз, то она будет делать это вне зависимости от точного положения глаз на изображении.

б) Устойчивость к шумам. Если при обучении ИНС используются зашумлённые данные (например, размытые изображения или изображения с помехами и артефактами сжатия), ИНС может быть обучена выделять полезные для решения поставленной задачи признаки.

в) Адаптивность к изменению условий. Уже обученную ИНС можно обучить на новых данных или приспособить для решения новой (схожей) задачи. Например, ИНС, выделяющую идентифицирующие лицо признаки из качественных цветных изображений, можно обучить выделять такие признаки из чёрно-

белых изображений камер наружного видеонаблюдения, определять возраст человека или выражение его лица.

г) Массивный параллелизм. Особенность ИНС – возможность параллельной обработки сигнала на множестве вычислительных устройств. Особенно существенно ускорение вычислений при использовании таких вычислительных устройств, как современные видеокарты и интегральные схемы специального назначения (в частности, Google TPU).

1.2.2 Сложности создания ИНС

При разработке и обучении ИНС существует ряд трудностей, связанных с созданием обучающего и тестового наборов данных, выбором архитектуры сети, функции ошибки, алгоритма обучения и его параметров.

1.2.2.1 Создание обучающего и тестового наборов данных

Для обучения ИНС требуется выборка из генеральной совокупности возможных прецедентов – описаний возможных входных и выходных данных в виде пары объект – ответ (при обучении с учителем). Для получения таких выборок существует ряд методов:

а) Использование уже готовых наборов данных для часто встречающихся задач. Например, для идентификации по лицу существует несколько готовых наборов данных: Labeled Faces in the Wild (LFW) [10, 11], YouTube Faces (YTF) [12], CASIA-WebFace [13] и MS-Celeb-1M [14].

б) Сбор данных в глобальной сети (Web Content Mining). Производится в автоматическом режиме и предназначен для сбора данных, находящихся в открытом доступе.

в) Аугментация (наращивание) данных. Применяется в случае, когда уже есть собранные данные, но они не полностью отображают вариативность в генеральной совокупности. Заключается в дублировании данных с внесением допустимых (в генеральной совокупности) искажений. Например, если есть изображения лиц людей и предполагается, что среди них могут быть отражённые по горизонтали изображения, в выборку добавляются отражённые по горизонтали копии всех изображений.

г) Синтез данных. Применяется, когда известна точная (с точки зрения решаемой задачи) модель возникновения исходных данных, и заключается в том, что по этой модели генерируется (в автоматическом режиме) любое (заданное пользователем) количество данных для обучения ИНС. К недостаткам этого метода относятся уязвимость к возможным ошибкам и ограничениям в модели генерации данных и невозможность использования этих данных для тестирования ИНС (вследствие того, что они не представляют собой выборку из генеральной совокупности прецедентов).

д) Сбор данных вручную. Применяется в случае невозможности использования вышеперечисленных методов, например, при сборе медицинских данных, данных из бумажных источников или данных физических экспериментов, требующих высокой квалификации экспериментатора.

В методах машинного обучения с учителем предполагается использование двух выборок данных:

а) Обучающая. Используется для поиска значений параметров математической модели зависимости выходных данных от входных, удовлетворяющих оптимальному решению поставленной задачи.

б) Тестовая. Используется для оценки качества решения задачи.

Для задач верификации и идентификации по изображению лица существует несколько широко используемых наборов данных:

а) LFW – содержит 13233 изображения 5749 людей. Использование этого набора данных для обучения ограничено тем фактом, что большинство людей представлены одним единственным изображением. Обычно применяется в качестве тестового набора данных в задачах верификации и идентификации, для чего разработано несколько протоколов тестирования [10, 11, 15, 16].

б) CASIA WebFace Database [13] – БД изображений лиц, содержащая 494414 изображений 10575 человек. Предназначена для обучения ГСИНС с последующим тестированием на наборе данных LFW по стандартному протоколу верификации или по протоколу BLUFR [15].

в) MS-Celeb-1M [14] – БД фотографий людей, содержащая тестовую и обучающую выборки. Обучающая выборка содержит примерно 10 миллионов фотографий примерно 100 тысяч персон. Создана подразделением Microsoft Research в рамках соревнования по распознаванию людей по фотографии в 2016 году. По сравнению с другими открытыми выборками, авторы этой БД намеренно не производили очистку от некорректных данных (шума).

1.2.2.2 Выбор архитектуры ИНС

Архитектура ИНС отображает используемый набор математических функций, операций, значений параметров, входных и выходных данных и связей между ними. Выбор архитектуры определяет трудоёмкость обучения и ограничение на сложность обработки сигнала в ИНС.

На данный момент не существует какого-либо универсального алгоритма подбора архитектуры ИНС для решения каждой конкретной задачи, и

разработчику приходится выбирать архитектуру, опираясь на свой опыт и опыт аналогичных разработок. Задачу подбора архитектуры упрощает то, что на данный момент в разработке и оптимизации архитектур ИНС уже накоплен достаточный опыт для формулирования правил и рекомендаций [17]. Также была разработана ИНС, оптимизирующая уже существующую архитектуру другой ИНС на заданном наборе обучающих данных [18]. Подбор архитектуры может производиться автоматически, с использованием генетического алгоритма [19, 20].

Среди экспериментально зарекомендовавших себя архитектур важно отметить две, оказавшие наибольшее влияние в области обработки изображений в последние годы: Inception-v1 [9] и ResNet [21]. Архитектура Inception-v1 включает в себя «составные» слои, включающие в себя операции свёртки с различными размерами ядер, что позволяет достигнуть той же точности при значительном снижении необходимых вычислительных ресурсов. Ключевой особенностью архитектуры ResNet является использование блоков из нескольких последовательных свёрточных слоёв, признаки на выходе которых суммируются с их входными признаками. Такое решение позволило существенно повысить глубину ГСИНС, а вместе с ней – сложность обработки сигнала.

1.2.2.3 Выбор алгоритма обучения и его параметров

Обучение ИНС – процесс минимизации функции ошибки ИНС по её весам на заданных входных данных. Для обучения ГСИНС применяется алгоритм обратного распространения ошибки – итеративный алгоритм оптимизации, основанный на алгоритме градиентного спуска и включающий 3 этапа:

a) Прямое распространение – вычисление функции ошибки на входных данных и запоминание активаций нейронов;

б) Вычисление градиента функции ошибки по весам сети по алгоритму обратного распространения ошибки;
в) Применение градиентов – коррекция весов ИНС в соответствии с вычисленными градиентами и правилом изменения весов.

В зависимости от того, какие объёмы данных используются на каждой итерации алгоритма, различают:

а) Пакетный градиентный спуск – если на каждом шаге функция ошибки вычисляется для всей обучающей выборки;
б) Стохастический градиентный спуск – на каждом шаге оптимизации используется один пример из обучающей выборки;
в) Мини-пакетный (mini-batch) градиентный спуск –на каждом шаге используется небольшое подмножество обучающей выборки.

Под алгоритмом оптимизации (оптимизатором) обычно понимают правило изменения весов на шаге 3. Существует множество оптимизаторов, каждый из которых имеет свои преимущества и недостатки. Также для каждого оптимизатора может существовать несколько вариантов реализации его основной идеи:

а) Обычный градиентный спуск, описываемый формулой (1.5).

$$w_{t+1} = w_t - \eta \nabla_w \mathcal{L}(w_t, \{\langle x_i, y_i \rangle\}) \quad (1.5)$$

где w_t – веса сети, η – скорость обучения (параметр), $\mathcal{L}$ - функция ошибки, $\{\langle x_i, y_i \rangle\}$ – множество обучающих примеров.

б) Momentum: для ускорения прохождения оврагов, к вектору обновления признаков добавляется инерционная составляющая. Правило обновления весов может иметь вид, представленный формулами (1.6) и (1.7).

$$v_t = \gamma v_{t-1} + \eta \nabla_w \mathcal{L}(w, \{\langle x_i, y_i \rangle\}) \quad (1.6)$$

$$w_{t+1} = w_t - v_t \quad (1.7)$$

в) Nesterov accelerated gradient – для обновления вектора скорости изменения весов градиент функции ошибки вычисляется для значений весов, прогнозируемых на основе инерционной составляющей, реализуя стратегию «заглядывания вперёд». Например, в формулах (1.8) и (1.9) вычисление градиента происходит в точке, в которую веса будут сдвинуты при применении к ним только инерционной составляющей.

$$v_t = \gamma v_{t-1} + \eta \nabla_w \mathcal{L}(w_t - \gamma v_{t-1}, \{\langle x_i, y_i \rangle\}) \quad (1.8)$$

$$w_{t+1} = w_t - v_t \quad (1.9)$$

г) Adaptive gradient (Adagrad): для выделения редких информативных признаков, скорость обучения для весов адаптируется таким образом, что редко обновляемые веса изменяются сильнее, чем часто изменяемые. Правило обновления весов описывается формулами (1.10) - (1.12). Недостатком этого правила является возможность накопления значительных значений G_j^t, что приводит к остановке («параличу») процесса обучения.

$$g_j = \nabla_w \mathcal{L}(w_t, \{\langle x_i, y_i \rangle\})_j \quad (1.10)$$

$$G_j^t = G_j^{t-1} + g_j^2 \quad (1.11)$$

$$w_j^t = w_j^{t-1} - \frac{\eta}{\sqrt{G_j^t + \epsilon}} g_j \quad (1.12)$$

д) RMSprop – исправление правила Adagrad, предотвращающее паралич обучения при накоплении больших значений G_j^t. Вместо суммы используется экспоненциальное плавающее среднее квадратов градиентов. Правило обновления весов описывается формулами (1.13) – (1.15).

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2 \quad (1.13)$$

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon} \quad (1.14)$$

$$w_{t,j} = w_{t-1} - \frac{\eta}{RMS[g]_t} g \quad (1.15)$$

е) Adadelta – другой вариант исправления указанной проблемы правила Adagrad, использующий также усреднение скорости изменения вектора признаков. Оптимизатор описывается формулами (1.13) , (1.14), (1.16) – (1.18).

$$E[v]_t = \gamma E[v]_{t-1} + (1-\gamma) v_t^2 \quad (1.16)$$

$$v_t = -\frac{RMS[v]_{t-1}}{RMS[g]_t} g \quad (1.17)$$

$$w_t = w_{t-1} + v_t \quad (1.18)$$

ж) Adaptive moment estimation (Adam) – оптимизатор, сочетающий идеи инерции и адаптивной скорости обучения. Описывается формулами (1.19) – (1.21).

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad (1.19)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_{12}) g_t^2 \quad (1.20)$$

$$v_t = \frac{\eta}{\sqrt{\frac{v_t}{1-\beta_2^t}+\epsilon}} \frac{m_t}{1-\beta_1^t} \quad (1.21)$$

Выбор оптимальных параметров оптимизаторов играет важную роль для получения хорошо обученной ИНС. Большинство параметров имеют рекомендуемые значения:

а) $\epsilon = 10^{-8}$ – малое число, необходимое для предотвращения деления на 0.

б) $0 < \eta < 1$ – скорость обучения. В некоторых алгоритмах не имеет большого значения за счёт их адаптивности.

в) $\beta_1 = \beta_2 = 0.999$ – рекомендуемые значения параметров алгоритма Adam.

г) $\gamma = 0.9$ – скорость затухания экспоненциального среднего в разных алгоритмах.

1.2.3 Определения функции ошибки

Определение функции ошибки является одной из ключевых компонент решения задачи с использованием ИНС. Она представляет собой математическую формулировку критерия качества работы ИНС и значительно влияет на время обучения и достижимую точность решения поставленной задачи. Функции ошибки ИНС для задачи распознавания многопараметрических объектов по изображению можно разделить на два вида: обучение классификатора и метрическое обучение.

Во избежание переобучения ИНС за счёт чрезмерного роста весов, часто вводится добавочная ошибка регуляризации, пропорциональная L_1 – или L_2 – норме вектора весов ИНС.

1.2.3.1 Обучение классификаторов

При обучении ИНС как классификатора, после слоя извлечения признаков добавляется полносвязный слой, активации нейронов которого нормируются функцией Softmax и представляют собой оценки вероятности принадлежности изображения соответствующим классам (каждому классу соответствует один нейрон). Функция ошибки для обучения таких классификаторов в литература часто называется Softmax loss и определяется как перекрёстная энтропия предсказанных и истинных вероятностей (1.22).

$$\mathcal{L}_{\mathrm{Softmax}} = -\sum_{\mathrm{i=1}}^{\mathrm{m}} \log \frac{e^{A_{y_i} \cdot f_i + b_{y_i}}}{\sum_{j=1}^{n} e^{A_j \cdot f_i + b_j}} \qquad (1.22)$$

где $f_k = F(x_k)$ – выделенные из изображений x_k признаки, A – матрица весов классифицирующего слоя, b – вектор смещений классифицирующего слоя, y_i – индекс истинного класса изображения x_i, m – количество обучающих примеров. Перекрёстная энтропия может усредняться по обучающим примерам, в этом случае функция ошибки соответствует формуле (1.23).

$$\mathcal{L}_{\text{softmax}} = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{e^{A_{y_i}\cdot f_i + b_{y_i}}}{\sum_{j=1}^{n} e^{A_j \cdot f_i + b_j}} \quad (1.23)$$

Если ИНС необходимо использовать для извлечения признаков, последний (классифицирующий) слой удаляется. Если решается задача идентификации или верификации на закрытом множестве, то классифицирующий слой может быть оставлен. В этом случае он играет роль регистрационной БД, возвращая для изображения на входе ИНС вероятности принадлежности каждому классу.

Преимуществом этого подхода к обучению является относительно высокая скорость обучения. Недостатками являются:

а) Неэффективное использование памяти. При обучении на большом количестве классов (порядка 10000 и более) матрица A занимает значительный объём памяти.

б) Чувствительность к сбалансированности обучающих данных. Если в обучающей выборке имеется небольшое количество классов, количество изображений для которых значительно больше среднего количества изображений на класс, то итоговая обобщающая способность ИНС сильно снижается [22].

в) Метод не гарантирует, что извлекаемые признаки f_i будут достаточно эффективны для решения задач идентификации или верификации на множестве персон, не встречавшихся в обучающей выборке, то есть на открытом множестве [23].

Для борьбы с этими недостатками разработано множество модификаций этой функции ошибки:

а) DeepID2 (2014). Предлагается обновлять веса ИНС путём комбинирования градиентов двух функций ошибки: ошибки верификации (1.24) и идентификации (1.25).

$$\mathrm{Verif}(f_i, f_j, y_{ij}) = \frac{1}{2}(y_{ij} - \sigma(w\frac{f_i f_j}{\|f_i\|_2\|f_j\|_2} + b))^2 \quad (1.24)$$

$$\mathrm{Ident}(f, t) = -\log \hat{p}_t = -\log\frac{e^{A_{t_i}\cdot f_i + b_{t_i}}}{\sum_{j=1}^{n} e^{A_j\cdot f_i + b_j}} \quad (1.25)$$

где f_i, f_j, f – извлечённые из изображений признаки, $y_{ij} = 1$, если f_i и f_j извлечены из изображений одного класса и –1 – если из разных, t – идентификатор класса изображения, из которого извлечены признаки f. Достигнутая точность верификации по протоколу LFW составляет 99.15% [24].

б) Center Loss (2016). К функции ошибки Softmax loss добавляется оценка разброса признаков от центров тяжести соответствующих классов (1.26).

$$\mathcal{L} = \mathcal{L}_s + \lambda\frac{1}{2}\sum_{i=1}^{m}\|x_i - c_{y_i}\|_2^2 \quad (1.26)$$

Достигнутая точность на LFW составляет 99.28% [25].

в) Range Loss (2016). Если обучающая выборка содержит классы со значительно большим количеством изображений, чем в остальных, качество извлекаемых ИНС признаков снижается. В этом случае предлагается добавить (с весовыми коэффициентами) к функции ошибки Softmax loss ошибку внутриклассовой вариативности (1.27) и ошибку межклассовой вариативности (1.28).

$$\mathcal{L}_{R_{intra}} = \sum_{i\subseteq I}\frac{k}{\sum_{j=1}^{k}\frac{1}{\mathcal{D}_{ij}}} \quad (1.27)$$

$$\mathcal{L}_{R_{inter}} = \max(m - \min_{a,b\in I}\|\bar{x}_a - \bar{x}_b\|_2^2, 0) \quad (1.28)$$

где $\mathcal{D}_{ij}$ – j-е по убыванию максимальное расстояние внутри класса i. Достигнутый результат на LFW 99.52% [22].

г) Large-Margin Softmax Loss (2016). Предлагается новое определение функции ошибки Softmax Loss, описываемое формулами (1.29), (1.30).

$$\mathcal{L}_s = -\frac{1}{N}\sum_i \log\left(\frac{e^{\|A_{y_i}\|\|x_i\|\psi(\theta_{y_i})}}{e^{\|A_{y_i}\|\|x_i\|\psi(\theta_{y_i})}+\sum_{j\neq y_i} e^{\|A_j\|\|x_i\|\psi(\theta_j)}}\right) \quad (1.29)$$

$$\psi(\theta) = (-1)^k \cos(m\theta) - 2k, \theta \in \left[\frac{k\pi}{m}, \frac{(k+1)\pi}{m}\right] \quad (1.30)$$

где $k \in [0, m-1]$. Такое определение позволяет ввести большую границу между классами и явно представить строки матрицы A как шаблонные признаки того или иного класса. Достигнутый результат на LFW 98.71% [26].

д) Center Invariant Loss (2017). К функции ошибки Softmax loss предлагается добавить функцию ошибки (1.31), минимизация которой приводит к тому, что центры тяжести классов равноудалены от центра:

$$\mathcal{L} = \mathcal{L}_{Softmax} + \gamma\frac{1}{4}\left(\|c_{y_i}\|_2^2 - \frac{1}{m}\sum_{k=1}^m \|c_k\|_2^2\right)^2 \quad (1.31)$$

Для снижения вариативности внутри классов, также добавляется функция ошибки Center loss. Полная функция ошибки имеет вид, представленный формулой (1.32).

$$\mathcal{L} = \mathcal{L}_{Softmax} + \lambda\frac{1}{2}\sum_{i=1}^m \|x_i - c_{y_i}\|_2^2 + \gamma\frac{1}{4}\left(\|c_{y_i}\|_2^2 - \frac{1}{m}\sum_{k=1}^m \|c_k\|_2^2\right)^2 \quad (1.32)$$

Этот подход позволил достигнуть точности на LFW 99.12% [27].

е) Marginal Loss (2017). Для улучшения различительной способности обучаемой сети предлагается к функции ошибки Softmax loss добавить (с весовым коэффициентом) такую функцию ошибки, что при её минимизации будет уменьшаться вариативность внутри классов и увеличиваться расстояние между классами за счёт воздействия на граничные примеры. Эта функция ошибки представлена равенством (1.33).

$$L_m = \frac{1}{m^2-m}\sum_{i,j,i\neq j}^m \max\left(\xi - y_{ij}\left(\theta - \left\|\frac{f_i}{\|f_i\|} - \frac{f_j}{\|f_j\|}\right\|_2^2\right), 0\right) \quad (1.33)$$

где m – количество обучающих примеров, $y_{ij} = 1$, если i и j принадлежат одному классу, и -1 - если разным. Достигнутая точность верификации на выборке LFW составляет 99.48% [28].

ж) L_2-constrained Softmax Loss (2017). Показана эффективность нормализации векторов признаков по формуле (1.34).

$$\widetilde{f_i} = \frac{\alpha f_i}{\|f_i\|_2} \tag{1.34}$$

где p – требуемая точность, C – количество классов, α – параметр, удовлетворяющий неравенству (1.35). Полученная ИНС показала точность верификации на наборе данных LFW 99.78% [23].

$$\alpha \geq \log\frac{p(C-2)}{1-p} \tag{1.35}$$

з) NormFace (2017). Предлагается применять нормализацию как к векторам признаков, так и к векторам шаблонов классов, а также не использовать вектор смещения b. Полученная функция ошибки представлена равенством (1.36).

$$\mathcal{L}_{S`} = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{e^{s\tilde{A}_{y_i}\cdot\tilde{f}_i}}{\sum_{j=1}^{n}e^{s\tilde{A}_j\cdot\tilde{f}_i}} \tag{1.36}$$

где $\widetilde{f_i} = \frac{f_i}{\|f_i\|_2}$, $\widetilde{A_j} = \frac{A_j}{\|A_j\|_2}$, $s > 0$ – заданный или обучаемый параметр, аналогичный параметру α (1.35). Достигнутая точность на выборке LFW составляет 99.22% [29].

и) SphereFace (2017). Для усиления различительной способности ИНС предлагается добавить границу вокруг центров классов, попадание в пределы которой является определяющим при принятии решения о классификации образа. Полученная функция ошибки (1.37) позволила достичь точность на выборке LFW 99.42% [2].

$$\mathcal{L}_{ang} = -\frac{1}{N}\sum_{i}\log\frac{e^{\|x_i\|\psi(\theta_{y_i,i})}}{e^{\|x_i\|\psi(\theta_{y_i,i})}+\sum_{j\neq y_i}e^{\|x_i\|\cos(\theta_{j,i})}} \tag{1.37}$$

где $\psi(\theta_{y_i,i}) = (-1)^k \cos(m\theta_{y_i,i}) - 2k, \quad \theta_{y_i,i} \in \left[\frac{k\pi}{m}, \frac{(k+1)\pi}{m}\right]$ – угол между векторами признаков i-го примера и шаблона его истинного класса, k – параметр.

к) Contrastive Center Loss (2017) – модификация (1.38) функции ошибки Center loss, также является добавочной к функции ошибки Softmax loss. Недостатком Center loss является то, что она позволяет добиться малых расстояний между векторами признаков в пределах одного класса, но не обеспечивает хорошей различимости между классами.

$$L_{ct-c} = \frac{1}{2} \sum_{i=1}^{m} \frac{\|f_i - c_{y_i}\|_2^2}{\sum_{j=1, j\neq y_i}^{k} \|f_i - c_j\|_2^2 + \delta} \tag{1.38}$$

Достигнутая точность верификации на выборке LFW составляет 98.68% [30].

л) COCO (2017). Предлагается максимально снизить вариативность векторов признаков в пределах класса и увеличить расстояния между классами. Для этого вводится функция ошибки, представленная равенством (1.39)

$$\mathcal{L}_{COCO} = -\sum_{i=1}^{m} \log \frac{\exp(\hat{c}_{y_i}^T \cdot \hat{f}_i)}{\sum_{j=1}^{m} \exp(\hat{c}_j^T \cdot \hat{f}_i)} \tag{1.39}$$

где $\hat{f}_i = \frac{\alpha f_i}{\|f_i\|_2}, \hat{c}_k = \frac{c_k}{\|c_k\|_2}$. В результате была достигнута точность 99.86% на выборке LFW [31].

м) Additive Margin Softmax (2018) – Модификация функции ошибки Large Margin Softmax Loss с более наглядным (с геометрической точки зрения) способом введения отступа m между классами. Функция ошибки представлена формулой (1.40).

$$\mathcal{L}_{AMS} = -\frac{1}{n} \sum_{i=1}^{n} \log \frac{\exp(s \cdot (\cos \theta_{y_i} - m))}{\exp(s \cdot (\cos \theta_{y_i} - m)) + \sum_{j=1, j\neq y_i}^{c} \exp(s \cdot \cos \theta_j)} \tag{1.40}$$

Авторы достигли результата в 99.12% точности верификации на выборке LFW и 97.96% по протоколу BLUFR [32].

н) Centralized Coordinate Learning (2018). Для того, чтобы извлекаемые признаки имели наибольшую различительную способность, вектор признаков преобразуется по формуле (1.41) после чего чтобы математическое ожидание и стандартное отклонение каждого признака равны 0 и 1 соответственно.

$$\Phi(\mathrm{f})_{\mathrm{k}} = \frac{\mathrm{f_k} - \mu_{\mathrm{k}}}{\sigma_{\mathrm{k}}} \tag{1.41}$$

где μ_k – математическое ожидание k – ого признака a, σ_k – стандартное отклонение. Значения μ_k и σ_k вычисляются как экспоненциальные скользящие средние. Для большей компактности распределений векторов признаков в пределах классов, функция ошибки $\mathcal{L}_{Softmax}$ (1.42) комбинируется с другой функцией ошибки $\mathcal{L}_{AAM}$ (1.43) по формуле (1.44).

$$\mathcal{L}_{\mathrm{Softmax}} = -\frac{1}{\mathrm{m}} \sum_{\mathrm{i=1}}^{\mathrm{m}} \log \frac{\exp(\|\Phi(\mathrm{f_i})\| \cos(\theta_{\mathrm{y_i,i}}))}{\sum_{\mathrm{j=1}}^{\mathrm{n}} \exp(\|\Phi(\mathrm{f_i})\| \cos(\theta_{\mathrm{j,i}}))} \tag{1.42}$$

$$\mathcal{L}_{\mathrm{AAM}} = -\sum_{\mathrm{i=1}}^{\mathrm{m}} \frac{\exp(\|\Phi(\mathrm{f_i})\| \cos(\eta\theta_{\mathrm{y_i,i}}))}{\exp(\|\Phi(\mathrm{f_i})\| \cos(\eta\theta_{\mathrm{y_i,i}})) + \sum_{\mathrm{j \neq y_i}}^{\mathrm{n}} \exp(\|\Phi(\mathrm{f_i})\| \cos(\theta_{\mathrm{k,i}}))} \tag{1.43}$$

$$\mathcal{L} = \frac{\lambda\mathcal{L}_{\mathrm{Softmax}} + \mathcal{L}_{\mathrm{AAM}}}{\lambda + 1} \tag{1.44}$$

где $\theta_{j,i}$ – угол между векторами $\Phi(f_i)$ и $\frac{A_j}{\|A_j\|}$. Точность верификации на наборе данных LFW составила 99.58% (99.47% без добавления $\mathcal{L}_{AAM}$), и 96.43% по протоколу тестирования SLLFW [33].

о) ArcFace (2018). Предлагается новый способ введения границы между классами (1.45), при котором граница принятия решения о принадлежности объекта классу является окружностью фиксированного радиуса на поверхности гиперсферы единичного радиуса в пространстве признаков.

$$\mathcal{L}_{\text{Arc}} = -\frac{1}{n}\sum_{i=1}^{n}\log\frac{\exp(s\cdot\cos(\theta_{y_i}+m))}{\exp(s\cdot\cos(\theta_{y_i}+m))+\sum_{j=1,j\neq y_i}^{c}\exp(s\cdot\cos(\theta_j))} \quad (1.45)$$

Авторами была достигнута точность верификации на выборке LFW 99.71% [34].

Из всех перечисленных модификацией функции ошибки Softmax loss можно выделить две основные цели, которых придерживались авторы:

а) Снижение вариативности выделенных признаков в пределах класса,
б) Увеличение расстояний между классами.

1.2.3.2 Метрическое обучение

Метрическое обучение – обучение ИНС отображать объекты на входе в такое пространство признаков, что расстояние в нём находится в соответствии с мерой различия этих объектов. Применительно к задаче идентификации и верификации по лицу это означает, что разница между признаками, извлечёнными из разных изображений лица одного человека, должна быть меньше, а между признаками изображений лиц разных людей – больше некоторого порогового значения. Методы метрического обучения могут комбинироваться с обучением ИНС как классификатора разными способами, например, использоваться после основного обучения ИНС как классификатора для улучшения различительной способности ИНС [34], или они использоваться одновременно [24].

Преимуществом метрического обучения является большая различающая способность получаемой сети.

Примерами таких функций ошибки являются:

а) Contrastive loss. Для обучения ИНС используются пары обучающих примеров с пометкой, относятся они к одному классу или к разным. Функция ошибки описывается формулой (1.46).

$$\mathcal{L}_{\text{contrastive}} = \frac{1}{2}\sum_{i=1}^{P} l_i \quad (1.46)$$

где $l_i = \|f_{i_1} - f_{i_2}\|_2^2$, если пара содержит примеры из одного класса и $l_i = \max(0, m - \|f_{i1} - f_{f2}\|_2)^2$, если из разных [35]. С использованием этой функции ошибки была достигнута точность верификации на наборе данных LFW 99.47% [36].

б) Triplet loss (2015). Функция ошибки (1.47), предложенная в [37], показала высокие результаты при обучении ИНС в задаче верификации (99.63%) [38]. ИНС обучается на тройках (триплетах) примеров, содержащих 2 изображения одного человека – якорь (anchor) и позитивный пример, и одно – другого (отрицательный пример).

$$\mathcal{L}_{\text{triplet}} = \sum_{i=1}^{N} \max(0, \|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha) \quad (1.47)$$

где α – параметр. Обучение триплетами также может использоваться для улучшения ИНС, уже обученных как классификаторы [34]. Главный недостаток этого метода – сложная процедура отбора обучающих примеров (триплетов).

Важным недостатком описанных функций метрического обучения ошибки является медленная скорость обучения ИНС.

1.2.4 Протоколы тестирования ИНС

Существует множество протоколов тестирования (benchmark) ИНС для задач идентификации и верификации. Протоколы основаны на конкретных тестовых наборах данных и представляют собой описания процедуры вычисления показателей качества работы ИНС на этих данных и условия их (протоколов) применимости. Предназначение протоколов тестирования – обеспечить (1) возможность сравнения качества работы разных алгоритмов и

(2) возможность проверки заявленных достижений предлагаемых технических решений независимыми экспериментаторами.

На данный момент существует множество протоколов тестирования ИНС для задач верификации и идентификации по изображению лица (как на открытом, так и на закрытом множествах). Основные из них:

а) Протокол верификации на наборе данных LFW. В зависимости от того, какие данные и как были использованы при обучении ИНС, для корректности сравнения различных алгоритмов определено несколько протоколов тестирования. В частности, в случае использования сторонних размеченных данных, используется протокол «Unrestricted with labeled outside data». Протокол тестирования содержит 6000 пар изображений, 3000 из которых являются совпадениями, а другие 3000 – несовпадениями. При вычислении точности и кривой PX предполагается использование перекрёстной проверки с 10-ю разбиениями [11].

б) Протокол верификации на наборе данных YTF устроен также, как и протокол LFW, и предполагает перекрёстную проверку на 10 разбиениях. Он содержит 5000 пар видеороликов, разделённых на 10 частей, по 250 совпадений и несовпадений в каждой [12].

в) Протокол BLUFR определён на наборе данных LFW. В отличии от стандартного протокола LFW, он использует все 13233 изображения и содержит сценарии проверки как для верификации, так и для идентификации на открытом множестве, что позволяет лучше оценивать точность при фиксированных малых значениях FAR. Включает 10 испытаний, каждое из которых содержит около 157 тысяч соответствий и около 47миллионов несоответствий [15].

г) SLLFW – модификация стандартного протокола верификации LFW с тем отличием, что примеры несовпадений заменены на пары похожих лиц [39, 40].

д) Протокол тестирования идентификации на открытом множестве, основанный на наборе данных LFW и предложенный в [16]. Предлагается разделить множество персон на 3 категории в зависимости от их наличия в регистрационной БД и участия в обучении ИНС: известные (зарегистрированы и участвовали в обучении), известные неизвестные (не зарегистрированы, но участвовали в обучении) и неизвестные (не зарегистрированы и не участвовали в обучении).

Указанные протоколы позволяют вычислить следующие показатели:

а) Показатели верификации:
   1) Кривая рабочей характеристики;
   2) Точность;
   3) (Дополнительно) Вероятность истинно-положительного допуска при фиксированной вероятности ложного допуска.

б) Показатели идентификации:
   1) DIR (Detection and Identification Rate) – кривая PX, отображающая зависимость вероятности истинной идентификации (по оси Y) и ложноположительной идентификации (FAR) (по оси X) от порога принятия решения при заданном ранге;
   2) Кривая характеристики совокупной схожести (для идентификации на закрытом множестве).

1.3 Выводы

Идентификация людей по изображению (в диапазоне видимого света) лица имеет ряд важных с точки зрения практического применения преимуществ по сравнению как с другими методами биометрической идентификации, так и с другими методами идентификации по лицу.

Одним из наиболее точных и надёжных методов извлечения пригодных для идентификации человека признаков из изображения его лица на данный момент являются глубокие свёрточные ИНС.

При разработке и обучении ИНС необходимо определить множество аспектов, включая архитектуру ИНС, обучающие и тестовые наборы данных, процедуру тестирования и показатели точности, алгоритм обучения и функцию ошибки.

Правильный выбор функции ошибки является одним из ключевых факторов, определяющих качество обученной ИНС. Анализ функций ошибки, используемых для обучения идентифицирующих человека по лицу ИНС, показал, что многие из них имеют серьёзные недостатки. Вследствие этого было решено разработать и проверить новую функцию ошибки.

В следующих главах изложены результаты разработки идентифицирующей людей по лицу ИНС, функции ошибки, и результаты обучения этой ИНС с новой функцией ошибки.

2 Разработка ИНС для идентификации по лицу

Разработка ГСИНС для выделения из изображений лиц пригодных для решения задач верификации и идентификации признаков включает ряд подзадач:

а) Создание обучающего и тестового наборов данных,
б) Определение входных и выходных данных ИНС,
в) Выбор архитектуры ГСИНС,
г) Выбор функции ошибки ИНС,
д) Определение протоколов тестирования ИНС,
е) Выбор средств программной реализации;
ж) Программная реализация процессов создания, обучения и сохранения ИНС для дальнейшего использования.

2.1 Выбор средств программной реализации ИНС

2.1.1 Выбор языка программирования

В качестве основного языка реализации программы для создания и обучения ИНС используется язык Python 3. Этот язык был выбран, так как обладает следующими преимуществами:

а) Python 3 - язык высокого уровня, содержащий функции обработки ошибок и обширную стандартную библиотеку.
б) Python 3 имеет простой синтаксис, что позволяет производить разработку и отладку с большой скоростью.
в) Существует множество библиотек, в том числе для обработки изображений (Pillow, OpenCV), машинного обучения (TensorFlow, Torch, Keras) и научных вычислений (NumPy, SciPy), реализованных на низкоуровневых языках (в частности, на C и C++)

и имеющих интерфейс высокого уровня для языка Python 3, что делает их использование простым и эффективным.

2.1.2 Выбор библиотеки машинного обучения

На данный момент существует множество библиотек, позволяющих создавать, обучать, сохранять и использовать ИНС, в том числе ГСИНС (а многие – также и рекуррентные ИНС). Основные из них:

а) Caffe/Caffe2,
б) TensorFlow,
в) Torch,
г) Keras,
д) Theano,
е) Scikit-Learn.

Для разработки ИНС была выбрана библиотека TensorFlow, так как она следующими преимуществами:

а) Библиотека TensorFlow имеет интерфейсы для многих широко используемых языков программирования, в том числе Python 2 и 3 (самый проработанный интерфейс), C++, Haskell, Java, Go, Rust и Javascript, что позволяет после обучения использовать обученную ИНС из этих языков.
б) Наличие утилиты TensorBoard, позволяющей визуализировать граф вычислений, строить графики и гистограммы любых величин, в том числе статистики процесса обучения, просматривать данные на входе или на выходе ИНС и т. д.
в) Богатая документация библиотеки.
г) Интенсивное развитие библиотеки.
д) В библиотеке реализована функция автоматического дифференцирования и большинство операций с тензорами

(многомерными массивами) являются дифференцируемыми, что убирает необходимость программисту самостоятельно реализовывать алгоритм обратного распространения ошибки.

е) Библиотека имеет встроенные функции сохранения и загрузки графа вычислений и весов ИНС из файлов.

ж) Библиотека содержит многие алгоритмы оптимизации, применяемые при обучении ИНС, а также позволяет определять свои.

2.2 Создание обучающего и тестового наборов данных

Для обучения и тестирования работы ИНС использованы наборы данных CASIA-WebFace и Labelled Faces in the Wild (LFW) соответственно.

На вход подсистемы выделения признаков изображения подаются после выравнивания и нормализации. Поэтому для обучения и тестирования ИНС, наборы данных должны быть выровнены. Это означает, что на всех изображениях лица должны быть расположены вертикально и иметь примерно одинаковый размер. Для этого была применена процедура фильтрации, состоящая из следующих шагов:

а) Детектирование лиц. Несмотря на то, что на изображениях в обоих наборах данных лицо находится в центре изображения, для детектирования ориентировочных точек лица необходимо знать точное положение и размеры рамки, включающей в себя лицо. Средние размеры такой рамки для обоих наборов данных отличаются: при размере изображений в обоих наборах данных 250×250 пикселей, в LFW средний размер рамки составляет 100×100 пикселей, а в CASIA-WebFace – 133×133 пикселя. В процессе фильтрации, для детектирования лиц использовался детектор лиц из библиотеки dlib. Если этот детектор не смог найти

лицо, использовался детектор из библиотеки OpenCV. В случае если этот детектор так же не смог найти лицо, в качестве рамки использованы средние размер и положение рамки по набору данных.

б) Детектирование ориентировочных точек (landmarks). Для детектирования ориентировочных точек (уголков глаз, рта, носа, точек линии челюсти) используется детектор из библиотеки dlib. Он позволяет локализовать 68 ориентировочных точек лица.

в) Вертикализация. Для вертикализации производятся два аффинных преобразования – вращение и перенос изображения. Параметры этих преобразований рассчитаны таким образом, чтобы ориентировочные точки с индексами 28 (переносица) и 9 (низ подбородка) имели координаты [0.5w, 0.3h] и [0.5w, 0.9h] соответственно (первая координата – ширина; w, h – соответственно ширина и высота результирующего изображения; начало координат – в левом верхнем углу изображения). В качестве результирующего размера изображений выбрано значение в 120×120 пикселей.

Процедура нормализации изображений производится непосредственно перед подачей изображения на вход ИНС. При нормализации, производится вычитание среднего и деление на стандартное отклонение значений в тензоре изображения.

2.3 Входные и выходные данные

На вход ИНС подаётся массив вещественных чисел с формой [m, h, w, d], где m – количество подаваемых ИНС изображений за 1 раз, h – высота изображений, w – ширина изображений, d=3 – количество цветовых каналов. Перед подачей на вход ИНС, изображения проходят обработку, включающую следующие этапы:

а) Аугментация – изображение с вероятностью 0.5 отражается по горизонтали;
б) Нормализация – линейное преобразование (описанное в п. 2.2), в результате которого среднее значение интенсивности пикселей становится равным 0, а стандартное отклонение – 1.

На выходе ГСИНС возвращает векторы признаков – массив вещественных чисел формы [m, f], где f=128 – размерность вектора признаков.

2.4 Архитектура ИНС

Для ИНС была выбрана архитектура, основанная на архитектуре NN2, описанной в [38], которая, в свою очередь, основана на архитектуре Inception-v1 [9]. Причиной выбора этой архитектуры является то, что она может быть обучена извлекать идентифицирующие признаки и показывать высокую точность верификации [38]. Эта архитектура имеет следующие особенности:

а) Использование операций свёртки. Операция свёртки – вычисление скалярных произведений входного массива (на первом слое – изображения) и ядер свёртки (массивов малого размера, также называемых фильтрами) при различных положениях последних (они передвигается входному массиву методом плавающего окна). Идея работы операции свёртки основана на принципе работы зрительной коры млекопитающих, а именно – на наличии в ней иерархии нервных клеток, нейроны каждого следующего слоя которой реагирует на определённую комбинацию активаций нейронов предыдущих слоёв, являясь таким образом детектором всё более сложных образов при движении на верх иерархии. Операция свёртки предлагается как замена использованию полносвязных слоёв и имеет следующие преимущества: (1) меньшее количество параметров и повторное их использование, (2)

локальное проявление извлекаемых признаков. Использование операции свёртки показало на практике свою эффективность [8, 9].

б) Использование модулей Inception [9]. Важнейшим преимуществом архитектур, основанных на архитектуре Inception, является использование Inception – модулей (рисунок 2.1), состоящих из нескольких параллельных «ветвей» выделения признаков.

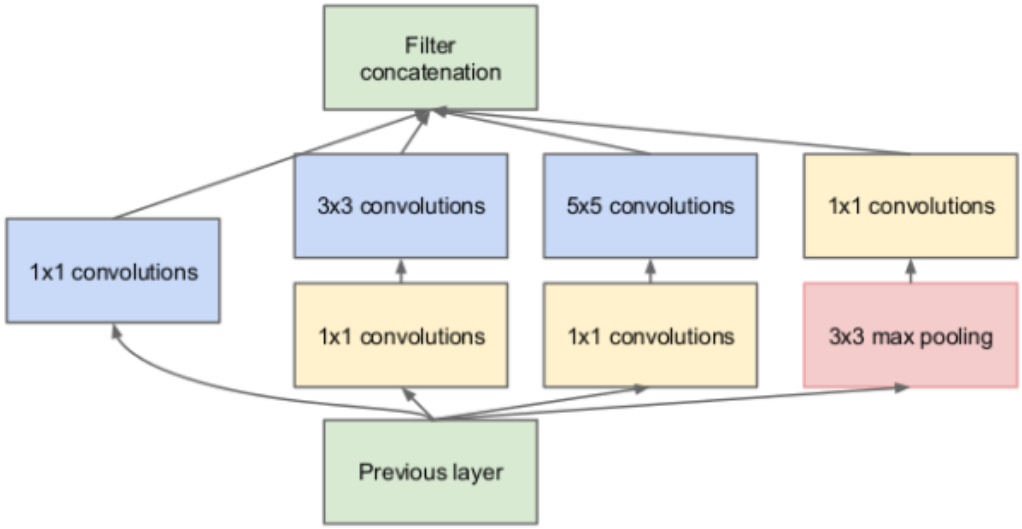

Рисунок 2.1 – Устройство Inception – модуля.

Каждая ветвь содержит слой снижения размерности (сжатие) – операцию свёртки с размером ядра 1×1 и количеством ядер (глубиной выхода) меньшим, чем на входе. Вторая и третья ветви содержат (после операции снижения размерности) операции свёртки с размерами ядер 3×3 и 5×5 соответственно. Четвёртая ветвь содержит операцию субдискретизации (подвыборки, pooling), выполняемую перед операцией снижения размерности. Такая организация архитектуры позволяет: (1) более оптимально использовать вычислительные ресурсы, (2) снизить количество параметров ИНС и количество передаваемых между слоями ИНС данных.

в) Использование операции L2-pooling. В более классических архитектурах ГСИНС (например, в ИНС AlexNet) после свёрточного слоя обычно расположен слой субдискретизации. Принцип работы такого слоя – вычисление максимума (реже – среднего, минимума или медианы) во входном тензоре в регионах заданного размера и с заданным шагом. Представляет собой средство постепенного снижения размерности выделяемых признаков в ущерб информации об их местоположении на изображении, снижения объёма вычислений и контроля переобучения ИНС. Операция L2-pooling заключается в вычислении в каждом окне L2 – нормы тензора.

Выбранная архитектура описана в таблице 2.1.

Таблица 2.1 – Описание архитектуры ГСИНС.

Тип	Шаг	#1×1	#3×3 сжатие	#3×3	#5×5 сжатие	#5×5	Тип операции pooling	#pool сжатие	Размер выхода	Количество параметров
Conv (7*7)	2								[60,60,64]	9K
Pooling	2						Max		[30,30,64]	
Inception	1		64	192					[30,30,192]	115K
Pooling	2						Max		[15,15,192]	
Inception	1	64	96	128	16	32	Max	32	[15,15,256]	164K
Inception	1	64	96		32	64	L2	64	[15,15,320]	228K
Inception	2		128	128	32	64	Max		[8,8,640]	398K
Inception	1	256	96	256	32	64	L2	128	[8,8,640]	545K
Inception	1	224	112	224	32	64	L2	128	[8,8,640]	595K
Inception	1	192	128	256	32	64	L2	128	[8,8,640]	654K
Inception	1	160	144	288	32	64	L2	128	[8,8,640]	722K
Inception	2		160	256	64	128	Max		[4,4,1024]	717K
Inception	1	384	192	384	48	128	L2	128	[4,4,1024]	1.6M
Inception	1	384	192	384	48	128	Max	128	[4,4,1024]	1.6M
Average									[1,1,1024]	
FC									[128]	131K
Всего										7.5M

Пояснения к таблице:

а) Conv (7×7) – слой с операцией свёртки и размером ядра 7×7.

б) Pooling – слой операции подвыборки. Размер окна для всех слоёв операций подвыборки составляет 3×3, конкретный тип функции подвыборки указан в столбце «Тип операции pooling».

в) Inception – обозначение Inception – модуля. Количества ядер в слоях снижения размерности указаны в столбцах «#3×3 сжатие», «#5×5

сжатие» и «#pool сжатие» для второй, третьей и четвёртой ветвей соответственно. Количество ядер в свёрточных слоях в первой, второй и третьей ветвях указано в столбцах «#1×1», «#3×3», «#5×5» соответственно.

г) Average – операция взятия среднего арифметического по первому и второму измерениям изображения.

д) FC (Fully Connected) – полносвязный слой ИНС с линейной функцией активации.

2.5 Функция ошибки

ГСИНС определяет отображение из пространства изображений в пространство признаков. Назовём координатами изображения в пространстве признаков (признаковом пространстве) координаты, в которые ИНС отображает это изображение. Координаты в пространстве признаков далее используются для решения различных прикладных задач: идентификации на закрытом множестве (классификации), идентификации на открытом множестве, кластеризации (группировки изображений по неизвестной заранее классовой принадлежности) и верификации (определения, что два изображения относятся к одному классу). Для корректного решения этих задач необходимо, чтобы отображение, определяемое ГСИНС, отвечало следующему критерию: расстояния от координат изображений до центров их классов должны быть малы относительно расстояний между центрами разных классов.

Определение: Центр класса – математическое ожидание координат изображений этого класса в пространстве признаков.

Назовём этот критерий критерием различимости классов. Для улучшения обобщающей способности ИНС, по сравнению с простейшим обучением классификатора с функцией ошибки Softmax, функцию ошибки формулируют таким образом, чтобы она учитывала этот критерий. Функции ошибки обучения

классификаторов вводят дополнительную ошибку, минимизация которой приводит к уменьшению расстояний внутри классов [25], или к увеличению расстояний между классами [26, 32], или к тому и другому одновременно [22, 24, 28, 31]. Функции ошибки метрического обучения, в свою очередь, являются математической формулировкой этого критерия и оперируют оценками (по отдельным обучающим примерам) расстояний между классами и в пределах классов.

2.5.1 Определение требований

Функции ошибки, перечисленные в п. 1.2.3, обладают рядом недостатков. Вследствие этого, желательно иметь функцию ошибки, обладающую следующими свойствами:

а) Не является функцией ошибки обучения классификатора;
б) Не требует сложной процедуры предварительного отбора обучающих примеров;
в) Оптимизирует как расстояния до центров классов, так и расстояния между центрами классов;
г) Чем сложнее обучающий пример, тем значительнее его воздействие на процесс обучения. Обучающий пример считается сложным, если вносит большой вклад в значение функции ошибки. В противном случае пример называется простым и должен оказывать слабое (или вообще не оказывать) влияние на процесс обучения.

Исходя из этих требований, можно сделать следующие выводы о возможном устройстве такой функции ошибки:

а) Указанная функция ошибки является функцией ошибки метрического обучения. Её минимизация приводит к улучшению выполнения критерия различимости классов, что означает, что функция ошибки должна зависеть не от обучающих примеров

напрямую, а от численных оценок этого критерия, зависящих от обучающих примеров. Например, в рассмотренных в пп. 1.2.3.2 функциях ошибки этими оценками являются расстояния в пространстве признаков между обучающими примерами одинаковых или разных классов. Критерий различимости классов оперирует расстояниями от центра класса до координат примеров этого класса и расстояниями между центрами классов.

б) Функция не требует сложной процедуры отбора примеров для обучения. «Сложная» здесь означает, что для формирования пакета обучающих примеров используется дополнительная информация, помимо факта принадлежности примеров одному классу. Например, в процедуре формирования триплетов обучающих примеров для функции Triplet loss используются признаки, извлечённые из изображений перед обучением [38].

в) Желательно иметь параметры, позволяющие разделять простые и сложные примеры, а также определять разницу в силе их влияния на процесс обучения. Например, методы метрического обучения, описанные в пп. 1.2.3.2, при отсеивании сложных примеров оперируют минимальным отступом между примерами разных классов.

2.5.2 Формулировка функции ошибки

Обозначим через F отображение из пространства изображений в пространство признаков, определяемое ГСИНС. Тогда координаты изображений I_{ij} определяются по формуле (2.1).

$$x_{ij} = F(I_{ij}), i = 1..p, j = 1..k_i \quad (2.1)$$

где p – количество классов изображений, подаваемых на вход ИНС, k_i – количество изображений i-го класса.

Координаты центра класса определяются по формуле (2.2) как математическое ожидание случайной величины – координат изображений, принадлежащих классу.

$$c_i = \frac{1}{n}\sum_{j=1..k_i} x_{ij} \tag{2.2}$$

Мерой разброса одинаково распределённых случайных величин (в данном случае – координат изображений некоторого класса) относительно их математического ожидания (центра класса) является стандартное отклонение. Для оценки стандартного отклонения используется несмещённая оценка (2.3).

$$\sigma_i = \sqrt{\frac{1}{k_i - 1}\sum_{j=1..k_i} d_{ij}^2} \tag{2.3}$$

где $d_{ij} = dist(x_{ij}, c_i)$ – мера расстояния между оценкой центра класса и координатами изображения, определяющая метрику в пространстве признаков.

Из работы [23] известно, что норма вектора признаков является показателем того, насколько сильно признаки проявлены на изображении. Зашумлённые и размытые изображения отображаются в векторы признаков с меньшей нормой, а чёткие – в векторы с большей нормой. Из-за этого, в ряде последних разработок предлагается оперировать не векторами признаков напрямую, а нормированными векторами признаков (вложения, embeddings) [23, 29, 31, 32, 33, 34, 38, 41], а в качестве меры расстояния использовать угол между векторами (2.4) (или косинус этого угла как меру сходства) [23, 29, 31, 32, 33, 34, 41].

$$\mathrm{dist}(a, b) = \cos^{-1}\left(\frac{a \cdot b}{\|a\|_2 \|b\|_2}\right) \tag{2.4}$$

Эта мера используется в настоящей работе.

Зависимость функции ошибки от разброса признаков внутри классов и расстояний между классами можно описать формулой (2.5). При минимизации этой функции ошибки, разброс координат в пределах класса должен снижаться,

а расстояния между классами – расти. Это накладывает ограничения (2.6) и (2.7) на производные функции ошибки.

$$\mathcal{L} = \mathcal{L}(\{\sigma_i\}_{i=1..p}, \{dist(c_i, c_j)\}_{i,j=1..p, i\neq j}) \quad (2.5)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_i} > 0 \quad (2.6)$$

$$\frac{\partial \mathcal{L}}{\partial dist(c_i, c_j)} < 0 \quad (2.7)$$

Возможны различные способы определения функции ошибки, удовлетворяющие этому определению.

Основываясь на идее, что классы должны отображаться в непересекающиеся участки пространства признаков, имеющие как можно меньший размер, можно предложить «первоначальный» вид функции, соответствующий формулам (2.8) и (2.9).

$$\mathcal{L} = \frac{1}{p^2} \sum_{i,j=1}^{p} \begin{cases} \lambda\sigma_i, \text{если } i = j \\ Intersect(i, j) \text{ в противном случае} \end{cases} \quad (2.8)$$

$$Intersect(i, j) = (\max(0, m + R(\sigma_i + \sigma_j) - dist(c_i, c_j)))^2 \quad (2.9)$$

где m – параметр, означающий минимальный отступ между областями классов, λ – параметр, определяющий отдельное влияние разброса (дисперсии) внутри классов на процесс обучения, R – коэффициент зависимости радиуса области класса от стандартного отклонения этого класса (по аналогии с $R\sigma$ – окрестностью математического ожидания случайной величины, вероятность непопадания измерения в которую ограничена неравенством Чебышёва (2.10)).

$$P(|X - \mu| \geq R\sigma) \leq \frac{1}{R^2} \quad (2.10)$$

Определение: область класса с центром c и стандартным отклонением σ – гипершар радиуса $R\sigma$ с центром в точке c в пространстве признаков с метрикой $dist$, R – заданный параметр.

Изображение считается принадлежащим к классу с центром в точке c, если этот центр – ближайший.

Функция *Intersect* определяет глубину пересечения областей двух классов (с учётом минимального отступа между ними). Если две области классов не пересекаются, то эта пара классов считается хорошо разделённой и не вносит вклада в расчёт градиентов. Классы сравниваются по схеме «все со всеми».

На практике (проверка проводилась на задаче по классификации изображений из набора данных MNIST), эта функция обладает двумя недостатками, не позволяющими получить с её помощью пригодную для практического применения ИНС:

а) При минимизации этой функции, ИНС попадает в локальный минимум, в котором все выделяемые признаки отображаются в малую область пространства признаков. Оценки дисперсии снижаются вместе с расстояниями между классами.

б) Для программной реализации сравнения областей классов по схеме «все со всеми» в выбранной библиотеке TensorFlow необходимо использовать функцию map_fn, что (вследствие особенностей программной реализации этой функции) многократно снизило скорость вычислений из-за переноса части вычислений с графического ускорителя на центральный процессор компьютера.

Для решения второй проблемы, сравнения областей классов производятся попарно, а не по схеме «все со всеми». Вследствие этого, формулировка функции ошибки меняется в соответствии с формулами (2.11) – (2.13).

$$x_{ij} = F(I_{ij}), i = 1..2p, j = 1..k_i \quad (2.11)$$

$$\mathcal{L} = \frac{1}{p}\sum_{i=1}^{p}\mathcal{L}_i \quad (2.12)$$

$$\mathcal{L}_i = L(c_{2i-1}, \sigma_{2i-1}, c_{2i}, \sigma_{2i}) \quad (2.13)$$

где L – оценка различимости двух классов по их областям.

Для решения первой проблемы применимы два совместимых подхода:

а) Разделить функцию ошибки на две независимые компоненты: ошибку внутриклассовой вариативности, зависящую только от оценки разброса признаков внутри класса, и ошибку межклассовой вариативности, зависящую от расстояний между классами.
б) Учитывать разброс внутри классов по отношению к расстоянию между классами.

В результате применения этих исправлений, функция ошибки приобретает вид согласно формуле (2.14).

$$L(c_i, \sigma_i, c_j, \sigma_j) = (\max(0, m - \mathrm{dist}(c_i, c_j)))^2 + R\frac{\sigma_i + \sigma_j}{\mathrm{dist}(c_i, c_j)} \qquad (2.14)$$

где m – минимальное расстояние между центрами классов.

Использование этой «исправленной» функции ошибки приводит к тому, что ИНС обучается выделять идентифицирующие класс признаки.

Важно отметить, что второе слагаемое этой функции само по себе (2.15) учитывает расстояние между центрами классов правильным образом, то есть в соответствии с формулой (2.7).

$$\frac{\partial\left[\frac{\sigma_i + \sigma_j}{\mathrm{dist}(c_i, c_j)}\right]}{\partial \mathrm{dist}(c_i, c_j)} < 0 \qquad (2.15)$$

Действительно, использование «упрощённой» функции ошибки (2.16), в которой есть только правое слагаемое, позволяет обучить ИНС выделять идентифицирующие признаки.

$$L(c_i, \sigma_i, c_j, \sigma_j) = \frac{\sigma_i + \sigma_j}{\mathrm{dist}(c_i, c_j)} \qquad (2.16)$$

Недостатком этого варианта функции ошибки является отсутствие параметров, определяющих границу сильного нарушения критерия различимости классов и определяющих разницу в силе воздействия сложных и

простых примеров. Для исправления этого недостатка функцию ошибки нужно представить в виде формулы (2.17)

$$\mathrm{L}(\mathrm{c_i}, \sigma_\mathrm{i}, \mathrm{c_j}, \sigma_\mathrm{j}) = \mathrm{F}\left(\frac{\sigma_\mathrm{i}+\sigma_\mathrm{j}}{\mathrm{dist}(\mathrm{c_i},\mathrm{c_j})}\right) \qquad (2.17)$$

где $F(x)$ – монотонно возрастающая функция, определённая на $x \geq 0$ и имеющая один или более параметров.

Исходя из определения области класса и параметра R, функция $F(x)$ должна обладать следующими свойствами:

а) Если $Rx < 1$, то $F(x)$ – малая величина.

б) Если $Rx > 1$, то $F(x)$ – большая величина.

в) F(x) имеет параметр, управляющий разницей воздействия на сильные и на слабые примеры.

Этим критериям соответствует функция ошибки (2.18).

$$\mathrm{F(x)} = \frac{1}{\mathrm{Rn}}(\mathrm{Rx})^\mathrm{n} \qquad (2.18)$$

Знаменатель Rn добавлен для того, чтобы производная этой функции в точке $x = \frac{1}{R}$ имела значение 1, что приводит к тому, что $F'(x) < 1$ при $x < \frac{1}{R}$, и $F'(x) > 1$ при $x > \frac{1}{R}$. Из-за наличия в производной функции операции возведения в степень, производная в указанных ситуациях будет принимать значения, значительно большие или меньшие 1 (предполагается, что $n > 1$). Таким образом, конечный вариант функции ошибки имеет вид (2.19).

$$\mathrm{L}(\mathrm{c_i}, \sigma_\mathrm{i}, \mathrm{c_j}, \sigma_\mathrm{j}) = \frac{1}{\mathrm{Rn}}\left(\mathrm{R}\frac{\sigma_\mathrm{i}+\sigma_\mathrm{j}}{\mathrm{dist}(\mathrm{c_i},\mathrm{c_j})}\right)^\mathrm{n} \qquad (2.19)$$

Параметр n определяет, насколько воздействие сложных примеров сильнее, чем простых.

Параметр R интерпретируется, как и ранее. Если сумма радиусов областей сравниваемых классов больше, чем расстояние между их центрами,

они считаются плохо различимыми и вносят значительный вклад в значение функции ошибки.

Определение оптимальных (позволяющих получить хорошо обученную ИНС за ограниченное время) значений параметров R и n является нетривиальной задачей для аналитического решения, поэтому для поиска этих параметров (для «эталонной» задачи обучения ИНС для классификации изображений из набора данных MNIST) применено 2 подхода: табличный поиск и алгоритм минимизации Нелдера-Мида.

2.6 Протоколы тестирования

Для тестирования ИНС используются 2 протокола:

a) Стандартный протокол LFW [11] «Unrestricted with labeled outside data». Является де-факто стандартом для оценки точности верификации (но на данный момент уже не является самым сложным). Имеет простую реализацию (в частности, за основу взята программная реализация этого протокола, использованная в одной из реализаций обучения ИНС FaceNet и доступная в сети интернет по адресу https://github.com/davidsandberg/facenet) и используется для тестирования работы ИНС в процессе обучения. Протокол включает в себя 6000 пар изображений, разделённых на 10 частей, в каждой из которых содержится 300 совпадений и 300 несовпадений. Эти 10 частей образуют 10 разделений на обучающий и тестовый наборы данных (в соотношении 9 к 1), используемые для перекрёстной проверки. Для каждого разделения вычисляется порог принятия решения, дающий максимальную точность на обучающем подмножестве и частоты (для разных значений порога принятия решений) истинно-положительных и ложно-положительных решений на тестовом множестве. Точности для каждого разделения

вычисляются на тестовом множестве с использованием вычисленного ранее лучшего порога принятия решений. Полученные значения точности и частоты истинно-положительных и ложно-положительных решений усредняются по всем разделениям. Усреднённые частоты истинно-положительных и ложно-положительных решений представляют собой кривую рабочей характеристики ИНС. Для вычисления частоты истинно-положительных решений при фиксированной частоте ложно-положительных также производится перекрёстная проверка с 10-ю разбиениями. На каждом разбиении производится поиск порога принятия решений, обеспечивающего заданную частоту ложно-положительных решений на обучающем подмножестве. Найденный порог используется для вычисления частот истинно-положительных и ложно-положительных решений на тестовом подмножестве каждого разделения. Результаты всех разделений усредняются.

б) Протокол BLUFR [15]. Протокол реализован на языке MATLAB и определяет точность работы ИНС по файлу .mat, содержащему выделенные признаки для каждого изображения из набора данных LFW. Это затрудняет его применение во время обучения ИНС. Протокол позволяет определить точность работы ИНС в задачах верификации и идентификации при заданных ранге и FAR.

2.7 Организация процесса обучения

При обучении минимизируется функция ошибки. Предложенная функция имеет два параметра, от значений которых существенно зависит результат обучения сети. При заданных значениях параметров цикл обучения выглядит следующим образом (псевдокод):

```
эпоха_обучения=1
Пока эпоха_обучения < количество_эпох_обучения:
    Выбор изображений для обучения
```

```
    Обучение на выбранных изображениях
    Тестирование по протоколу LFW
    Сохранение ИНС
    эпоха_обучения+=1
Тестирование по протоколу LFW
Вычисление признаков для тестирования по протоколу BLUFR.
```

2.8 Выводы

Произведена разработка ИНС для выделения идентифицирующих признаков: определены средства разработки, описаны входные и выходные данные ИНС, протоколы тестирования, архитектура ИНС, описана процедура обучения.

Разработана новая функция ошибки для обучения разрабатываемой ИНС. Результаты оптимизация параметров функции ошибки и проверки её эффективности изложены в следующей главе.

3 Обучение и оценка качества сети

3.1 Проверка эффективности предложенной функции ошибки

Для проверки эффективности использования предложенной функции ошибки она применяется для обучения ИНС для решения задачи классификации изображений из набора данных MNIST.

3.1.1 Описание задачи

MNIST [42] – база данных монохромных изображений размера 28*28 пикселей, содержащих рукописные цифры. База данных содержит 60000 обучающих и 10000 тестовых примеров. Примеры изображений разных классов из этого набора данных представлены на рисунке 3.1.

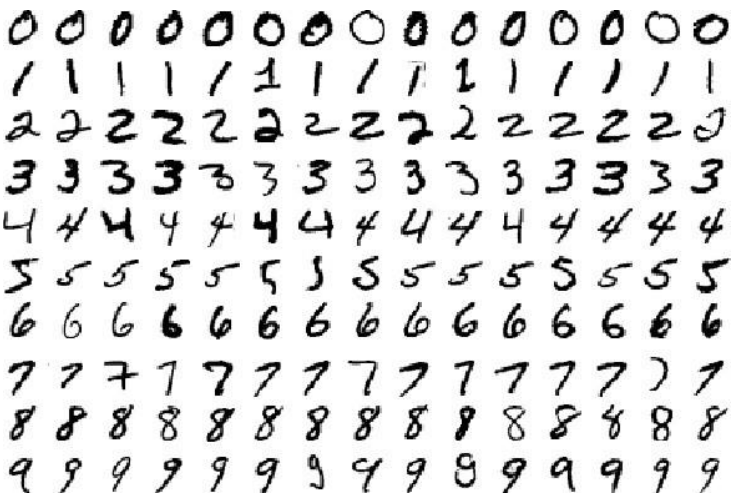

Рисунок 3.1 – Примеры изображений в наборе данных MNIST.

Цель ИНС – определить, какая цифра изображена на изображении. Эта задача сводится к классификации изображений в один из десяти классов.

3.1.2 Описание архитектуры ИНС

Для решения поставленной задачи классификации изображений из набора данных MNIST используется свёрточная ИНС, архитектура которой описана в таблице 3.1.

Таблица 3.1 – Описание архитектуры ИНС для экспериментов на наборе данных MNIST.

Тип слоя	Размер ядра (окна)	Шаг	Размер выхода	Количество параметров
Input			[28, 28, 1]	
Conv+ReLU	5	1	[28, 28, 32]	832
MaxPool	2	2	[14, 14, 32]	
Conv+ReLU	5	1	[14, 14, 64]	51264
MaxPool	2	2	[7, 7, 64]	
FC+ReLU			[1024]	3212288
FC			[16]	16400
Всего				3280784

Пояснения к таблице:

а) Input – слой входных данных ИНС,

б) Conv+ReLU – свёрточный слой с функцией активации ReLU,

в) MaxPool – слой подвыборки по максимальному значению,

г) FC – полносвязный слой.

Функция активации ReLU является одной из самых часто используемых функций активации и определяется формулой (3.1).

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.1}$$

Описанная архитектура используется для извлечения из классифицируемого изображения пригодных для классификации признаков (выбранная размерность вектора признаков составляет 16).

3.1.3 Обучение с функцией ошибки Softmax loss

Для сравнения, результаты экспериментов по обучению ИНС с предложенными видами функции ошибки сопоставляются с результатами ИНС, обученной с функцией ошибки Softmax loss. При обучении ИНС с этой функцией ошибки, после слоя извлечения признаков (3.2) добавляются полносвязный слой с лилейной функцией активации (3.3) и нормализация функцией Softmax (3.4). Функция ошибки определяется как среднее арифметическое (по обучающим примерам) перекрёстной энтропии истинных и предсказанных вероятностей (3.5).

$$\mathrm{x_k} = \mathrm{F(I_k)} \tag{3.2}$$

$$\mathrm{p_k} = \mathrm{Wx_k} + \mathrm{b} \tag{3.3}$$

$$\mathrm{y^p_{k,i}} = \frac{\mathrm{e^{p_{k,i}}}}{\sum_{\mathrm{j=1..10}} \mathrm{e^{p_{k,j}}}} \tag{3.4}$$

$$\mathcal{L} = -\frac{1}{\mathrm{N}} \sum_{\mathrm{k=1}}^{\mathrm{N}} \sum_{\mathrm{i=1}}^{10} \mathrm{y^t_{k,i}} \log(\mathrm{y^p_{k,i}}) \tag{3.5}$$

Где I_k – классифицируемое изображение, x_k – извлечённые из него признаки, W и b – весовые коэффициенты полносвязного слоя, $y^p_{k,i}$ и $y^t_{k,i}$– предсказанная и истинная вероятности принадлежности k-го изображения i-му классу соответственно.

Точность классификации определяется как относительная частота таких примеров из тестовой выборки, для которых выполняется равенство (3.6).

$$\mathrm{argmax_i}(\mathrm{y^t_{k,i}}) = \mathrm{argmax_i}(\mathrm{y^p_{k,i}}) \tag{3.6}$$

Обучение производилось в течении 20000 итераций, каждая итерация включает в себя 50 обучающих примеров. Для оптимизации во всех экспериментах на наборе данных MNIST используется оптимизатор Adam с параметрами learning_rate=0.0001, $\mathrm{beta_1}$=0.9, $\mathrm{beta_2}$=0.999, epsilon=10^{-8}.

Точность обученной ИНС составила 98.93%. График прогресса обучения (Рисунок 3.2) показывает точность обучаемой ИНС на тестовой выборке набора данных MNIST в зависимости от итерации обучения. Время обучения составило 5 минут 20 секунд на компьютере с видеокартой Nvidia GeForce GTX 660ti, процессором Intel Core i5-3570 и объёмом ОЗУ 8 гигабайт (все указанные далее эксперименты проведены на этой конфигурации оборудования).

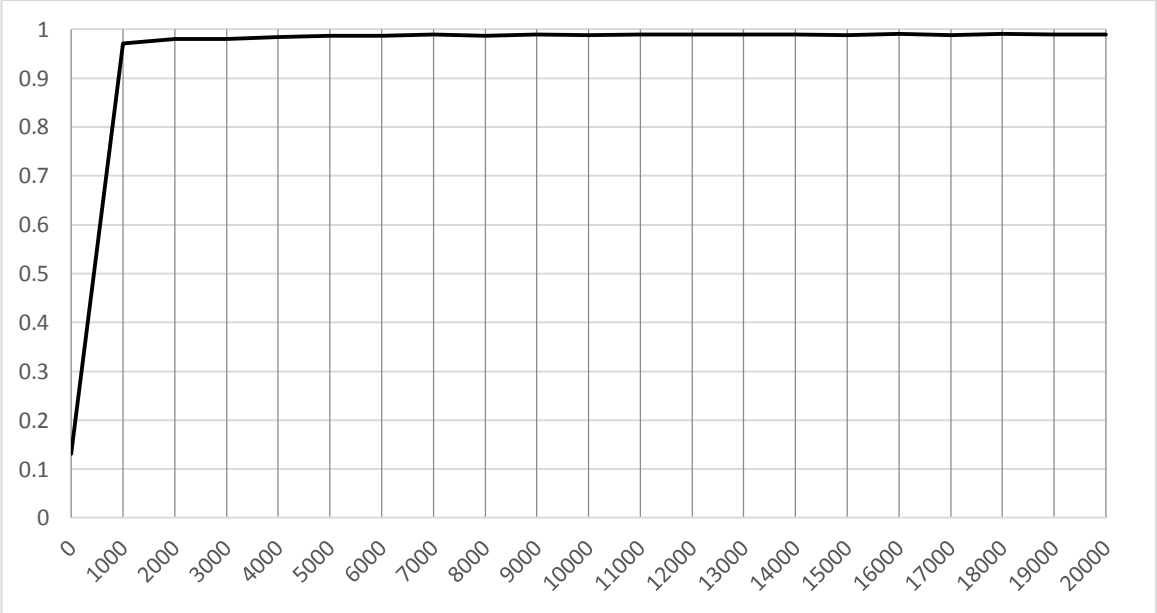

Рисунок 3.2 – График изменения точности ИНС на тестовой выборке в процессе обучения с функцией ошибки Softmax loss.

3.1.4 Обучение с «первоначальной» функцией ошибки

При обучении ИНС с «первоначальной» версией функции ошибки обучающие примеры каждого класса, для упрощения программой реализации, выбираются группами одинакового размера. Функция ошибки описывается формулами (3.7) и (3.8).

$$\mathcal{L} = \frac{1}{p^2} \sum_{i,j=1}^{p} \begin{cases} \lambda\sigma_i, \text{если } i = j \\ \text{Intersect}(i, j), \text{в противном случае} \end{cases} \quad (3.7)$$

$$\text{Intersect}(i, j) = \max(0, m + R(\sigma_i + \sigma_j) - \text{dist}(c_i, c_j))^2 \quad (3.8)$$

где R и m – параметры, определяющие зависимость радиуса области класса от стандартного отклонения примеров в нём и минимальный отступ между областями классов соответственно, λ – параметр, определяющий отдельное влияние стандартных отклонений σ на значение функции ошибки.

Для определения точности на тестовой выборке, необходим классифицирующий слой. Этот слой описывается формулой (3.3).

Так же, как и прежде, точность классификации вычисляется как относительная частота примеров из тестовой выборки, для которых выполняется равенство (3.9).

$$\mathrm{argmax_i}(y^{t}_{k,i}) = \mathrm{argmax_i}(p_{k,i}) \tag{3.9}$$

Строки матрицы W из формулы (3.3) представляют собой эталонные векторы признаков классов [26], вследствие чего должны совпадать с центрами классов. Для этого, к функции ошибки (3.10) добавляется мера их разницы.

$$\mathcal{L}_{\mathrm{full}} = \mathcal{L} + \|W - C\|_2 \tag{3.10}$$

где C – матрица центров классов, рассчитываемая для каждого пакета обучающих примеров. Для того, чтобы порядок классов в матрице центров совпадал с порядком в матрице W, пакет обучающих примеров формируется таким образом, что в нём последовательно расположены группы одинакового размера примеров всех классов: n изображений нулей, n изображений единиц, и т.д. Пакет обучающих примеров включает в себя 50 примеров, по 5 примеров каждого класса.

График прогресса обучения ИНС при значениях параметров R=3, m=0.5 и λ=0.1 (в сравнении с аналогичным графиком для функции ошибки Softmax loss) изображён на рисунке 3.3. Результирующая точность ИНС составила 89.19%, обучение длилось 20000 итераций и заняло 1 час 49 минут.

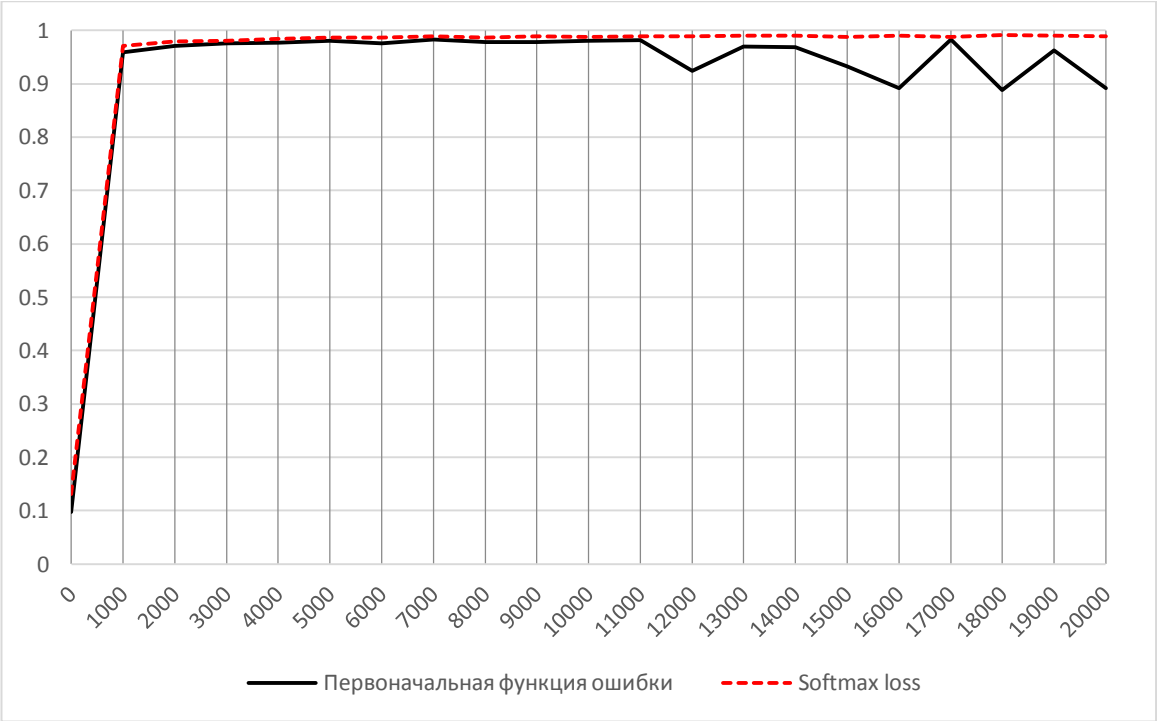

Рисунок 3.3 – Прогресс точности ИНС в процессе обучения в задаче классификации на тестовом наборе данных MNIST с первоначальным вариантом функции ошибки.

По графику 3.3 видно, что применение предложенной функции ошибки даёт полезный результат. Причиной падения точности ИНС ближе к концу обучения является сужение области в пространстве признаков, в которую отображаются изображения.

3.1.5 Обучение с исправленной функцией ошибки

Для исправления недостатков функции ошибки, испытанной в п. 3.1.4, она изменяется в соответствии с формулами (3.11) и (3.12).

$$\mathcal{L} = \frac{1}{p}\sum_{i=1}^{p} L(c_{2i-1}, \sigma_{2i-1}, c_{2i}, \sigma_{2i}) \tag{3.11}$$

$$L(c_i, \sigma_i, c_j, \sigma_j) = \max(0, m - \mathrm{dist}(c_i, c_j))^2 + R\frac{\sigma_i + \sigma_j}{\mathrm{dist}(c_i, c_j)} \tag{3.12}$$

Во избежание переобучения, на каждой итерации обучения сравниваются области разных пар классов. Для этого, группы обучающих примеров перетасовываются произвольным образом.

Для определения точности на тестовой выборке, как и в предыдущих экспериментах, после слоя извлечения признаков добавляется классифицирующий слой (3.3). Так как классы обучающих примеров не идут последовательно в пакете обучающих примеров, как прежде, веса матрицы W обучаются отдельно с функцией ошибки Softmax loss, после обучения извлекающей признаки ИНС с использованием исправленной функции ошибки.

Обучение ИНС производилось, на протяжении 20000 итераций. Использованы значения параметров m=0.2, R=1.0. Время, затраченное на обучение, составило 8 минут. График, изображённый на рисунке 3.4, показывает прогресс изменения точности ИНС на тестовом наборе данных MNIST в процессе обучения, в сравнении с аналогичным графиком для функции ошибки Softmax loss. Точность ИНС в результате обучения составляет 97.33%.

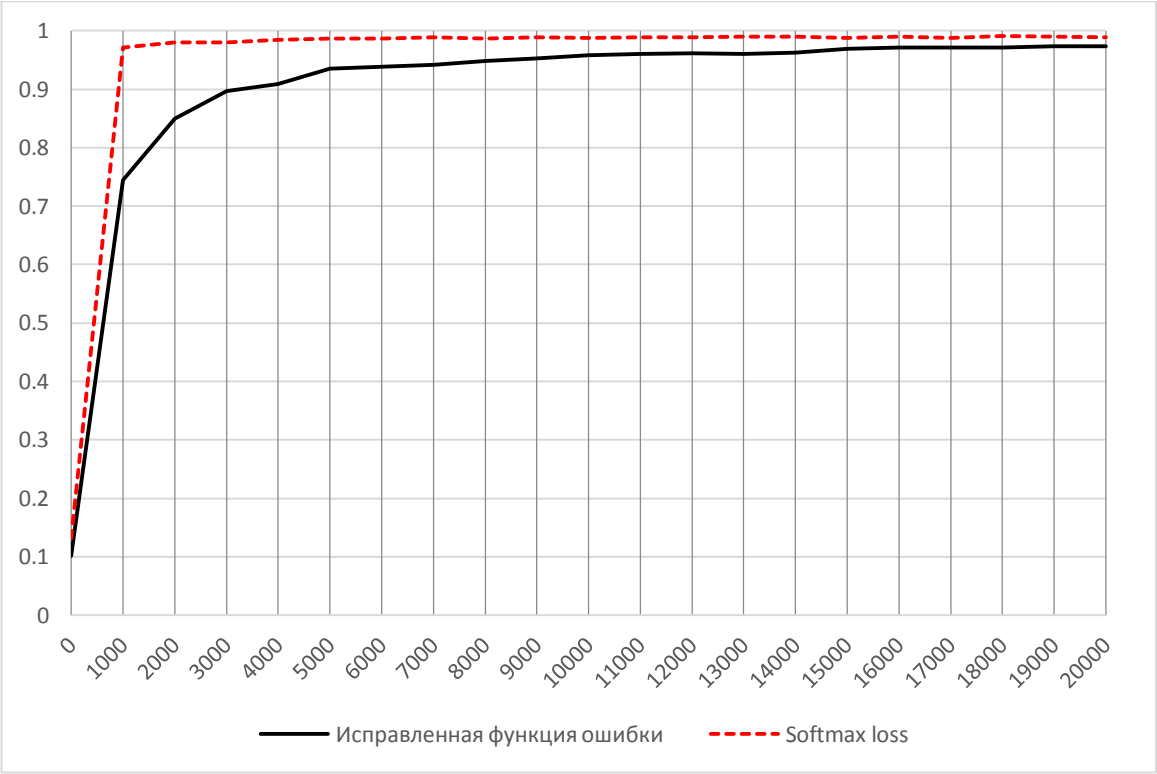

Рисунок 3.4 – Прогресс обучения ИНС для задачи классификации на наборе данных MNIST с исправленной функцией ошибки.

График 3.4 демонстрирует планомерный рост точности ИНС в процессе обучения. Точность ИНС растёт медленнее, чем при обучении с функцией ошибки Softmax loss.

3.1.6 Обучение с упрощённой функцией ошибки

Условия обучения ИНС с использованием упрощённой функции ошибки совпадают с условиями, описанными в п. 3.1.5. Упрощённая функция ошибки описывается формулами (3.11) и (3.13).

$$L(c_i, \sigma_i, c_j, \sigma_j) = \frac{\sigma_i + \sigma_j}{dist(c_i, c_j)} \tag{3.13}$$

График прогресса обучения с упрощённой функцией ошибки изображён на рисунке 3.5, в сравнении с аналогичным графиком исправленной функции ошибки. Обучение заняло 7 минут 50 секунд, при этом была достигнута точность 97.38%.

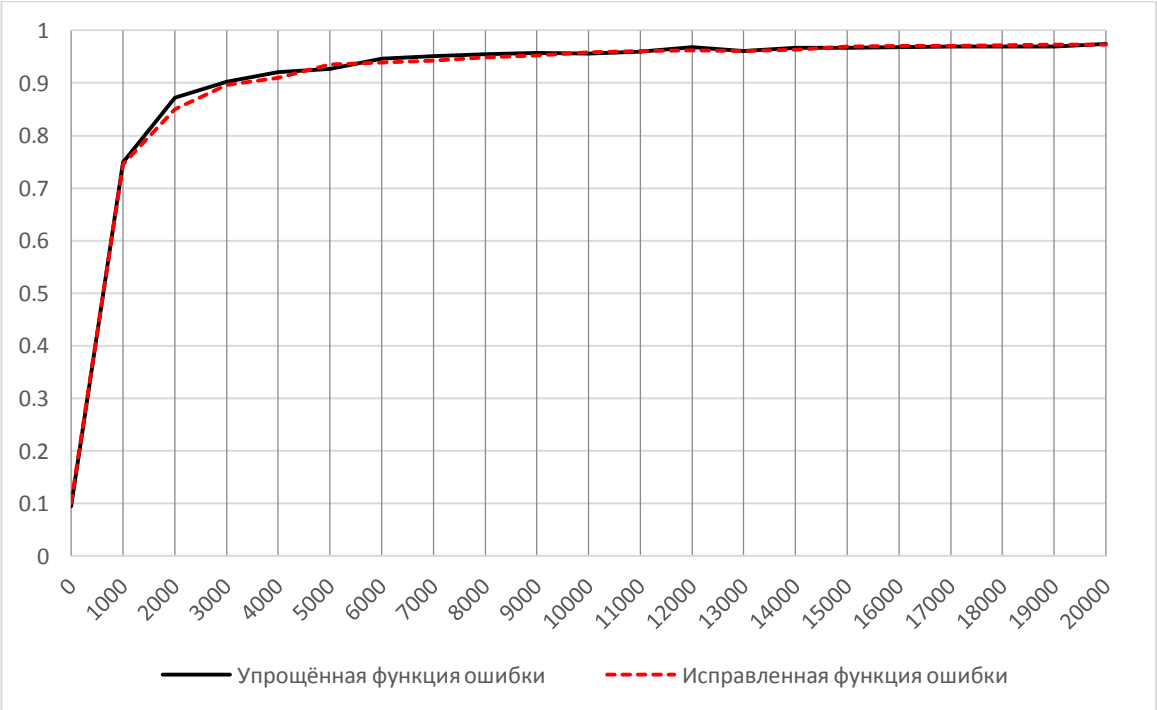

Рисунок 3.5 – Сравнение прогресса обучения ИНС с исправленной и с упрощённой функциями ошибки.

По графику 3.5 видно, что упрощение функции ошибки практически не влияет на скорость обучения и результирующую точность.

3.1.7 Обучение с конечным вариантом функции ошибки

Для введения параметров, позволяющих контролировать процесс обучения, используется конечный вариант функции ошибки, определяемый формулами (3.11) и (3.14).

$$L(c_i, \sigma_i, c_j, \sigma_j) = \frac{\left(R\frac{\sigma_i+\sigma_j}{dist(c_i,c_j)}\right)^n}{Rn} \quad (3.14)$$

Обучение с этой функцией ошибки производилось со значениями параметров R=3, n=10. Как и предыдущие эксперименты, обучение длилось 20000 итераций. Оно заняло 8 минут и позволило достигнуть точности 98.55%. На рисунке 3.6 изображён график прогресса точности ИНС в процессе обучения, для сравнения также изображены графики прогресса обучения с упрощённой функцией ошибки и функцией ошибки Softmax loss.

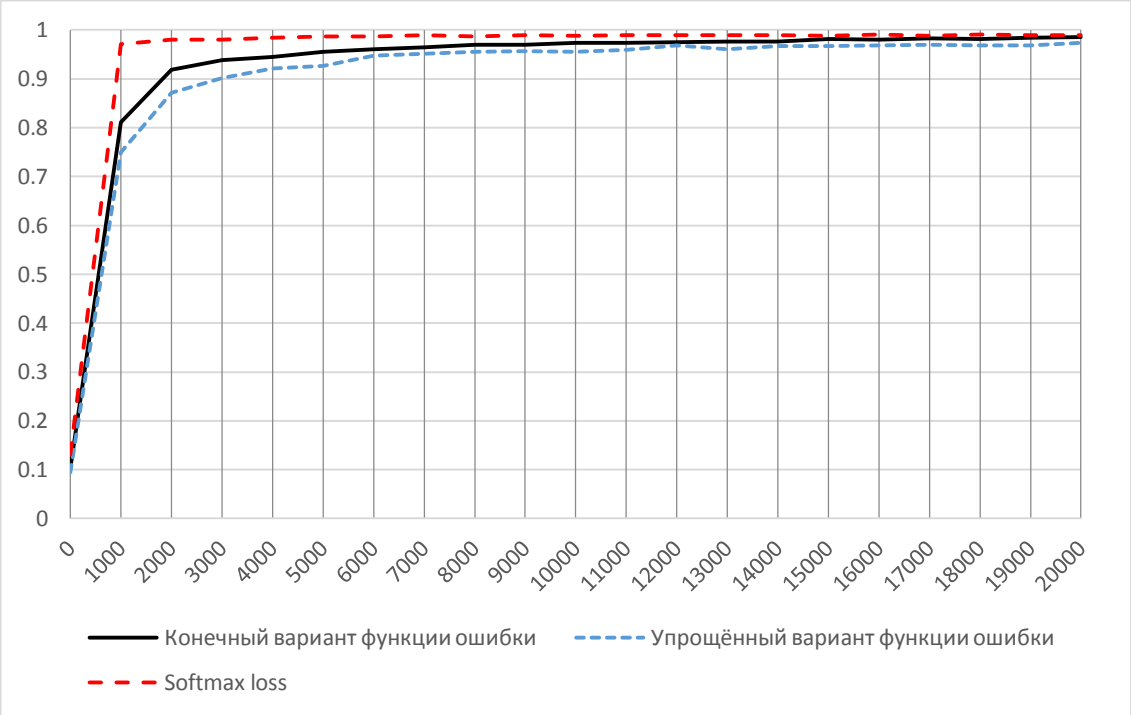

Рисунок 3.6 – Сравнение прогресса обучения с финальной функцией ошибки, с упрощённой и функцией ошибки Softmax loss.

На графике 3.6 видно, что при обучении с конечной функцией ошибки точность растёт быстрее, чем с упрощённой.

3.2 Сравнение функции ошибки с аналогами

Для сравнения предложенной функции ошибки и функции ошибки Softmax loss были обучены две ИНС (о описанной выше архитектурой): одна – с функцией ошибки Softmax loss, другая – с предложенной функцией ошибки (конечный вариант) с параметрами R=3, n=6. Длительность обучения в обоих случаях составила 100000 итераций. Оно продлилось 26 минут 40 секунд для функции ошибки Softmax loss и 39 минут для предложенной функции ошибки. Графики изменения точности обеих ИНС в процессе обучения представлены на рисунке 3.7.

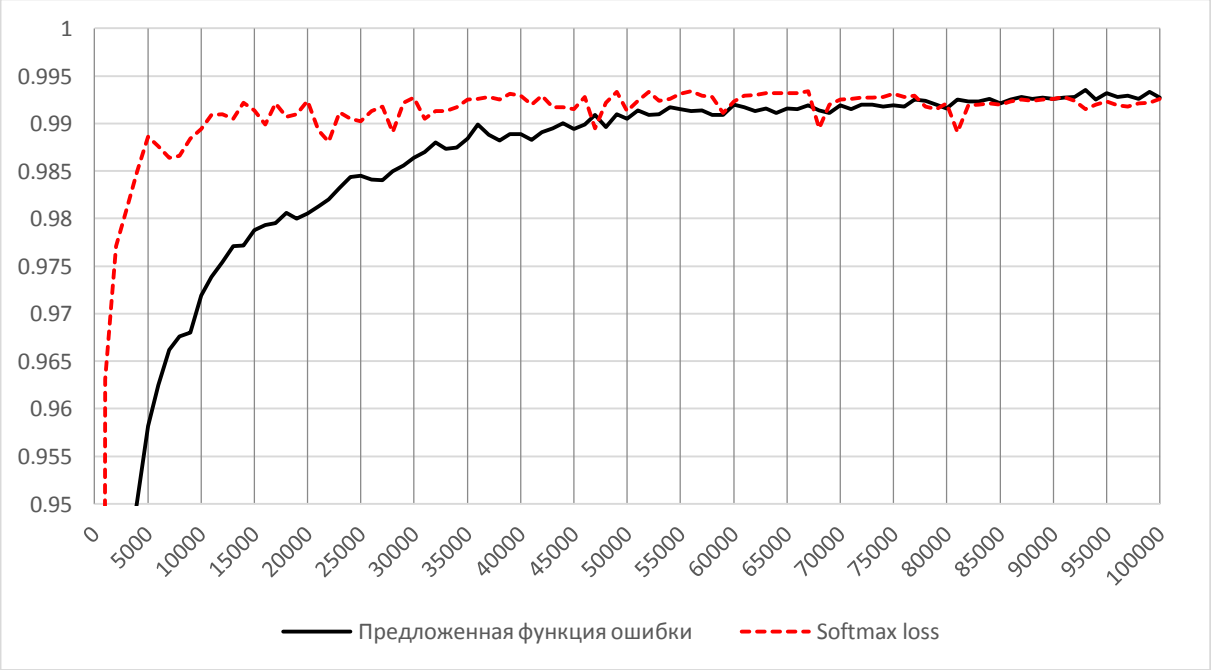

Рисунок 3.7 – Сравнение прогресса обучения ИНС с предложенной функцией ошибки и функции ошибки Softmax loss.

По графику видно, что ИНС, обучаемая с предложенной функцией ошибки, достигла той же точности, что и ИНС, обучаемая с функцией ошибки Softmax loss. Сравнение точности ИНС, обученных с разными функциями ошибки, представлено в таблице 3.4.

Таблица 3.2 – Сравнение результатов обучения с различными функциями ошибки в задаче классификации изображений из тестового набора данных MNIST.

Функция ошибки	Точность
COCO [31]	99.7%
Large-Margin Softmax Loss [26]	99.69%
Предложенная функция ошибки	**99.27%**
Softmax loss	99.26%
Contrastive Center Loss [30]	99.17%
L_2-constrained Softmax Loss [23]	99.05%
Center Invariant Loss [27]	95.03%

Полученный результат уступает только ИНС с функциями ошибок Large-Margin Softmax Loss и COCO. Результат не является окончательным – его можно улучшать за счет усложнения архитектуры ИНС, подбора лучших значений параметров и более продолжительного обучения.

3.3 Подбор параметров функции ошибки

Конечный вариант функции ошибки имеет два параметра, значение которых влияет на точность обученной ИНС. Для анализа характера этой зависимости и подбора оптимальных значений были использованы два метода:

а) Табличный поиск,

б) Оптимизация по алгоритму Нелдера-Мида.

3.3.1 Табличный поиск

При табличном поиске ИНС обучаются и тестируются на сетке с узлами, определёнными формулами (3.15) и (3.16).

$$R = 1,2,3,4,5 \tag{3.15}$$

$$n = 2,3,4,5,\dots,19 \tag{3.16}$$

Для каждого узла обучение производится 3 раза, после чего результат усредняется. Длительность обучения составляет 50000 итераций.

Результаты табличного поиска изображены в виде графика на рисунке 3.8. На рисунке 3.9 изображены усреднённые по значениям параметра n значения точности обученных ИНС, в зависимости от значения параметра R.

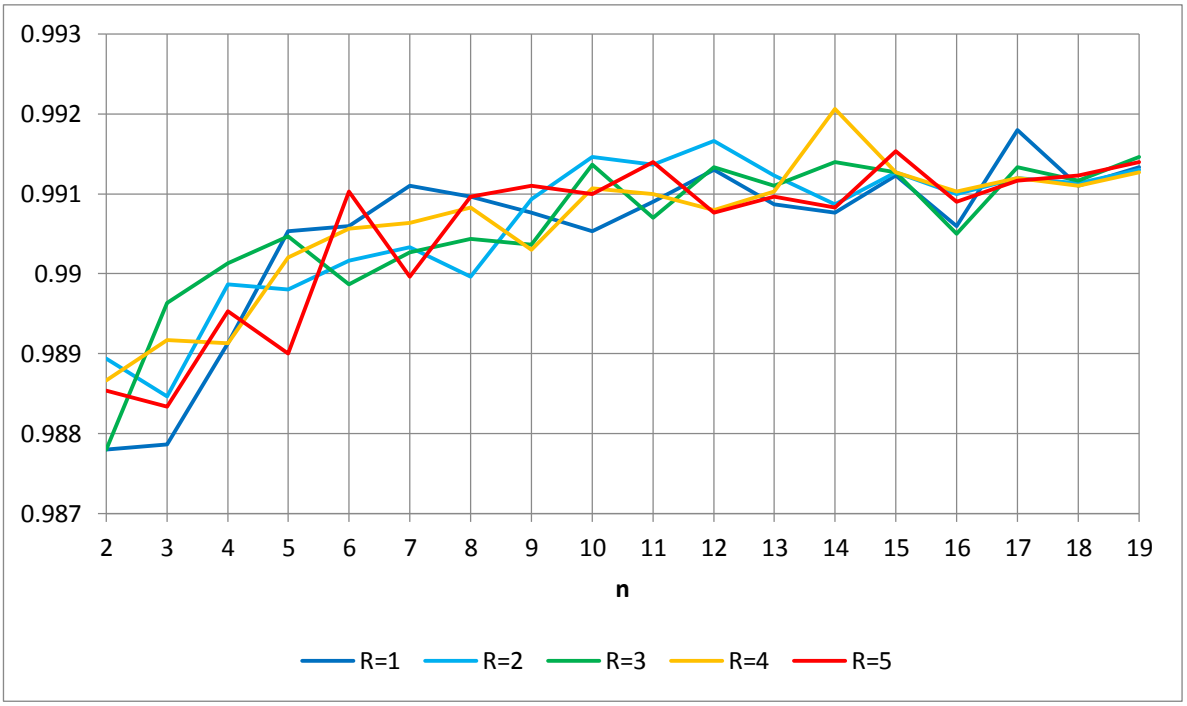

Рисунок 3.8 – Результаты табличного поиска в виде графика зависимости точности обученных ИНС от параметра n при различных значениях параметра R.

По графику 3.8 видно, что точность обученных ИНС возрастает при значениях параметра $n \leq 10$, после чего точность перестаёт заметно возрастать.

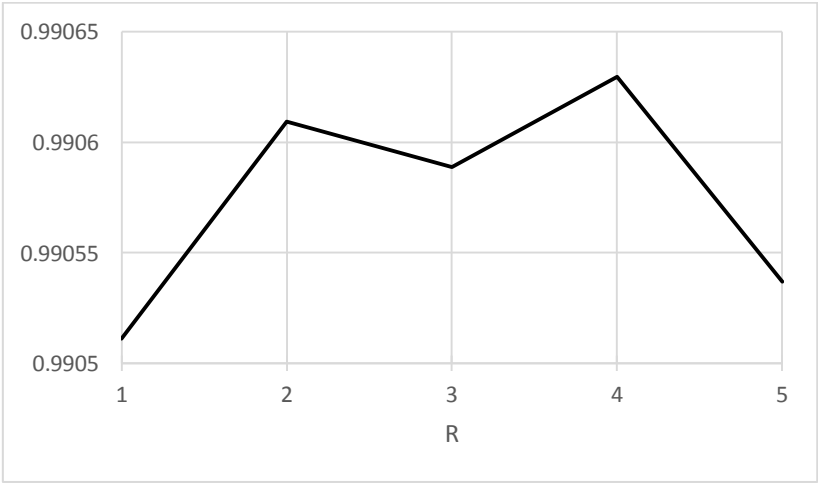

Рисунок 3.9 – График зависимости средней точности по всем значениям параметра n для различных значений параметра R.

График 3.9 показывает, что параметр R в слабо влияет на точность обученных ИНС, тем не менее крайние значения R=1 и R=5 являются наихудшими. В таблице 3.2 показаны лучшие 10% результатов табличного поиска.

Таблица 3.3 – Лучшие 10% результатов при табличном поиске.

R	n	Точность, %
4	14	99.207
1	17	99.18
2	12	99.167
5	15	99.153
2	10	99.147
3	19	99.147
3	14	99.14
5	11	99.14
5	19	99.14

Средняя точность на всех наборах параметров составила 99.06%, худший результат из всех составляет 98.78%. Лучший из результатов превосходит худший на 0.43% и средний на 0.15%.

3.3.2 Поиск по алгоритму Нелдера-Мида

Для поиска оптимальных значений параметров произведён поиск по алгоритму Нелдера-Мида. Была использована реализация этого алгоритма, входящая в состав библиотеки SciPy (в этой реализации отсутствует процедура восстановления симплекса). Использованы следующие параметры алгоритма $\alpha = 1, \beta = 2, \gamma = 0.5, \delta = 0.5$. В качестве начального симплекса использованы 3 лучших результата табличного поиска (значения продублированы в таблице 3.3).

Таблица 3.4 – Начальный симплекс при поиске по алгоритму Нелдера-Мида.

R	n
4	14
1	17
2	12

Алгоритм Нелдера-Мида остановился после 18 итераций. Результатом оптимизации являются значения R=1.134, n=19.25. Точность обученной с этими значениями параметров ИНС составила 99.27%, что на 0.063% лучше, чем лучший результат табличного поиска.

Большое найденное значение параметра n=19.25 говорит о том, что точность обученной ИНС продолжает возрастать при увеличении $n>10$. Однако, неограниченное повышение значения этого параметра не является панацеей для повышения точности обученной ИНС из-за рисков переобучения ИНС и возникновения ошибки переполнения.

Низкое найденное значение параметра R говорит о сложном характере его влияния на скорость обучения ИНС. Этот параметр используется для разделения простых и сложных пар обучающих примеров. Высокая точность обученной ИНС при малом (близком к 1) значении этого параметра говорит о том, что в начале обучения более эффективными являются малые значения. В то же время, для достижения большей компактности областей классов нужны большие значения этого параметра.

3.4 Обучение ИНС для идентификации людей по лицу

3.4.1 Описание процедуры обучения

Архитектура обучаемой ИНС описана в таблице 2.1. Длительность обучения составляет 1500 эпох. Одна эпоха обучения содержит 10500 разных классов (персон), по 5 разных изображений на каждый (всего в обучающем наборе данных 10575 классов, 10572 из них представлены как минимум 5-ю изображениями). Один пакет обучающих примеров содержит 20 классов, одна эпоха содержит 525 пакетов обучающих примеров.

Для оптимизации используется алгоритм градиентного спуска. Начальная скорость обучения (learning rate) составляет 0.1. Для повышения точности

обученной ИНС, используется экспоненциальное затухание скорости обучения: каждые 20 эпох скорость обучения уменьшается на 10%.

Для обучения были выбраны параметры функции ошибки $R=3$ и $n=6$. Значение $R=3$ выбрано как среднее между 1 и 5, так как по графику 3.8 видно, что крайние значения $R=1$ и $R=5$ – хуже, чем значения между ними. Значение $n=6$ выбрано потому, что в результате эксперимента при значении $n=10$ ИНС мгновенно переобучилась и перестала обучаться, в то время как при $n=6$ обучение ИНС на наборе данных MNIST даёт приемлемый результат.

3.4.2 Результаты обучения

Процедура обучения продлилась 9 дней и 22 часа. В результате обучения была достигнута точность 78.1±1.3% по стандартному протоколу LFW (1.1% при FAR = 0.001). График изменения точности по протоколу LFW в процессе обучения изображён на рисунке 3.10. Кривая PX обученной ИНС изображена на рисунке 3.11.

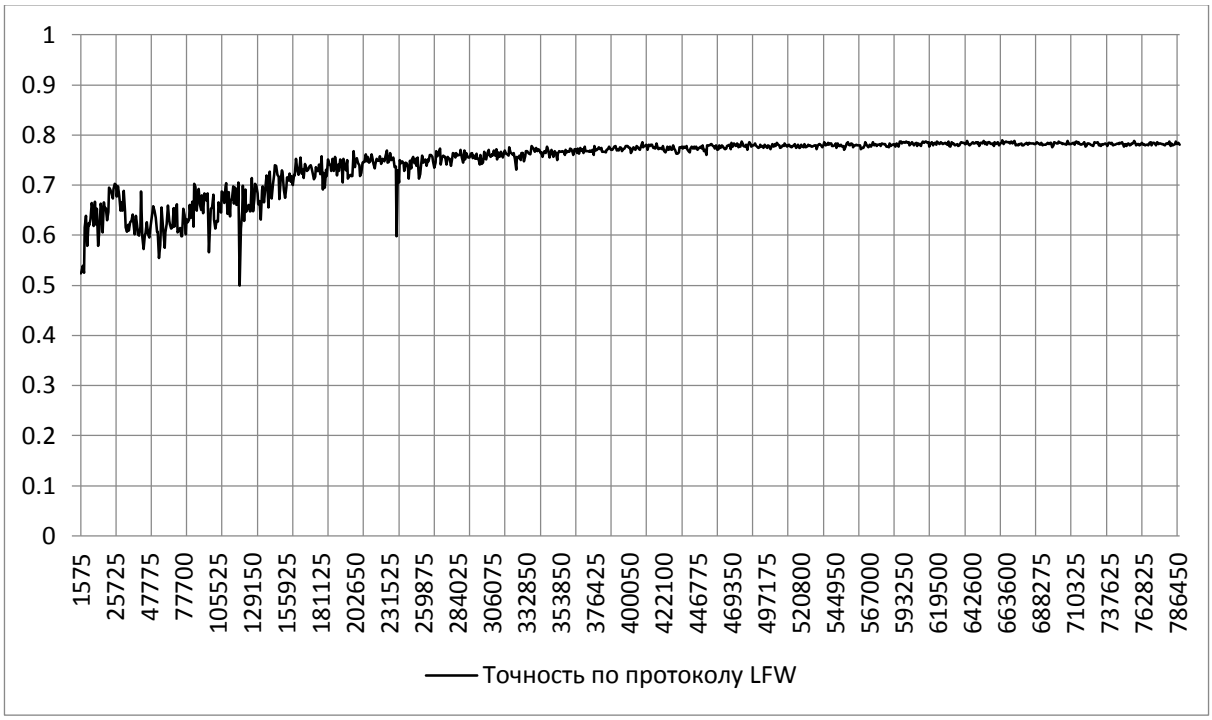

Рисунок 3.10 – Прогресс изменения точности верификации по протоколу LFW.

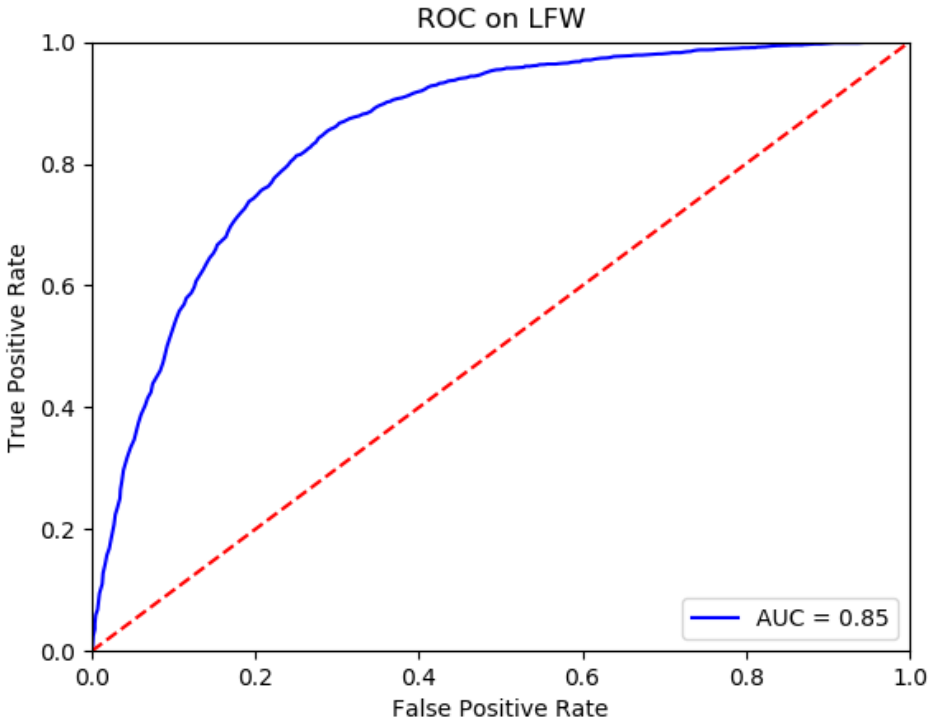

Рисунок 3.11 – Кривая РХ обученной ИНС по протоколу LFW.

Точность верификации по протоколу BLUFR составила 1.2% при FAR=0.1%. Точность идентификации ранга 1 на открытом множестве при FAR=1% по протоколу BLUFR составляет 0%. На рисунках 3.12 и 3.13 изображены графики кривых РХ в задачах верификации и идентификации на открытом множестве по протоколу BLUFR соответственно.

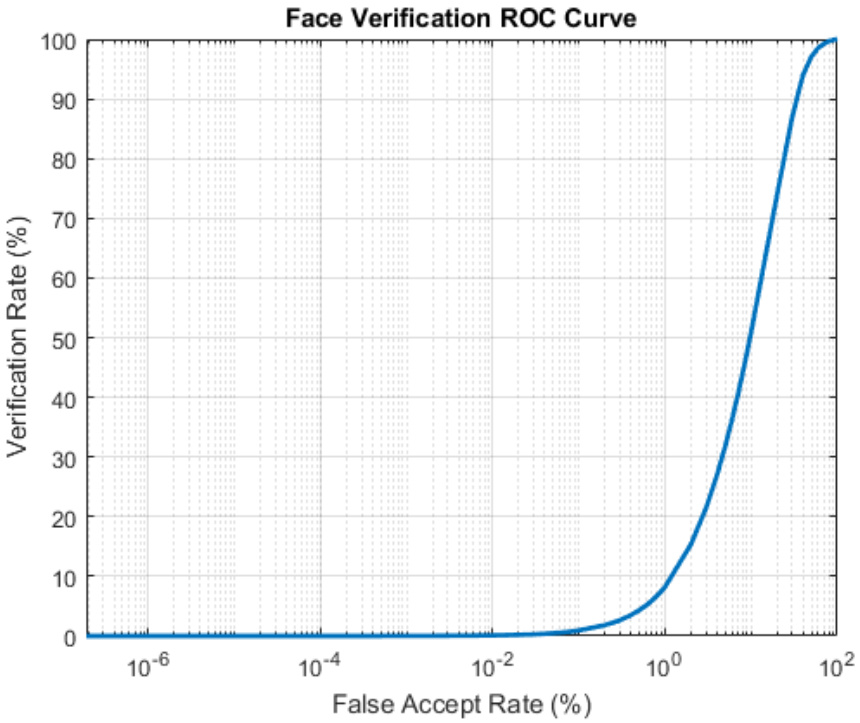

Рисунок 3.12 – График кривой РХ обученной ИНС в задаче верификации по протоколу BLUFR.

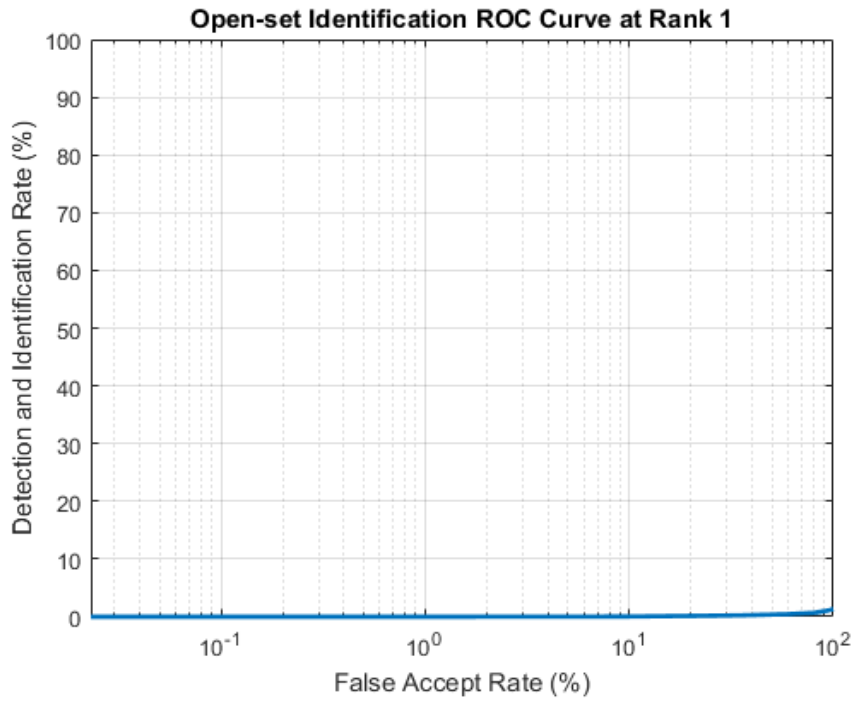

Рисунок 3.13 – График кривой РХ обученной ИНС в задаче идентификации на открытом множестве по протоколу BLUFR.

3.5 Использование разработанной ИНС для решения прикладных задач

Полученная ИНС может быть использована как подсистема обработки сигналов для решения множества прикладных задач:

а) Авторизация в информационных системах,
б) Поиск правонарушителей,
в) Контроль и управление доступом,
г) Управление коллекциями фотографий.

Для улучшения показателей точности разработанной ИНС могут быть приняты следующие меры:

а) Продолжение обучения ИНС;
б) Использование при обучении ИНС иного правила обновления весов (оптимизатора), подбор лучших значений параметров оптимизатора и затухания скорости обучения или иного правила затухания скорости обучения;
в) Использование схемы сравнения областей классов, отличной от попарного сравнения;
г) Дообучение ИНС на других наборах данных, например MS-Celeb-1M;
д) Использование более сложной архитектуры ИНС, например ResNet или Inception-Resnet-v1;
е) Использование меняющегося в процессе обучения параметра R ИНС;
ж) Сжатие ИНС – удаление близких к 0 весов и дообучение ИНС за счёт оптимизации оставшихся.

з) Аугментация обучающего набора данных с использованием порождающих соревновательных ИНС (GANs, Generative Adversarial Networks);

3.6 Выводы

Экспериментальная проверка предложенной функции ошибки на упрощённой задаче классификации показала возможность её использования для обучения извлекающей из изображений идентифицирующие признаки ИНС. Сравнение предложенной функции ошибки с аналогами показало, что при её использовании может быть достигнута большая точность, чем при использовании иных функций ошибки.

Предложенная функция ошибки имеет параметры. Влияние их изменения на точность обученной ИНС было проанализировано методом табличного поиска. С помощью алгоритма Нелдера-Мида найдены оптимальные для упрощённой задачи значения этих параметров.

Разработанная ранее ИНС для выделения идентифицирующих признаков из изображений лиц была обучена с новой функцией ошибки и протестирована. Предложены возможные практические применения этой ИНС и меры для дальнейшего улучшения её показателей точности.

ЗАКЛЮЧЕНИЕ

В результате анализа методов идентификации человека по лицу, было дано обоснование использованию ГСИНС для выделения признаков из изображения лиц.

При разработке ГСИНС, были выбраны среда программной реализации, архитектура, собраны и подготовлены обучающие и тестовые данные, описаны используемые процедуры тестирования.

В результате анализа множества определений функции ошибки для обучения ИНС для идентификации по лицу, было решено разработать новую функцию ошибки. Она была разработана и протестирована на упрощённой задаче классификации изображений. Экспериментально доказано, что точность ИНС, обученной с использованием предложенной функции ошибки, может превосходить аналоги.

Промежуточные результаты разработки новой функции ошибки были представлены на XXIV Международной научно-технической конференции студентов и аспирантов «Радиоэлектроника, Электротехника и Энергетика» [43].

Произведены анализ влияния параметров функции ошибки на точность обученной ИНС и поиск наилучших значений параметров по алгоритму Нелдера-Мида (на упрощённой задаче).

Разработанная ИНС была обучена с использованием предложенной функции ошибки. Поставленная задача была частично решена. Были предложены меры для дальнейшего повышения качества обучения ИНС и перечислены возможные практические приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ Р ИСО/МЭК 19795-1-2007. Автоматическая идентификация. Идентификация биометрическая. Эксплуатационные испытания и протоколы испытаний в биометрии. Часть 1. Принципы и структура [Текст]. – Введ. 2008–12–25. – М.: Стандартинформ, 2009. – 57 с.
2. Liu W. et al. SphereFace: Deep Hypersphere Embedding for Face Recognition [Текст] / W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, L. Song // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Honolulu, 2017. – P. 6738-6746.
3. Kakadiaris I. A. et al. 3D-2D face recognition with pose and illumination normalization [Текст] / I. A. Kakadiaris, G. Toderici, G. Evangelopoulos, G. Passalis, D. Chu, X. Zhao, S. K. Shah, T. Theoharis // Computer Vision and Image Understanding. – 2017. – Vol. 154. – P. 137-151.
4. Zhao R. et al. Fast and precise face alignment and 3d shape reconstruction from a single 2d image [Текст] / R. Zhao, Y. Wang, C. F. Benitez-Quiroz, Y. Liu, A. M. Martinez // Computer Vision – ECCV 2016 Workshops. ECCV 2016. Lecture Notes in Computer Science – Cham, 2016. – Vol. 2. – P. 590-603.
5. Ghiass R. S. et al. Infrared face recognition: A comprehensive review of methodologies and databases [Текст] / R. S. Ghiass, O. Arandjelović, A. Bendada, X. Maldaguea // Pattern Recognition. – 2014. – Vol. 47. – №. 9. – P. 2807-2824.
6. Taigman Y. et al. DeepFace: Closing the gap to human-level performance in face verification [Текст] / Y. Taigman, M. Yang, M. Ranzato, L. Wolf // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Columbus, 2014. – P. 1701-1708.
7. Мищенкова Е. С. Сравнительный анализ алгоритмов распознавания лиц [Текст] / Е. С. Мищенкова // Вестник Волгоградского государственного

университета. Серия 9: Исследования молодых ученых. – 2013. – №. 11. – С. 74-76.

8. Ciregan D., Meier U., Schmidhuber J. Multi-column deep neural networks for image classification [Текст] / D. Ciregan, U. Meier, J. Schmidhuber // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Providence, 2012. – P. 3642-3649.
9. Szegedy C. et al. Going deeper with convolutions [Текст] / C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Boston, 2015. – P. 1-9.
10. Huang G. B. et al. Labeled faces in the wild: A database for studying face recognition in unconstrained environments [Текст] / G.B. Huang, M. Ramesh, T. Berg, E. Learned-Miller. – Amherst, 2007. – P. 3-37. – (Technical Report / University of Massachusetts; № 07-49).
11. Huang G. B., Learned-Miller E. Labeled faces in the wild: Updates and new reporting procedures [Текст] / G. B. Huang, E. Learned-Miller. – Amherst, 2014. – 5 p. – (Technical Report / Dept. Comput. Sci. / University of Massachusetts; № UM-CS-2014-003).
12. Wolf L., Hassner T., Maoz I. Face recognition in unconstrained videos with matched background similarity [Текст] / L. Wolf, T. Hassner, I. Maoz // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Providence, 2011. – P. 529-534.
13. Yi D. et al. Learning face representation from scratch [Электронный ресурс] / D. Yi, Z. Lei, S. Liao, S. Z. Li // arXiv:1411.7923. – Дата обновления: 28.11.2014. – 9 p. – (Prepr. / URL: https://arxiv.org/abs/1411.7923 (Дата обращения: 27.05.2018)).
14. Guo Y. et al. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition [Текст] / Y. Guo, L. Zhang, Y. Hu, X. He, J. Gao // European Conference on Computer Vision (ECCV). – Amsterdam, 2016. – P. 87-102.

15. Liao S. et al. A benchmark study of large-scale unconstrained face recognition [Текст] / S. Liao, Z. Lei, D. Yi, S. Z. Li // IEEE International Joint Conference on Biometrics (IJCB). – Clearwater, 2014. – P. 1-8.
16. Günther M. et al. Toward open-set face recognition [Текст] / M. Günther, S. Cruz, E. M. Rudd, T. E. Boult // IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). – Honolulu, 2017. – P. 573-582.
17. Szegedy C. et al. Rethinking the inception architecture for computer vision [Текст] / C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Las Vegas, 2016. – P. 2818-2826.
18. Zoph B. et al. Learning transferable architectures for scalable image recognition [Электронный ресурс] / B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le // arXiv:1707.07012. – Дата обновления: 21.07.2017. – 14 p. – (Prepr. / URL: https://arxiv.org/abs/1707.07012 (Дата обращения: 27.05.2018)).
19. Suganuma M., Shirakawa S., Nagao T. A genetic programming approach to designing convolutional neural network architectures [Текст] / M. Suganuma, S. Shirakawa, T. Nagao // Genetic and Evolutionary Computation Conference (GECCO). – Berlin, 2017. – P. 497-504.
20. Such F. P. et al. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning [Электронный ресурс] / F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, J. Clune // arXiv:1712.06567. – Дата обновления: 18.12.2017. – 16 p. – (Prepr. / URL: https://arxiv.org/abs/1712.06567 (Дата обращения: 27.05.2018)).
21. He K. et al. Deep residual learning for image recognition [Текст] / K. He, X. Zhang, S. Ren and J. Sun // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Las Vegas, 2016. – P. 770-778.
22. Zhang X. et al. Range loss for deep face recognition with long-tailed training data [Текст] / X. Zhang, Z. Fang, Y. Wen, Z. Li, Y. Qiao // IEEE International Conference on Computer Vision (ICCV). – Venice, 2017. – P. 5409-5418.

23. Ranjan R., Castillo C. D., Chellappa R. L2-constrained softmax loss for discriminative face verification [Электронный ресурс] / R. Ranjan, C. D. Castillo, R. Chellappa // arXiv: 1703.09507. – Дата обновления: 28.03.2017. – 10 p. – (Prepr. / URL: https://arxiv.org/abs/1703.09507 (Дата обращения: 27.05.2018)).
24. Sun Y. et al. Deep learning face representation by joint identification-verification [Текст] / Y. Sun, Y. Chen, X. Wang, X. Tang // Neural Information Processing Systems (NIPS) Conference. – Montreal, 2014. – P. 1988-1996.
25. Wen Y. et al. A discriminative feature learning approach for deep face recognition [Текст] / Y. Wen, K. Zhang, Z. Li, Y. Qiao // European Conference on Computer Vision (ECCV),. – Amsterdam, 2016. – P. 499-515.
26. Liu W. et al. Large-Margin Softmax Loss for Convolutional Neural Networks [Текст] / W. Liu, Y. Wen, Z. Yu, M. Yang // 33rd International Conference on Machine Learning (ICML-2016). – New York, 2016. – Vol. 48. – P. 507-516.
27. Wu Y. et al. Deep Face Recognition with Center Invariant Loss [Текст] / Y. Wu, H. Liu, J. Li, Y. Fu // Thematic Workshops of ACM Multimedia. – Mountain View, 2017. – P. 408-414.
28. Deng J., Zhou Y., Zafeiriou S. Marginal Loss for Deep Face Recognition [Текст] / J. Deng, Y. Zhou, S. Zafeiriou // IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). – Honolulu, 2017. – P. 2006-2014.
29. Wang F. et al. NormFace: L_2 Hypersphere Embedding for Face Verification [Текст] / F. Wang, X. Xiang, J. Cheng, A. L. Yuille // ACM on Multimedia Conference. – Mountain View, 2017. – P. 1041-1049.
30. Qi C., Su F. Contrastive-center loss for deep neural networks [Текст] / C. Qi, F. Su // IEEE International Conference on Image Processing (ICIP). – Beijing, 2017. – P. 2851-2855.
31. Liu Y., Li H., Wang X. Rethinking feature discrimination and polymerization for large-scale recognition [Электронный ресурс] / Y. Liu, H. Li, X. Wang //

arXiv:1710.00870. – Дата обновления: 02.10.2017. – 13 p. – (Prepr. / URL: https://arxiv.org/abs/1710.00870 (Дата обращения: 27.05.2018)).

32. Wang F. et al. Additive Margin Softmax for Face Verification [Электронный ресурс] / F. Wang, W. Liu, H. Liu, J. Cheng // arXiv:1801.05599. – Дата обновления: 17.01.2018. – 7 p. – (Prepr. / URL: https://arxiv.org/abs/1801.05599 (Дата обращения: 27.05.2018)).
33. Qi X., Zhang L. Face Recognition via Centralized Coordinate Learning [Электронный ресурс] / X. Qi, L. Zhang // arXiv:1801.05678. – Дата обновления: 17.01.2018. – 14 p. – (Prepr. / URL: https://arxiv.org/abs/1801.05678 (Дата обращения: 27.05.2018)).
34. Deng J., Guo J., Zafeiriou S. Arcface: Additive angular margin loss for deep face recognition [Электронный ресурс] / J. Deng, J. Guo, S. Zafeiriou // arXiv:1801.07698. – Дата обновления: 23.01.2018. – 13 p. – (Prepr. / URL: https://arxiv.org/abs/1801.07698 (Дата обращения: 27.05.2018)).
35. Hadsell R., Chopra S., LeCun Y. Dimensionality reduction by learning an invariant mapping [Текст] / R. Hadsell, S. Chopra, Y. LeCun // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – New York, 2006. – Vol. 2. – P. 1735-1742.
36. Sun Y., Wang X., Tang X. Deeply learned face representations are sparse, selective, and robust [Текст] / Y. Sun, X. Wang, X. Tang // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Boston, 2015. – P. 2892-2900.
37. Weinberger K. Q., Saul L. K. Distance metric learning for large margin nearest neighbor classification [Текст] / K. Q. Weinberger, L. K. Saul // Journal of Machine Learning Research. – 2009. – № 10. – P. 207-244.
38. Schroff F., Kalenichenko D., Philbin J. Facenet: A unified embedding for face recognition and clustering [Текст] / F. Schroff, D. Kalenichenko, J. Philbin // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Boston, 2015. – P. 815-823.

39. Deng W. et al. Fine-grained face verification: FGLFW database, baselines, and human-DCMN partnership [Текст] / W. Deng, J. Hu, N. Zhang, B. Chen, J. Guo // Pattern Recognition. – 2017. – Vol. 66. – P. 63-73.
40. Zhang N., Deng W. Fine-grained LFW database [Текст] / N. Zhang, W. Deng // International Conference on Biometrics (ICB). – Halmstad, 2016. – P. 1-6.
41. Wang H. et al. CosFace: Large Margin Cosine Loss for Deep Face Recognition [Электронный ресурс] / H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, W. Liu // arXiv:1801.09414. – Дата обновления: 29.01.2018. – 11 p. – (Prepr. / URL: https://arxiv.org/abs/1801.09414 (Дата обращения: 27.05.2018)).
42. LeCun Y. et al. Gradient-based learning applied to document recognition [Текст] / Y. Lecun, L. Bottou, Y. Bengio, P. Haffner // Proceedings of the IEEE. – 1998. – Vol. 86. – №. 11. – P. 2278-2324.
43. Апарнев А. Н., Бартеньев О. В. Разработка метода обучения искусственной нейронной сети для идентификации человека по фотографии лица / А. Н. Апарнев, О. В. Бартеньев // «РАДИОЭЛЕКТРОНИКА, ЭЛЕКТРОТЕХНИКА И ЭНЕРГЕТИКА: Двадцать четвертая Междунар. науч.-техн. конф. студентов и аспирантов»: Тез. докл. конф., Москва, 15–16 марта 2018 г. – М.: ООО «Центр полиграфических услуг „Радуга“», 2018. – С. 554. – ISBN 978-5-905486-08-1.

ПРИЛОЖЕНИЕ А

Программа для обработки входных данных

```
import sys
import argparse
import os
import numpy as np
import threading
import dlib
import cv2
import math


detector = dlib.get_frontal_face_detector()
landmark_predictor =
dlib.shape_predictor(r'dlib_data\shape_predictor_68_face_landmarks.dat')

cv_models = ['haarcascade_frontalface_default.xml',
             'haarcascade_frontalface_alt.xml',
             'haarcascade_frontalface_alt2.xml',
             'haarcascade_frontalface_alt_tree.xml',
             'haarcascade_profileface.xml']
cv_classifiers = [cv2.CascadeClassifier(os.path.join(r'opencv_data', m)) for m in
cv_models]

rects = []


def read_image(folder):
    im = cv2.imread(folder, cv2.IMREAD_COLOR)
    return im


def detect_face(gray):
    '''
    :param im:
    :return: (x,y,w,h)
    '''
    faces = detector(gray, 2)
    t = len(faces) > 0
    minsz = 90

    if len(faces) > 0:
        for f in faces:
            if min(f.right() - f.left(), f.bottom() - f.top()) >= minsz and abs(250 -
f.left() - f.right()) + abs(
                    250 - f.top() - f.bottom()) < 50 and min(f.left(), f.top()) > 0:
                return (f.left(), f.top(), f.right() - f.left(), f.bottom() - f.top()),
t

    for i, classifier in enumerate(cv_classifiers):
        faces = classifier.detectMultiScale(gray,
                                            scaleFactor=1.1,
                                            minNeighbors=1,
                                            minSize=(30, 30),
                                            flags=cv2.CASCADE_SCALE_IMAGE)
        if len(faces) > 0:
```

```
            for det in faces:
                if min(det[2], det[3]) >= minsz and abs(250 - det[0] - (det[0] +
det[2])) + abs(
                        250 - det[1] - (det[1] + det[3])) < 50:
                    return (det[0], det[1], det[2], det[3]), t
    return (), t


def predict_shape(gray, face):
    shape = landmark_predictor(gray, dlib.rectangle(face[0], face[1], face[0] +
face[2], face[1] + face[3]))

    coords = np.empty((68, 2), dtype=int)

    for i in range(68):
        coords[i] = [shape.part(i).x, shape.part(i).y]

    return coords


def make_transform(im, shape, outsz, margin, debug_draw):
    a0 = shape[27]
    b0 = shape[8]
    if debug_draw:
        cv2.circle(im, (int(a0[0]), int(a0[1])), 3, (255, 255, 0), -1)
        cv2.circle(im, (int(b0[0]), int(b0[1])), 3, (0, 0, 255), -1)
    # лицо выравнивается так, что в точке (0.5,0.3) находится переносица, в точке
(0.5,0.9) - подбородок
    # 0 относится к оригиналу изображения, 1 - к изображению после преобразования
    # выполняется перенос, поворот и масштабирование
    a1 = np.asarray([margin + (outsz - 2 * margin) * 0.5, margin + (outsz - 2 * margin)
* 0.3], dtype=float)
    b1 = np.asarray([margin + (outsz - 2 * margin) * 0.5, margin + (outsz - 2 * margin)
* 0.90], dtype=float)
    # перенос расчитывается исходя из центров отрезков
    m0 = (a0 + b0) / 2
    m1 = (a1 + b1) / 2

    d0 = b0 - a0
    d1 = b1 - a1

    e0 = d0 / np.linalg.norm(d0)
    e1 = d1 / np.linalg.norm(d1)
    # угол поворота
    d_angle = -(math.acos(e1[0]) - math.acos(e0[0]))
    dl = np.linalg.norm(d1) / np.linalg.norm(d0)
    # матрица поворота
    lR = dl * np.asarray([[math.cos(d_angle), math.sin(d_angle)],
                          [-math.sin(d_angle), math.cos(d_angle)]], dtype=float)
    # перенос
    ms = m1 - np.matmul(lR, m0)
    # полная матрица преобразования
    mx = np.zeros(shape=[2, 3], dtype=float)
    mx[:, :2] = lR
    mx[:, 2] = ms

    imout = cv2.warpAffine(im, mx, (outsz, outsz), flags=cv2.INTER_CUBIC)
    # cv2.imshow('after',imout)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    return imout
```

```
def process(im, out_sz, margin, debug_draw, defaultBox):
    gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    face, t = detect_face(gray)
    rejected = len(face) == 0
    if rejected:
        face = defaultBox
    if debug_draw:
        if rejected:
            cv2.rectangle(im, (face[0], face[1]), (face[0] + face[2], face[1] +
face[3]), (0, 0, 255), 2)
        else:
            if t:
                cv2.rectangle(im, (face[0], face[1]), (face[0] + face[2], face[1] +
face[3]), (0, 255, 0), 2)
            else:
                cv2.rectangle(im, (face[0], face[1]), (face[0] + face[2], face[1] +
face[3]), (255, 0, 0), 2)
    shape = predict_shape(gray, face)
    if debug_draw:
        for (x, y) in shape:
            cv2.circle(im, (x, y), 3, (0, 255, 255), -1)
    imout = make_transform(im, shape, out_sz, margin, debug_draw)
    return imout


mutex2 = threading.Lock()


def process_subfolder_slave(files, targetdir, output_size, margin, debug_draw,
defaultBox,thread_i):
    for fname in files:
        im = read_image(fname)
        im = process(im, output_size, margin, debug_draw, defaultBox)
        cv2.imwrite(os.path.join(targetdir, os.path.basename(fname)), im)
        print('.', end='', flush=True)


def process_subfolder(srcdir, targetdir, output_size, process_threads, margin,
debug_draw, defaultBox):
    imgtypes = ['jpg', 'png', 'jpeg', 'gif', 'bmp']
    files = [os.path.join(srcdir, fname) for fname in os.listdir(srcdir) if
fname.split('.')[-1] in imgtypes]
    parts = np.array_split(files, min(process_threads, len(files)))
    if len(parts) == 1:
        process_subfolder_slave(parts[0], targetdir, output_size, margin, debug_draw,
defaultBox)
    else:
        threads = [threading.Thread(target=process_subfolder_slave,
                                    args=(parts[i], targetdir, output_size, margin,
debug_draw, defaultBox,i))
                   for i in range(len( parts))]
        for th in threads:
            th.start()
        for th in threads:
            th.join()


def filter(inDir, outDir, outSZ, addMargin, threads, debugDraw, defaultBox):
    global rejects
```

```
    rejects = 0

    dirs = [os.path.join(inDir, f) for f in os.listdir(inDir) if
            os.path.isdir(os.path.join(inDir, f))]

    if not os.path.exists(outDir):
        os.makedirs(outDir)

    for i, subdir in enumerate(dirs):
        targetdir = os.path.join(outDir, os.path.basename(subdir))
        if not os.path.exists(targetdir):
            os.mkdir(targetdir)

        srcdir = os.path.join(inDir, os.path.basename(subdir))
        print('\nprocessing {}({}/{})'.format(subdir, i + 1, len(dirs)), end='',
flush=True)
        process_subfolder(srcdir, targetdir, outSZ, threads, addMargin, debugDraw,
defaultBox)
    fcs = np.asarray(rects)
    print('\n')
    print('done')
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print(rejects)


def main(args):
    filter(args.input_dir, args.output_dir, args.output_size, args.add_margin,
args.process_threads, args.debug_draw,
           [args.default_x, args.default_y, args.default_w, args.default_h])

# default x y w h
# casia
# [58, 68, 133, 133]
# lfw
# [75, 82, 100, 100]

def args_parse(argv):
    parser = argparse.ArgumentParser()
    parser.add_argument('--input_dir', type=str, default=r'D:\datasets\lfw\images',
                        help='')
    # параметры на выходе
    parser.add_argument('--output_size', type=int, default=160,
                        help='размер квадрата с лицом на выходном изображении')
    parser.add_argument('--add_margin', type=int, default=0)
    parser.add_argument('--output_dir', type=str, default=r'D:\datasets\lfw-my-filter',
                        help='')
    # общие параметры
    parser.add_argument('--process_threads', type=int, default=1,
                        help='количество потоков обработки')
    parser.add_argument('--debug_draw', type=bool, default=False,
                        help='вывод отладочной информации на изображения')
    parser.add_argument('--default_x', type=int)
    parser.add_argument('--default_y', type=int)
    parser.add_argument('--default_w', type=int)
    parser.add_argument('--default_h', type=int)
    return parser.parse_args(argv)

if __name__ == '__main__':
    main(args_parse(sys.argv[1:]))
```

ПРИЛОЖЕНИЕ Б

Модуль, реализующий протокол LFW

```
import os
import numpy as np
from sklearn.model_selection import KFold
from scipy import interpolate
import math


def detect_image_format(folder):
    f1 = [os.path.join(folder, f) for f in os.listdir(folder) if
os.path.isdir(os.path.join(folder, f))][0]
    f2 = [f.split('.')[-1] for f in os.listdir(f1) if f.split('.')[-1] in ['jpg',
'png', 'bmp', 'jpeg']][0]
    return f2


def read_pairs(folder, pairs_file):
    with open(pairs_file, 'r') as f:
        flines = f.readlines()
    fformat = detect_image_format(folder)
    parsed = [line.rstrip('\n').split('\t') for line in flines if len(line.split('\t'))
> 2]
    fnames = []
    for line in parsed:
        if len(line) == 3:
            name1 = line[0]
            name2 = name1
            n1 = int(line[1])
            n2 = int(line[2])
            same = True
        else:
            name1 = line[0]
            n1 = int(line[1])
            name2 = line[2]
            n2 = int(line[3])
            same = False
        f1 = os.path.join(folder, name1, '{}_{:0>4}.{}'.format(name1, n1, fformat))
        f2 = os.path.join(folder, name2, '{}_{:0>4}.{}'.format(name2, n2, fformat))
        fnames.append((f1, f2, same))
    return fnames


def eval_accuracy(embs, issame, target_far):
    # обработка исходных данных
    emb_a = embs[:, 0, :]
    emb_b = embs[:, 1, :]
    dists = np.arccos(np.sum(np.multiply(emb_a, emb_b), 1).clip(-1., 1.))
    folds_cnt = 10
    thresholds = np.arange(0, math.pi + 0.01, 0.01)
    # roc - кривая и точность
    tpr, fpr, acc_mean, acc_std = eval_roc(dists, issame, thresholds, folds_cnt)
    # точность при различных частотах ложных допусков far
    thresholds = np.arange(0, math.pi + 0.001, 0.001)
    val_mean, val_std, far = eval_val(dists, issame, target_far, folds_cnt, thresholds)
    return tpr, fpr, acc_mean, acc_std, val_mean, val_std, far
```

```
def eval_acc(dists, issame, threshold):
    '''
    вычисляет точность на тестовой выборке при заданном пороге
    :param dists:
    :param issame:
    :param threshold:
    :return:
    '''
    predict_issame = np.less(dists, threshold)
    tp = np.sum(np.logical_and(predict_issame, issame))
    fp = np.sum(np.logical_and(predict_issame, np.logical_not(issame)))
    tn = np.sum(np.logical_and(np.logical_not(predict_issame), np.logical_not(issame)))
    fn = np.sum(np.logical_and(np.logical_not(predict_issame), issame))

    tpr = 0 if (tp + fn == 0) else (tp / (tp + fn))
    fpr = 0 if (fp + tn == 0) else (fp / (fp + tn))
    acc = (tp + tn) / dists.size
    return tpr, fpr, acc


def eval_roc(dists, issame, thresholds, folds_cnt):
    '''
    вычисляет roc - кривую и точность
    :param dists:
    :param issame:
    :param thresholds:
    :param folds_cnt:
    :return:
    '''
    # разбиение для перекрёстной валидации
    k_fold = KFold(n_splits=folds_cnt, shuffle=False)
    # частоты верно - положительных и ложно - положительных обнаружений
    # на тестовом множестве на каждом разбиении и при каждом пороге
    tps = np.zeros((folds_cnt, thresholds.shape[0]), dtype=float)
    fps = np.zeros((folds_cnt, thresholds.shape[0]), dtype=float)
    # точность на каждом разбиении
    acc = np.zeros((folds_cnt), dtype=float)
    # разбиваемые индексы
    indices = np.arange(0, dists.shape[0], 1, dtype=int)
    # разбиение
    split = k_fold.split(indices)
    for fold_id, (train, test) in enumerate(split):
        # находим лучший порог принятия решения
        # для этого вычисляем точности для всех порогов на обучающем подмножестве
        acc_train = np.zeros((thresholds.shape[0]), dtype=float)
        for threshold_id, threshold in enumerate(thresholds):
            _, _, acc_train[threshold_id] = eval_acc(dists[train], issame[train],
threshold)
        best_threshold_id = np.argmax(acc_train)
        # вычисляем частоты для всех порогов на тестовом множестве
        for threshold_id, threshold in enumerate(thresholds):
            tps[fold_id, threshold_id], fps[fold_id, threshold_id], _ = \
                eval_acc(dists[test], issame[test], threshold)
        # вычисляем точность при лучшем пороге
        _, _, acc[fold_id] = eval_acc(dists[test], issame[test],
thresholds[best_threshold_id])
    # всё усредняется по разбиениям
    tpr = np.mean(tps, 0)
    fpr = np.mean(fps, 0)
```

```
    acc_mean = np.mean(acc)
    acc_std = np.std(acc)
    return tpr, fpr, acc_mean, acc_std


def eval_val(dists, issame, target_far, folds_cnt, thresholds):
    '''
    вычисляет частоту верных допусков при заданной частоте неверных
    :param dists:
    :param issame:
    :param target_far:
    :param folds_cnt:
    :param thresholds:
    :return:
    '''
    # разбиения
    k_fold = KFold(n_splits=folds_cnt, shuffle=False)
    indices = np.arange(0, dists.shape[0], 1, dtype=int)
    split = k_fold.split(indices)
    val = np.zeros(folds_cnt)
    far = np.zeros(folds_cnt)
    for fold_id, (train, test) in enumerate(split):
        # частоты ложных допусков на обучающем множестве при разных порогах
        far_train = np.zeros(thresholds.shape[0])
        for threshold_idx, threshold in enumerate(thresholds):
            _, far_train[threshold_idx] = eval_val_far(dists[train], issame[train],
threshold)
        if np.max(far_train) >= target_far:
            # постройка графика зависимости порога от частоты ложных допусков
            f = interpolate.interp1d(far_train, thresholds, kind='slinear')
            # выбор порога с требуемой частотой
            threshold = f(target_far)
        else:
            # елси порог не найден, выбираем порог, не допускающий допусков
            threshold = 0.0
        # вычисляем частоты на тестовом множестве с полученным порогом
        val[fold_id], far[fold_id] = eval_val_far(dists[test], issame[test], threshold)
    # усредняем по разбиениям
    val_mean = np.mean(val)
    val_std = np.std(val)
    far_mean = np.mean(far)
    return val_mean, val_std, far_mean


def eval_val_far(dist, issame, threshold):
    predicted_same = np.less(dist, threshold)
    # количество верных допусков
    true_accept = np.sum(np.logical_and(predicted_same, issame))
    # количество ложных допусков
    false_accept = np.sum(np.logical_and(predicted_same, np.logical_not(issame)))
    n_same = np.sum(issame)
    n_diff = np.sum(np.logical_not(issame))
    # частота верных допусков
    val = true_accept / n_same  # must be float
    # частота ложных допусков
    far = false_accept / n_diff
    return val, far
```

ПРИЛОЖЕНИЕ В

Модуль, определяющий функцию ошибки

```
import tensorflow as tf


def angles(a, b):
    s = tf.clip_by_value(tf.reduce_sum(tf.multiply(a, b), axis=-1), -1., 1.)
    return tf.acos(s)


def get_centers(features, images_per_class):
    # вычисляем центры для каждой персоны
    with tf.variable_scope('centers_evaluation'):
        features = tf.reshape(features, (-1, images_per_class, features.shape[-1]))
        centers = tf.reduce_mean(features, axis=1, keepdims=False)
        centers = tf.nn.l2_normalize(centers, name='centers')
    return centers


def get_dists(fs, centers):
    # расчитываем расстояния каждого оторбажения до центра соотв. класса
    with tf.variable_scope('distances_to_centers'):
        cs = tf.expand_dims(centers, 1)
        dsts = angles(tf.nn.l2_normalize(fs), cs)
    return dsts


def get_sd(dists):
    '''
    вычисляет стандартные отклонения по известным расстояниям с помощью несмещённой
оценки
    :param dists:
    :return:
    '''
    with tf.variable_scope('standart_deviations'):
        d = tf.sqrt(tf.to_float(tf.subtract(tf.shape(dists)[1], 1)))
        return tf.sqrt(tf.reduce_sum(tf.square(dists), axis=1)) / d


def SphereIntersections(sd1, cs1, sd2, cs2, R, n):
    with tf.name_scope('Intersection'):
        return tf.pow(R * (sd1 + sd2) / angles(cs1, cs2), n) / (n * R)


def get_intersection_by_pairs(sd, centers, R, n):
    sdp = tf.reshape(sd, [-1, 2])
    sd1, sd2 = tf.split(sdp, 2, 1)
    sd1 = tf.reshape(sd1, [-1], name='sdA')
    sd2 = tf.reshape(sd2, [-1], name='sdB')

    csp = tf.reshape(centers, [-1, 2, centers.shape[-1]])
    cs1, cs2 = tf.split(csp, 2, 1)
    cs1 = tf.reshape(cs1, [-1, centers.shape[-1]], name='mA')
    cs2 = tf.reshape(cs2, [-1, centers.shape[-1]], name='mB')
    return SphereIntersections(sd1, cs1, sd2, cs2, R, n)
```

```
def get_sphere_loss(features, images_per_class, R=3., n=6.):
    with tf.variable_scope('my_loss_evaluation'):
        embeddings = tf.nn.l2_normalize(features, axis=-1, name='embeddings')
        # разбираем отображения по персонам
        embs_rs = tf.reshape(embeddings, (-1, images_per_class, embeddings.shape[-1]),
name='embeddings_grouped')
        centers = get_centers(features, images_per_class)
        dists = get_dists(embs_rs, centers)
        # стандартное отклонение (корень из дисперсии)
        sd = get_sd(dists)
        # определение матрицы пересечений
        mx = get_intersection_by_pairs(sd, centers, R, n)
    return mx, sd, centers, embeddings
```

ПРИЛОЖЕНИЕ Г

Модуль, реализующий архитектуру ИНС

```
import tensorflow as tf
import tensorflow.contrib.slim as slim


def l2pool(input, ksz=3, stride=2,name='l2pool'):
    with tf.variable_scope(name):
        net=tf.square(input)
        net=ksz*ksz*slim.avg_pool2d(net,kernel_size=ksz,stride=stride)
        net=tf.sqrt(net)
    return net


def inception(input, c1out, c3red, c3out, c5red, c5out,
              pooltype='max', pool_red=0, out_stride=1, name='inception'):
    branches = []
    with tf.variable_scope(name):
        if c1out > 0 and out_stride == 1:
            with tf.variable_scope('1x1'):
                net=slim.conv2d(input,c1out,kernel_size=1,stride=1,scope='reduce')
                branches.append(net)
        if c3red > 0 and c3out > 0:
            with tf.variable_scope('3x3'):
                net=slim.conv2d(input,c3red,kernel_size=1,stride=1,scope='reduce')
                net=slim.conv2d(net,c3out,kernel_size=3, stride=out_stride,scope='out')
                branches.append(net)
        if c5red > 0 and c5out > 0:
            with tf.variable_scope('5x5'):
                net=slim.conv2d(input,c5red,kernel_size=1, stride=1,scope='reduce')
net=slim.conv2d(net,c5out,kernel_size=5,stride=out_stride,scope='out')
                branches.append(net)
        if pooltype is not None:
            with tf.variable_scope('pool_red'):
                if pooltype == 'max':
                    net = slim.max_pool2d(input,kernel_size=3,
stride=out_stride,scope='pool')
                else:
                    net = l2pool(input, 3, out_stride)
                if pool_red > 0:
                    net = slim.conv2d(net,pool_red,kernel_size=1,stride=1)
                branches.append(net)
        return tf.concat(branches, -1)


def inference(images, keep_probability=1., phase_train=True,
              bottleneck_layer_size=128, weight_decay=0.0,
              reuse=None,use_locrespnorm=False):
        batch_norm_params = {
            # Decay for the moving averages.
            'decay': 0.995,
            # epsilon to prevent 0s in variance.
            'epsilon': 0.001,
            # force in-place updates of mean and variance estimates
            'updates_collections': None,
            # Moving averages ends up in the trainable variables collection
```

```
            'variables_collections': [tf.GraphKeys.TRAINABLE_VARIABLES],
        }

        with slim.arg_scope([slim.conv2d, slim.fully_connected],
                            weights_initializer =
tf.truncated_normal_initializer(stddev=0.1),
                            weights_regularizer = slim.l2_regularizer(weight_decay),
                            normalizer_fn = slim.batch_norm,
                            normalizer_params = batch_norm_params):
            with tf.variable_scope('Inception', 'Inception', [images], reuse=reuse):
                with slim.arg_scope([slim.batch_norm, slim.dropout],
is_training=phase_train):
                    with slim.arg_scope([slim.conv2d, slim.max_pool2d,
slim.avg_pool2d], stride=1, padding='SAME'):
                        #1
                        net = slim.conv2d(images,64,kernel_size=7,
stride=2,scope='Conv1_7x7')
                        net = slim.max_pool2d(net,kernel_size=3,
stride=2,scope='MaxPool1')
                        # 2
                        net = inception(net, 0, 64, 192, 0, 0, None, 0, 1,
'inception2')
                        net = slim.max_pool2d(net,kernel_size=3,
stride=2,scope='MaxPool1')
                        # 3
                        net = inception(net, 64, 96, 128, 16, 32, 'max', 32, 1,
'inception3a')
                        net = inception(net, 64, 96, 128, 32, 64, 'l2', 64, 1,
'inception3b')
                        net = inception(net, 0, 128, 256, 32, 64, 'max', 0, 2,
'inception3c')
                        # 4
                        net = inception(net, 256, 96, 192, 32, 64, 'l2', 128, 1,
'inception4a')
                        net = inception(net, 224, 112, 224, 32, 64, 'l2', 128, 1,
'inception4b')
                        net = inception(net, 192, 128, 256, 32, 64, 'l2', 128, 1,
'inception4c')
                        net = inception(net, 160, 144, 288, 32, 64, 'l2', 128, 1,
'inception4d')
                        net = inception(net, 0, 160, 256, 64, 128, 'max', 0, 2,
'inception4e')
                        # 5
                        net = inception(net, 384, 192, 384, 48, 128, 'l2', 128, 1,
'inception5a')
                        net = inception(net, 384, 192, 384, 48, 128, 'max', 128, 1,
'inception5b')
                        # final
                        net = tf.reduce_mean(net, axis=[1, 2], keepdims=False,
name='final_avg_pool')

net=slim.fully_connected(net,bottleneck_layer_size,None,scope='Bottleneck',
reuse=False,)
                        return net
```

ПРИЛОЖЕНИЕ Д

Программа для выполнения табличного поиска

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

from datetime import datetime

import tensorflow as tf
import sphere_loss_v36 as sphere_loss
import scipy.optimize as opt
import numpy as np

from tensorflow.examples.tutorials.mnist import input_data

FLAGS = None


def conv2d(x, W, b):
    return tf.nn.relu(tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME') + b)


def max_pool(x, size=2, stride=2):
    """max_pool_2x2 downsamples a feature map by 2X."""
    return tf.nn.max_pool(x, ksize=[1, size, size, 1], strides=[1, stride, stride, 1],
padding='SAME')


def weight_variable(shape):
    """weight_variable generates a weight variable of a given shape."""
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)


def bias_variable(shape):
    """bias_variable generates a bias variable of a given shape."""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)


def variable_summaries(var, name):
    """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
    with tf.name_scope(name):
        mean = tf.reduce_mean(var)
        tf.summary.scalar('mean', mean)
        # with tf.name_scope('stddev'):
        #    stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
        # tf.summary.scalar('stddev', stddev)
        tf.summary.scalar('max', tf.reduce_max(var))
        tf.summary.scalar('min', tf.reduce_min(var))
        # tf.summary.histogram('histogram', var)


def network(x, features_sz):
    with tf.name_scope('network'):
        with tf.name_scope('reshape'):
```

```
            x_image = tf.reshape(x, [-1, 28, 28, 1])
            # tf.summary.image('input', x_image, 10)

        # First convolutional layer - maps one grayscale image to 32 feature maps.
        with tf.name_scope('conv1'):
            W_conv1 = weight_variable([5, 5, 1, 32])
            variable_summaries(W_conv1, 'W_conv1')
            b_conv1 = bias_variable([32])
            variable_summaries(b_conv1, 'b_conv1')
            h_conv1 = conv2d(x_image, W_conv1, b_conv1)

            # Pooling layer - downsamples by 2X.
        with tf.name_scope('pool1'):
            h_pool1 = max_pool(h_conv1)

        # Second convolutional layer -- maps 32 feature maps to 64.
        with tf.name_scope('conv2'):
            W_conv2 = weight_variable([5, 5, 32, 64])
            variable_summaries(W_conv2, 'W_conv2')
            b_conv2 = bias_variable([64])
            variable_summaries(b_conv2, 'b_conv2')
            h_conv2 = conv2d(h_pool1, W_conv2, b_conv2)

            # Second pooling layer.
        with tf.name_scope('pool2'):
            h_pool2 = max_pool(h_conv2)

        with tf.name_scope('fc1'):
            W_fc1 = weight_variable([7 * 7 * 64, 1024])
            variable_summaries(W_fc1, 'W_fc1')
            b_fc1 = bias_variable([1024])
            variable_summaries(b_fc1, 'b_fc1')

            h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
            h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

        with tf.name_scope('fc2'):
            W_fc2 = weight_variable([1024, features_sz])
            variable_summaries(W_fc2, 'W_fc2')
            b_fc2 = bias_variable([features_sz])
            variable_summaries(b_fc2, 'b_fc2')

            out = tf.matmul(h_fc1, W_fc2) + b_fc2

    return tf.identity(out, name='features')


def main(iters, features_sz, lr, R, n, output=True):
    try:
        batch_sz = 50
        images_per_class = batch_sz // 10

        mnist = input_data.read_data_sets('/tmp/tensorflow/mnist/input_data')
        tf.reset_default_graph()
        with tf.Session() as sess:

            x = tf.placeholder(tf.float32, [None, 784])
            y_ = tf.placeholder(tf.int64, [None])

            features = network(x, features_sz)
```

```
            mx, sd, centers, embeddings = sphere_loss.get_sphere_loss(features,
images_per_class, R, n)
            with tf.name_scope('classifier'):
                centers_var = weight_variable([features_sz, 10])
                variable_summaries(centers_var, 'W_fc2')
                variable_summaries(centers, 'centers_values')
                y_conv = tf.matmul(features, centers_var)
            with tf.name_scope('loss'):
                loss = tf.reduce_mean(mx)
                sm_loss = tf.losses.sparse_softmax_cross_entropy(labels=y_,
logits=y_conv)
            with tf.name_scope('optimizer'):
                train_net = tf.train.AdamOptimizer(learning_rate=lr).minimize(loss)
                train_classifier =
tf.train.AdamOptimizer(learning_rate=lr).minimize(sm_loss, var_list=[centers_var])
            with tf.name_scope('accuracy'):
                correct_prediction = tf.equal(tf.argmax(y_conv, 1), y_)
                correct_prediction = tf.cast(correct_prediction, tf.float32)
            accuracy = tf.reduce_mean(correct_prediction)
            with tf.name_scope('stats'):
                tf.summary.scalar('loss', loss)
                tf.summary.scalar('train_accuracy', accuracy)
            logdir = r'D:\models\mnist_conv_myloss_v3_6_grid_search-' +
datetime.strftime(datetime.now(), '%Y%m%d-%H%M%S') + '_features=' + str(
                features_sz) + '_R=' + str(R) + '_n=' + str(n) + '_lr=' + str(lr)
            if output:
                print('features=' + str(
                    features_sz) + ' R=' + str(R) + ' n=' + str(n) + ' lr=' + str(lr))
            def unison_shuffled_copies(a, b):
                assert len(a) == len(b)
                p = np.random.permutation(len(a))
                return a[p], b[p]
            def get_batch(training, unordered=True, size=50):
                if not training:
                    return mnist.test.images, mnist.test.labels
                else:
                    if unordered:
                        b = mnist.train.next_batch(size)
                        return b[0], b[1]
                    else:
                        sz = 0
                        imgs = []
                        lbls = []
                        for i in range(10):
                            imgs.append([])
                            lbls.append([])
                        images_p_class = size // 10
                        while sz < images_p_class * 10:
                            b = mnist.train.next_batch(1)
                            if len(imgs[b[1][0]]) < images_p_class:
                                imgs[b[1][0]].append(b[0][0])
                                lbls[b[1][0]].append(b[1][0])
                                sz = sz + 1
                        res_im, res_lb = np.asarray(imgs), np.asarray(lbls)
                        res_im, res_lb = unison_shuffled_copies(res_im, res_lb)
                        return res_im.reshape([-1, 784]), res_lb.reshape([-1])

            merged = tf.summary.merge_all()
            train_writer = tf.summary.FileWriter(logdir, sess.graph)
            sess.run(tf.global_variables_initializer())
            for i in range(iters):
```

```
                if i % 1000 == 0:
                    tbat = get_batch(False)
                    imgs = np.split(tbat[0], 200, 0)
                    lbls = np.split(tbat[1], 200, 0)
                    acc = 0.
                    for j in range(200):
                        acc = acc + sess.run(accuracy, feed_dict={x: imgs[j], y_:
lbls[j]})
                    if output:
                        print('test accuracy %g' % (acc / 200))
                    s = tf.Summary()
                    s.value.add(tag='stats/test_accuracy', simple_value=acc / 200)
                    train_writer.add_summary(s, i)
                batch = get_batch(True, False, batch_sz)
                if i % 100 == 0:
                    train_accuracy, summ = sess.run([accuracy, merged], feed_dict={
                        x: batch[0], y_: batch[1]})
                    if output:
                        print('step %d, training accuracy %g' % (i, train_accuracy))
                    train_writer.add_summary(summ, i)
                sess.run([train_net], feed_dict={x: batch[0], y_: batch[1]})
                sess.run([train_classifier], feed_dict={x: batch[0], y_: batch[1]})
            tbat = get_batch(False)
            imgs = np.split(tbat[0], 200, 0)
            lbls = np.split(tbat[1], 200, 0)
            acc = 0.
            for j in range(200):
                acc = acc + sess.run(accuracy, feed_dict={x: imgs[j], y_: lbls[j]})
            if output:
                print('test accuracy %g' % (acc / 200))
            s = tf.Summary()
            s.value.add(tag='stats/test_accuracy', simple_value=acc / 200)
            train_writer.add_summary(s, iters)
            train_writer.close()
        return acc / 200
    except:
        return 0.

if __name__ == '__main__':
    iters = 50000
    Rs = [1, 2, 3, 4, 5]
    ns =range(2,20,1)
    k = 3
    res = np.zeros([len(Rs), len(ns), k], dtype=float)
    for i, R in enumerate(Rs):
        for j, n in enumerate(ns):
            for l in range(k):
                res[i, j, l] = main(iters, 16, 1e-4, R, n, False)
    with open(r'D:\models\grid-results.txt','w') as f:
        f.write('res\n')
        f.write(str(res))
        f.write('\navg:\n')
        f.write(str(np.average(res, 2)))
    print('done')
```

ПРИЛОЖЕНИЕ Е

Программа для оптимизации параметров по алгоритму Нелдера-Мида

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

from datetime import datetime

import tensorflow as tf
import sphere_loss_v36 as sphere_loss
import scipy.optimize as opt
import numpy as np
import random

from tensorflow.examples.tutorials.mnist import input_data

FLAGS = None


def conv2d(x, W, b):
    return tf.nn.relu(tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME') + b)


def max_pool(x, size=2, stride=2):
    """max_pool_2x2 downsamples a feature map by 2X."""
    return tf.nn.max_pool(x, ksize=[1, size, size, 1], strides=[1, stride, stride, 1],
padding='SAME')


def weight_variable(shape, seed=None):
    """weight_variable generates a weight variable of a given shape."""
    initial = tf.truncated_normal(shape, stddev=0.1, seed=seed)
    return tf.Variable(initial)


def bias_variable(shape):
    """bias_variable generates a bias variable of a given shape."""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)


def network(x, features_sz, seed=None):
    with tf.name_scope('network'):
        with tf.name_scope('reshape'):
            x_image = tf.reshape(x, [-1, 28, 28, 1])
        with tf.name_scope('conv1'):
            W_conv1 = weight_variable([5, 5, 1, 32], seed)
            b_conv1 = bias_variable([32])
            h_conv1 = conv2d(x_image, W_conv1, b_conv1)
        with tf.name_scope('pool1'):
            h_pool1 = max_pool(h_conv1)
        with tf.name_scope('conv2'):
            W_conv2 = weight_variable([5, 5, 32, 64], seed)
            b_conv2 = bias_variable([64])
            h_conv2 = conv2d(h_pool1, W_conv2, b_conv2)
        with tf.name_scope('pool2'):
```

```
            h_pool2 = max_pool(h_conv2)
        with tf.name_scope('fc1'):
            W_fc1 = weight_variable([7 * 7 * 64, 1024], seed)
            b_fc1 = bias_variable([1024])
            h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
            h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
        with tf.name_scope('fc2'):
            W_fc2 = weight_variable([1024, features_sz], seed)
            b_fc2 = bias_variable([features_sz])
            out = tf.matmul(h_fc1, W_fc2) + b_fc2
    return tf.identity(out, name='features')


def main(iters, features_sz, lr, R, n, output=True, seed=137):
    try:
        if n < 1:
            return 0.
        if R < 0.5:
            return 0.

        if seed is not None:
            random.seed(seed)
            np.random.seed(seed)

        print('began training n={}, R={}, seed={}'.format(n,R,seed))

        tf.reset_default_graph()
        if seed is not None:
            tf.set_random_seed(seed)
        batch_sz = 50
        images_per_class = batch_sz // 10

        mnist = input_data.read_data_sets('/tmp/tensorflow/mnist/input_data')
        with tf.Session() as sess:
            x = tf.placeholder(tf.float32, [None, 784])
            y_ = tf.placeholder(tf.int64, [None])
            features = network(x, features_sz, seed)
            mx, sd, centers, embeddings = sphere_loss.get_sphere_loss(features,
images_per_class, R, n)
            with tf.name_scope('classifier'):
                centers_var = weight_variable([features_sz, 10], seed)
                y_conv = tf.matmul(features, centers_var)
            with tf.name_scope('loss'):
                loss = tf.reduce_mean(mx)
                sm_loss = tf.losses.sparse_softmax_cross_entropy(labels=y_,
logits=y_conv)
            with tf.name_scope('optimizer'):
                train_net = tf.train.AdamOptimizer(learning_rate=lr).minimize(loss)
                train_classifier =
tf.train.AdamOptimizer(learning_rate=lr).minimize(sm_loss, var_list=[centers_var])
            with tf.name_scope('accuracy'):
                correct_prediction = tf.equal(tf.argmax(y_conv, 1), y_)
                correct_prediction = tf.cast(correct_prediction, tf.float32)
            accuracy = tf.reduce_mean(correct_prediction)
            with tf.name_scope('stats'):
                tf.summary.scalar('loss', loss)
                tf.summary.scalar('train_accuracy', accuracy)

            logdir = r'C:\models\mnist_conv_myloss_v3_6_nelder_mead-' +
datetime.strftime(datetime.now(), '%Y%m%d-%H%M%S') + '_features=' + str(
features_sz) + '_R=' + str(R) + '_n=' + str(n) + '_lr=' + str(lr)
```

```
            if output:
                print('features=' + str(
                    features_sz) + ' R=' + str(R) + ' n=' + str(n) + ' lr=' + str(lr))

            def unison_shuffled_copies(a, b, seed=None):
                assert len(a) == len(b)
                if seed is not None:
                    random.seed(seed)
                    np.random.seed(seed)
                p = np.random.permutation(len(a))
                return a[p], b[p]

            def get_batch(training, unordered=True, size=50, seed=None):
                if not training:
                    return mnist.test.images, mnist.test.labels
                else:
                    if seed is not None:
                        random.seed(seed)
                        np.random.seed(seed)
                    if unordered:
                        b = mnist.train.next_batch(size)
                        return b[0], b[1]
                    else:
                        sz = 0
                        imgs = []
                        lbls = []
                        for i in range(10):
                            imgs.append([])
                            lbls.append([])
                        images_p_class = size // 10
                        while sz < images_p_class * 10:
                            b = mnist.train.next_batch(1)
                            if len(imgs[b[1][0]]) < images_p_class:
                                imgs[b[1][0]].append(b[0][0])
                                lbls[b[1][0]].append(b[1][0])
                                sz = sz + 1
                        res_im, res_lb = np.asarray(imgs), np.asarray(lbls)
                        res_im, res_lb = unison_shuffled_copies(res_im, res_lb, seed)
                        return res_im.reshape([-1, 784]), res_lb.reshape([-1])

            merged = tf.summary.merge_all()
            train_writer = tf.summary.FileWriter(logdir, sess.graph)
            sess.run(tf.global_variables_initializer())
            for i in range(iters):
                if i % 1000 == 0:
                    tbat = get_batch(False)
                    imgs = np.split(tbat[0], 200, 0)
                    lbls = np.split(tbat[1], 200, 0)
                    acc = 0.
                    for j in range(200):
                        acc = acc + sess.run(accuracy, feed_dict={x: imgs[j], y_:
lbls[j]})
                    if output:
                        print('test accuracy %g' % (acc / 200))
                    s = tf.Summary()
                    s.value.add(tag='stats/test_accuracy', simple_value=acc / 200)
                    train_writer.add_summary(s, i)
                batch = get_batch(True, False, batch_sz, seed=seed + i)
                if i % 100 == 0:
```

```
                    train_accuracy, summ = sess.run([accuracy, merged], feed_dict={x:
batch[0], y_: batch[1]})
                    if output:
                        print('step %d, training accuracy %g' % (i, train_accuracy))
                    train_writer.add_summary(summ, i)
                sess.run([train_net], feed_dict={x: batch[0], y_: batch[1]})
                sess.run([train_classifier], feed_dict={x: batch[0], y_: batch[1]})
            tbat = get_batch(False)
            imgs = np.split(tbat[0], 200, 0)
            lbls = np.split(tbat[1], 200, 0)
            acc = 0.
            for j in range(200):
                acc = acc + sess.run(accuracy, feed_dict={x: imgs[j], y_: lbls[j]})
            if output:
                print('test accuracy %g' % (acc / 200))
            s = tf.Summary()
            s.value.add(tag='stats/test_accuracy', simple_value=acc / 200)
            train_writer.add_summary(s, iters)
            train_writer.close()
        print('Done training, lr={}, iters={}, R={},n={}, accuracy={:.4f}'.format(lr,
iters, R, n, acc / 200))
        return acc / 200
    except Exception as exc:
        print('exception!!!\n')
        return -1


if __name__ == '__main__':
    iters = 50000
    start_simplex = np.asarray([[4, 14], [1, 17], [2, 12]], dtype=float)
    with open(r'C:\models\search_res.csv', mode='w') as f:
        f.write('iters,R,n,accuracy\n')
    def fun(x):
        acc = main(iters, 16, 1e-4, x[0], x[1], False,3)
        with open(r'C:\models\search_res.csv', mode='a') as f:
            f.write('{},{},{},{}\n'.format(iters, x[0], x[1], acc))
        return 1. - acc
    res = opt.minimize(fun, method='Nelder-Mead', x0=start_simplex[0],
                       options={  # 'func': f,
                           'maxiter': None,  # 500,
                           'maxfev': 280,
                           'disp': True,
                           'return_all': True,
                           'initial_simplex': start_simplex,
                           'xtol': 0.3,
                           'ftol': 0.00005})
    print(res)
    print('done')
```

ПРИЛОЖЕНИЕ Ж

Программа для обучения ИНС для выделения идентифицирующих признаков из изображений лиц

```
import argparse
import sys
import os
import random
from datetime import datetime
import numpy as np
import tensorflow as tf
import importlib
from timer import timer
import LFW
import sphere_loss_v36 as sphere_loss
import math
import BLUFR
import matplotlib.pyplot as plt
import nmodels.facenet_inception as net


def read_train_set(folder, minfilter=1):
    names = [name for name in os.listdir(folder) if os.path.isdir(os.path.join(folder,
name))]
    names.sort()
    res = []
    img_formats = ['jpg', 'jpeg', 'png', 'bmp']
    i = 0
    for name in names:
        ls = [os.path.join(folder, name, fname) for fname in
os.listdir(os.path.join(folder, name)) \
              if os.path.isfile(os.path.join(folder, name, fname)) and
fname.split('.')[-1] in img_formats]
        if len(ls) >= minfilter:
            res.append((name, ls))
        # if i % 100 == 0:
        #     print('.', end='', flush=True)
        i = i + 1
    return res


def save_model(sess, saver, summary_writer, model_dir, step, global_step):
    # Save the model checkpoint
    print('Saving variables')
    checkpoint_path = os.path.join(model_dir, 'model.ckpt')
    saver.save(sess, checkpoint_path, global_step=global_step, write_meta_graph=False)
    metagraph_filename = os.path.join(model_dir, 'model.meta')
    if not os.path.exists(metagraph_filename):
        print('Saving metagraph')
        saver.export_meta_graph(metagraph_filename)


def augment(image, use_random_crop, add_noise, use_random_flip, image_sz):
    if use_random_crop:
        image = tf.random_crop(image, (image_sz, image_sz, 3))
    else:
```

```
        mx = (image.shape[1] - image_sz) // 2
        my = (image.shape[0] - image_sz) // 2
        image = tf.image.crop_to_bounding_box(image, my, mx, image_sz, image_sz)
    if add_noise:
        image = image + tf.random_uniform(tf.shape(image), -10, 10, dtype=tf.int32)
    if use_random_flip:
        image = tf.image.random_flip_left_right(image)
    image.set_shape((image_sz, image_sz, 3))
    return image


def sdiv(a, b):
    return int(math.ceil(a / b))


def select_images(sess, epoch_sz, train_data, enque_op, embeddings, loss,
image_patches, labels_out,
                  persons_per_batch, images_per_person):
    '''
    :return: Список списков имён файлов изображений
    '''
    subset = random.sample(train_data, epoch_sz * persons_per_batch)
    subset = np.asarray([random.sample(p[1], images_per_person) for p in subset],
dtype=str)
    return subset


def test(test_pairs, enque_op, embeddings, image_patches, labels, labels_out,
batch_size,
         images_per_person, sess, embedding_size, phase_train_placeholder,
persons_per_batch, summary_op,
         summary_writer, write_summary, global_step, log_dir, plot=False):
    print('Running LFW verification test protocol...')
    fnames = []
    issame = np.empty(shape=(len(test_pairs)), dtype=bool)
    for i, t in enumerate(test_pairs):
        fnames.append([t[0], t[1]])
        issame[i] = t[2]
    fnames = np.asarray(fnames)
    if_fnames_sz = batch_size * (fnames.size // batch_size + (1 if fnames.size %
batch_size > 0 else 0))
    appendix = if_fnames_sz - fnames.size
    lin = np.reshape(fnames, (-1))
    if appendix > 0:
        lin = np.concatenate((lin, np.full((appendix), lin[0])))
    fnames_in = np.reshape(lin, (-1, images_per_person))
    labesl_in = np.arange(0, fnames_in.size, 1, int).reshape(fnames_in.shape)
    embs = np.full((lin.shape[0], embedding_size), -100.)
    fd = {
        image_patches: fnames_in,
        labels: labesl_in,
        phase_train_placeholder: False
    }
    sess.run(enque_op, feed_dict=fd)
    cntr = lin.shape[0]
    fd = {
        phase_train_placeholder: False}
    while cntr > 0:
        emb_vals, label_vals = sess.run([embeddings, labels_out], feed_dict=fd)
        embs[label_vals] = emb_vals
        cntr = max(0, cntr - len(label_vals))
```

```
    embs = embs[0:fnames.size, :]
    embs = embs.reshape((embs.shape[0] // 2, 2, embs.shape[1]))
    tpr, fpr, acc_mean, acc_std, val_mean, val_std, far = LFW.eval_accuracy(embs,
issame, 0.001)
    print("accuracy = {:.3f}+-{:.3f}, validation rate = {:.3f}+-{:.3f} at
FAR={}".format(acc_mean, acc_std, val_mean, val_std, far))
    if write_summary:
        summary = tf.Summary()
        summary.value.add(tag='lfw/accuracy', simple_value=acc_mean)
        summary.value.add(tag='lfw/val_rate', simple_value=val_mean)
        summary_writer.add_summary(summary, sess.run(global_step))
        with open(os.path.join(log_dir, 'lfw_result.txt'), 'at') as f:
            f.write('%d\t%.5f\t%.5f\n' % (sess.run(global_step), acc_mean, val_mean))

    if plot:
        plt.gcf().clear()
        plt.interactive(False)
        plt.title('ROC on LFW')
        plt.plot(fpr, tpr, 'b', label='AUC = %0.2f' % np.trapz(tpr, fpr))
        plt.legend(loc='lower right')
        plt.plot([0, 1], [0, 1], 'r--')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.savefig(os.path.join(log_dir,'LFW_ROC_gs=%d.png'%sess.run(global_step)))


def get_train_op(total_loss, learning_rate_ph, global_step, epoch_sz, lr_decay_epoch,
learning_rate_decay_factor):
    decayed_learning_rate = tf.train.exponential_decay(learning_rate_ph, global_step,
epoch_sz * lr_decay_epoch, learning_rate_decay_factor)
    summ = tf.summary.scalar('learning_rate', decayed_learning_rate)
    opt = tf.train.GradientDescentOptimizer(decayed_learning_rate
    minop = opt.minimize(total_loss, global_step, tf.global_variables(),
name='optimization')
    return minop, decayed_learning_rate


def get_total_loss(loss):
    regloss = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
    total_loss = tf.add_n([loss] + regloss, name='total_loss')
    return total_loss


def InitPipeline(image_sz, images_per_person, image_patches, labels, batch_size,
keep_prob, phase_train_placeholder, weight_decay, embedding_size, R, n,
use_random_crop, add_noise, use_random_flip, network):
    with tf.name_scope('input_pipeline'):
        input_queue = tf.FIFOQueue(capacity=15000,
                                   dtypes=[tf.string, tf.int32],
                                   shapes=[(images_per_person,), (images_per_person,)],
name='input_queue')
        enque_op = input_queue.enqueue_many(vals=[image_patches, labels])
        input_sz = input_queue.size()
        images_and_labels = []
        preprocess_threads = 4
        for _ in range(preprocess_threads):
            fnames, labels = input_queue.dequeue()
            images = []
            for fname in tf.unstack(fnames):
```

```
                file = tf.read_file(fname)
                img = tf.image.decode_image(file, channels=3)
                img = augment(img, use_random_crop, add_noise, use_random_flip,
image_sz)
                img = tf.image.per_image_standardization(img)
                images.append(img)
            images_and_labels.append([images, labels])

        images_batch, labels_out = tf.train.batch_join(images_and_labels,
                    batch_size=batch_size,
                    shapes=[(image_sz, image_sz, 3), ()],
                    enqueue_many=True,
                    capacity=batch_size * preprocess_threads,
                    allow_smaller_final_batch=True)

    images_batch = tf.identity(images_batch, 'image_batch')
    labels_out = tf.identity(labels_out, 'label_batch')

    features = net.inference(images_batch, keep_prob,
                             phase_train=phase_train_placeholder,
                             bottleneck_layer_size=embedding_size,
                             weight_decay=weight_decay)
    features = tf.identity(features, 'features')
    mx, sd, centers, embeddings = sphere_loss.get_sphere_loss(features,
images_per_person, R=R, n=n)  # m=m, l=l)
    loss = tf.reduce_mean(mx, name='my_loss')
    total_loss = get_total_loss(loss)
    return enque_op, loss, embeddings, labels_out, total_loss, input_sz


def train_epoch(sess, train_data, enque_op, embeddings, loss, total_loss, epoch,
image_patches, labels_in, labels_out, phase_train_placeholder,
learning_rate_placeholder, lr, batch_sz, train_op, epoch_sz, images_per_person,
persons_per_batch, global_step, input_sz, summary_op, summary_writer):
    '''
                Одна эпоха обучения ИНС.
    '''
    train_images = select_images(sess, epoch_sz, train_data, enque_op, embeddings,
loss, image_patches, labels_out, persons_per_batch, images_per_person)
    labels_inval = np.arange(0, train_images.size, 1,
dtype=int).reshape(train_images.shape)
    fd = {image_patches: train_images,
          labels_in: labels_inval,
          phase_train_placeholder: True}
    sess.run(enque_op, feed_dict=fd)
    train_persons_count = train_images.shape[0]
    fd = {phase_train_placeholder: True,
          learning_rate_placeholder: lr}
    # i=0
    batches_per_epoch = train_persons_count // persons_per_batch + (
        1 if train_persons_count % persons_per_batch > 0 else 0)
    # while train_persons_count>0:
    for i in range(batches_per_epoch):
        with timer() as t:
            epoch_v, _, total_loss_v, loss_v, labels_v, embs_v, gs = sess.run(
                [epoch, train_op, total_loss, loss, labels_out, embeddings,
global_step], feed_dict=fd)
            print('epoch {:6d}, batch {:4d}/{}, global_step {:10d}, loss ={:10.3f},
total_loss ={:10.3f}, t={:7.3f}s'  # , in_sz={}, embs_v_mean_.max={:.8f},
embs_v_std_.min={:.8f}'.format(epoch_v + 1, i + 1, batches_per_epoch, gs, loss_v,
total_loss_v, t.now()))
```

```
            if i % 5 == 0:
                summary = tf.Summary()
                summary.value.add(tag='train/loss', simple_value=loss_v)
                summary.value.add(tag='train/total_loss', simple_value = total_loss_v)
                summary_writer.add_summary(summary, gs)
            i = i + 1
            train_persons_count = max(0, train_persons_count - persons_per_batch)


def write_arguments_to_file(args, filename):
    with open(filename, 'w') as f:
        for key, value in args.__dict__.items():
            f.write('%s: %s\n' % (key, str(value)))


def eval_embeddings_and_save_for_BLUFR(file_list, enque_op, embeddings, image_patches,
labels, labels_out, batch_size, images_per_person, sess, embedding_size,
phase_train_placeholder, BLUFR_dir, global_step):
    print('evaluating embeddings for BLUFR')
    fnames = file_list
    fnames = np.asarray(fnames)
    if_fnames_sz = batch_size * (fnames.size // batch_size + (1 if fnames.size %
batch_size > 0 else 0))
    appendix = if_fnames_sz - fnames.size
    lin = np.reshape(fnames, (-1))
    if appendix > 0:
        lin = np.concatenate((lin, np.full((appendix), lin[0])))
    fnames_in = np.reshape(lin, (-1, images_per_person))
    labesl_in = np.arange(0, fnames_in.size, 1, int).reshape(fnames_in.shape)
    embs = np.full((lin.shape[0], embedding_size), -100.)
    fd = {
        image_patches: fnames_in,
        labels: labesl_in,
        phase_train_placeholder: False
    }
    sess.run(enque_op, feed_dict=fd)
    cntr = lin.shape[0]
    fd = {
        phase_train_placeholder: False}
    while cntr > 0:
        emb_vals, label_vals = sess.run([embeddings, labels_out], feed_dict=fd)
        embs[label_vals] = emb_vals
        cntr = max(0, cntr - len(label_vals))
        # print('.',end='',flush=True)
    embs = embs[0:fnames.size, :]
    print('\nSaving embeddings...', end='')
    BLUFR.SaveEmbeddings(os.path.join(BLUFR_dir, str(format(sess.run(global_step)))),
embs)
    print('done')


def main(args):
    # параметры пакетов
    images_per_person = args.images_per_person
    persons_per_batch = args.persons_per_batch
    batch_size = persons_per_batch * images_per_person
    persons_per_epoch = args.persons_per_epoch
    epoch_sz = persons_per_epoch // persons_per_batch  # batches per epoch
    epoch_count = args.epoch_count  # 5000
    test_rate = args.test_rate
    validation = args.validation
```

```
    # размеры входа и выхода сети
    image_sz = args.image_sz
    embedding_size = args.embedding_size
    # аугментация
    use_random_flip = args.use_random_flip
    use_random_crop = args.use_random_crop
    add_noise = args.add_noise
    # параметры
    R = args.R
    n = args.n
    # регуляризация, скорость обучения
    lr_decay_epoch = 20
    learning_rate_decay_factor = 0.9
    learning_rate = 0.1
    keep_prob = 0.8
    weight_decay = 1e-4
    # директории
    # model_module = args.model_module
    train_set_dir = args.train_set_dir
    lfw_dir = args.lfw_dir
    lfw_pairs_file = args.lfw_pairs_file
    blufr_list_file = args.blufr_list_file
    continue_training = args.continue_training
    if continue_training != '':
        dir = continue_training[:continue_training.rfind("\\")-6]
    else:
        subdir = datetime.strftime(datetime.now(), 'face_my_loss_GD-%Y%m%d-%H%M%S')
        dir = os.path.join(r'D:\models', subdir)
    print('model dir {}'.format(dir))
    gpu_memory_fraction = 0.9
    save_dir = os.path.join(dir, 'model')
    if not os.path.isdir(save_dir):
        os.makedirs(save_dir)
    log_dir = os.path.join(dir, 'logs')
    if not os.path.isdir(log_dir):
        os.makedirs(log_dir)
    seed = 123
    random.seed(seed)
    np.random.seed(seed)
    write_arguments_to_file(args, os.path.join(log_dir, 'arguments.txt'))

    # build model
    with tf.Graph().as_default():
        # tf.set_random_seed(seed)
        # placeholders
        image_patches = tf.placeholder(dtype=tf.string, shape=[None,
images_per_person], name='input_filenames')
        labels = tf.placeholder(dtype=tf.int32, shape=[None, images_per_person],
name='labels')
        phase_train_placeholder = tf.placeholder(dtype=tf.bool, name='training')
        learning_rate_placeholder = tf.placeholder(dtype=tf.float32,
name='learning_rate_placeholder')
        # vars
        global_step = tf.Variable(0, trainable=False, name='global_step')
        epoch = global_step // epoch_sz
        # pipeline
        # network =  importlib.import_module(model_module)
        enque_op, loss, embeddings, labels_out, total_loss, input_sz =
InitPipeline(image_sz, images_per_person, image_patches,
             labels, batch_size, keep_prob,
             phase_train_placeholder,
```

```
            weight_decay, embedding_size, R, n,
            use_random_crop, add_noise,
            use_random_flip, net)
        train_op, decayed_lr = get_train_op(total_loss, learning_rate_placeholder,
global_step, epoch_sz, lr_decay_epoch, learning_rate_decay_factor)
        saver = tf.train.Saver(tf.trainable_variables(), max_to_keep=5)
        summary_op = tf.summary.merge_all()
        gpu_options =
tf.GPUOptions(per_process_gpu_memory_fraction=gpu_memory_fraction)
        with tf.Session(config = tf.ConfigProto(gpu_options =
gpu_options)).as_default() as sess:
            summary_writer = tf.summary.FileWriter(log_dir, sess.graph)
            with timer() as t:
                print('init variables...', end='')
                sess.run(tf.global_variables_initializer(),
feed_dict={phase_train_placeholder: True})
                sess.run(tf.local_variables_initializer(),
feed_dict={phase_train_placeholder: True})
                print('done in {} s'.format(t.now()))
                t.restart()
                print('start quere runners...', end='')
                tf.train.start_queue_runners(sess, coord=tf.train.Coordinator())
                print('done in {} s'.format(t.now()))
            with timer() as t:
                print('reading training data')
                train_data = read_train_set(train_set_dir, images_per_person)
                print('\ndone in {} s'.format(t.now()))
            with timer() as t:
                print('reading LFW pairs')
                lfw_pairs = LFW.read_pairs(lfw_dir, lfw_pairs_file)
                print('\ndone in {} s'.format(t.now()))
            if continue_training != '':
                print('Restoring pretrained model: {}'.format(continue_training))
                saver.restore(sess, continue_training)
                sess.run(global_step.assign(
int(os.path.basename(continue_training).split('-')[-1])))
            if validation:
                test(lfw_pairs, enque_op, embeddings, image_patches, labels,
labels_out, batch_size, images_per_person, sess, embedding_size,
phase_train_placeholder, persons_per_batch, summary_op, summary_writer, False,
global_step, log_dir, True)
                blufr_list = BLUFR.ReadLFWList(lfw_dir, blufr_list_file)
                eval_embeddings_and_save_for_BLUFR(blufr_list, enque_op, embeddings,
image_patches, labels, labels_out, batch_size, images_per_person, sess, embedding_size,
phase_train_placeholder, log_dir, global_step)
                return

            while sess.run(epoch) < epoch_count:
                # тест по протооклу LFW
                if sess.run(epoch) % test_rate == 0:
                    test(lfw_pairs, enque_op, embeddings, image_patches, labels,
labels_out, batch_size, images_per_person, sess, embedding_size,
phase_train_placeholder, persons_per_batch, summary_op, summary_writer, True,
global_step, log_dir, sess.run(epoch) % (10*test_rate)==0)
                # 1 эпоха обучения сети
                train_epoch(sess, train_data, enque_op, embeddings, loss, total_loss,
epoch, image_patches, labels, labels_out, phase_train_placeholder,
learning_rate_placeholder, learning_rate, batch_size, train_op, epoch_sz,
images_per_person, persons_per_batch, global_step, input_sz, summary_op,
summary_writer)
                save_model(sess, saver, summary_writer, save_dir, epoch, global_step)
```

```
            # сохранение отображений для тестирования по протоколу BLUFR
            blufr_list = BLUFR.ReadLFWList(lfw_dir, blufr_list_file)
            eval_embeddings_and_save_for_BLUFR(blufr_list, enque_op, embeddings,
image_patches, labels, labels_out, batch_size, images_per_person, sess, embedding_size,
phase_train_placeholder, log_dir, global_step)
            test(lfw_pairs, enque_op, embeddings, image_patches, labels, labels_out,
batch_size, images_per_person, sess, embedding_size, phase_train_placeholder,
persons_per_batch, summary_op, summary_writer, True, global_step, log_dir, True)
            summary_writer.close()
    print('done')


def parse_args(argv):
    parser = argparse.ArgumentParser()
    parser.add_argument('--validation', type=bool, default=False)  # устанавливается
для проверки сети, без обучения
    parser.add_argument('--images_per_person', type=int, default=5)
    parser.add_argument('--persons_per_batch', type=int, default=20)
    parser.add_argument('--persons_per_epoch', type=int, default=10500)
    parser.add_argument('--epoch_count', type=int, default=1500)
    parser.add_argument('--test_rate', type=int, default=1)
    parser.add_argument('--image_sz', type=int, default=120)
    parser.add_argument('--embedding_size', type=int, default=128)
    parser.add_argument('--use_random_flip', type=bool, default=True)
    parser.add_argument('--use_random_crop', type=bool, default=True)
    parser.add_argument('--add_noise', type=bool, default=False)
    parser.add_argument('--R', type=float, default=3)
    parser.add_argument('--n', type=float, default=6)
    parser.add_argument('--lr_decay_epoch', type=int, default=20)
    parser.add_argument('--learning_rate_decay_factor', type=float, default=0.9)
    parser.add_argument('--learning_rate', type=float, default=0.1)
    parser.add_argument('--keep_prob', type=float, default=0.8)
    parser.add_argument('--weight_decay', type=float, default=1e-4)
    # директории
    # parser.add_argument('-- continue_training', type=str, default=None)
    parser.add_argument('--train_set_dir', type=str,
default=r'D:\datasets\verticalized\CASIA-120')
    parser.add_argument('--lfw_dir', type=str, default=r'D:\datasets\verticalized\LFW-
120')
    parser.add_argument('--lfw_pairs_file', type=str,
default=r'D:\datasets\lfw\view2\pairs.txt')
    parser.add_argument('--blufr_list_file', type=str)
    return parser.parse_args(argv)


if __name__ == '__main__':
    main(parse_args(sys.argv[1:]))
```