**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра технологий программирования**

**ПРОЕКТИРОВАНИЕ СИСТЕМЫ ОЦЕНКИ КАЧЕСТВА НАПИСАНИЯ РУКОПИСНЫХ СИМВОЛОВ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ ONE-SHOT ОБУЧЕНИЯ**

|  |  |
| --- | --- |
|  |  |

Курсовая работа

|  |
| --- |
| Фурса Дмитрия Сергеевича |
| студента 3 курса,  специальность «прикладная информатика» |
| Научный консультант:  инженер-программист ООО «Техартгрупп»  Ю.В. Ветров  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Научный руководитель:  старший преподаватель  М.Л. Солодовникова  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Минск 2020

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 7](#_Toc39426146)

[**Глава 1** **ПОДХОДЫ К СРАВНЕНИЮ ИЗОБРАЖЕНИЙ** 9](#_Toc39426147)

[**1.1 Описание подходов к сравнению изображений** 9](#_Toc39426148)

[**1.1.1 Keypoint matching** 9](#_Toc39426149)

[**1.1.2 Метод гистограмм** 10](#_Toc39426150)

[**1.1.3 Метод хэшей** 12](#_Toc39426151)

[**1.1.4 Общие особенности** 14](#_Toc39426152)

[**Глава 2** **СИАМСКИЕ НЕЙРОННЫЕ СЕТИ ДЛЯ СРАВНЕНИЯ ИЗОБРАЖЕНИЙ** 15](#_Toc39426153)

[**2.1. Введение в нейронные сети** 15](#_Toc39426154)

[**2.2. Свёрточные нейронные сети** 20](#_Toc39426155)

[**2.3. Архитектура сиамских нейронных сетей** 24](#_Toc39426156)

[**Глава 3** **РЕАЛИЗАЦИЯ ПРОТОТИПА СИСТЕМЫ ОЦЕНКИ КАЧЕСТВА НАПИСАНИЯ РУКОПИСНЫХ СИМВОЛОВ.** 31](#_Toc39426157)

[**3.1 Проектирование системы** 31](#_Toc39426158)

[**3.2 Тестирование и оценка качества результатов** 34](#_Toc39426159)

[**3.3 Выбор оптимальной архитектуры сети** 35](#_Toc39426160)

[**Глава 4** **РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПЛАТФОРМЫ ANDROID.** 39](#_Toc39426161)

[**4.1 Подготовка разработанной модели** 39](#_Toc39426162)

[**4.2 Экраны приложения** 40](#_Toc39426163)

[**4.2.1 Главное меню** 41](#_Toc39426164)

[**4.2.2 История попыток** 45](#_Toc39426165)

[**4.2.3 Сведения о модели** 46](#_Toc39426166)

[**ЗАКЛЮЧЕНИЕ** 48](#_Toc39426167)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 50](#_Toc39426168)

[**ПРИЛОЖЕНИЯ** 52](#_Toc39426169)

[*Приложение А* 52](#_Toc39426170)

[*Приложение Б* 54](#_Toc39426171)

[*Приложение В* 55](#_Toc39426172)

[*Приложение Г* 57](#_Toc39426173)

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

|  |  |
| --- | --- |
| SIFT | масштабно-инвариантная трансформация признаков (Scale-invariant feature transform), |
| PCA | метод главных компонент (Principal component analysis), |
| LSH | локально-чувствительное хэширование (Locality-sensitive hashing), |
| ReLU | усечённое линейное преобразование (Rectified linear unit), |
| Adam | адаптивная оценка момента (Adaptive moment estimation), |
| MNIST | Modified National Institute of Standards and Technology, |
| MAE | средняя абсолютная ошибка (Mean absolute error), |
| MSE | среднеквадратическая ошибка (Mean squared error), |
| RMSE | квадратный корень из среднеквадратической ошибки (Root mean squared error). |

РЕФЕРАТ

Курсовая работа, 50 c., 20 рис.

**Ключевые слова:** СИАМСКИЕ НЕЙРОННЫЕ СЕТИ, МАШИННОЕ ОБУЧЕНИЕ, PYTHON, KERAS, ANDROID, JAVA, ONE-SHOT ОБУЧЕНИЕ, СРАВНЕНИЕ ИЗОБРАЖЕНИЙ, ОЦЕНКА КАЧЕСТВА НАПИСАНИЯ СИМВОЛОВ.

**Объект исследования —** сиамские нейронные сети и их использование в рамках технологии One-shot обучения для сравнения изображений.

**Цели работы —** исследовать подходы к сравнению изображений и разработать систему оценки качества написания рукописных символов с использованием технологии One-shot обучения.

**Методы исследования —** анализ, моделирование, эксперимент.

**Результатами являются —** система для оценки качества написания рукописных символов.

**Область применения —** для любого смартфона с операционной системой Android.

РЭФЕРАТ

Курсавая работа, 50 ст., 20 мал.

**Ключавыя словы**: СІЯМСКІЯ НЕЙРОНАВЫЯ СЕТКІ, МАШЫННАЕ НАВУЧАННЕ, PYTHON, KERAS, ANDROID, JAVA, ONE-SHOT НАВУЧАННЕ, ПАРАЎНАННЕ МАЛЮНКАЎ, АЦЭНКА ЯКАСЦІ НАПІСАННЯ ЗНАКАЎ.

**Аб'ект даследавання** **—** сіямскія нейронавыя сеткі і іх выкарыстанне ў межах тэхналогіі One-shot навучання для параўнання малюнкаў.

**Мэты працы** **—** даследаваць падыходы да параўнанні малюнкаў і распрацаваць сістэму ацэнкі якасці напісання рукапісных знакаў з выкарыстаннем тэхналогіі One-shot навучання.

**Метады даследавання** **—** аналіз, мадэляванне, эксперымент.

**Вынікамі з'яўляюцца** **—** сістэма для ацэнкі якасці напісання рукапісных знакаў.

**Вобласць ужывання** **—** для любога смартфона з аперацыйнай сістэмай Android.

ESSAY

Course work, 50 p., 20 illustrations.

**Keywords**: SIAMESE NEURAL NETWORKS, MACHINE LEARNING, PYTHON, KERAS, ANDROID, JAVA, ONE-SHOT LEARNING, IMAGE COMPARISON, ASSESSING THE QUALITY OF WRITING CHARACTERS.

**Object of research is** siamese neural networks and their use as a part of One-shot learning technology for image comparison.

**Purpose is** to investigate approaches to image comparison and to develop a system for assessing the quality of writing handwritten characters using One-shot learning technology.

**Methods of research** **are** analysis, modeling, experiment.

**The results are** a system used to evaluate the quality of handwritten characters.

**Scope** **for** any smartphone with Android operating system.

# **ВВЕДЕНИЕ**

В современном мире первый этап образования предполагает обучение. Обучение, в свою очередь, необходимо не только детям, но и людям, которые стремятся изменить сферу своей деятельности, развить новые навыки и улучшить уже приобретённые способности.

Во время обучения люди используют различные материалы и приложения, которые облегчают процесс приобретения новых знаний, помогают лучше усваивать новые сведения. В настоящее время все большее и большее количество детей получает первые знания задолго до прихода в школу. Это продиктовано стремительно развивающимся миром современных технологий. В связи с этим возникает необходимость создания новых приложений, разработок, призванных стать интерактивными помощниками на пути к дошкольному образованию.

Разрабатываемая система предлагает осуществлять тренировку навыков написания рукописных символов с целью максимального приближения к идеалу. Система производит оценку качества написания символов, позволяет развивать мелкую моторику, что в свою очередь оказывает влияние на процессы мозговой деятельности, взаимодействует с мышлением, воображением, зрительной и двигательной памятью, наблюдательностью, координацией.

Система может быть полезна не только среди детей дошкольного и школьного возраста, но и может применяться для развития навыков написания у людей с ограниченными возможностями.

Система разрабатывается для любого смартфона с операционной системой Android. Для разработки использован язык Java.

Объектом исследования являются сиамские нейронные сети и их использование в рамках технологии One-shot обучения для сравнения изображений.

Методы исследования, которые применялись в работе, включают в себя анализ, моделирование и эксперимент.

В главе 1 описываются классические подходы к сравнению изображений, кратко рассматриваются их особенности, достоинства и недостатки.

В главе 2 описываются нейронные сети, их базовая структура, особенности архитектуры, область применения, а также свёрточные нейронные сети. Далее представляется технология One-shot обучения, а также сиамские нейронные сети и возможность их использования для задачи сравнения изображений.

В главе 3 показана реализация прототипа системы оценки качества написания рукописных символов с использованием технологии One-shot обучения. Осуществляется выбор оптимальной архитектуры сети, проводится тестирование и оценка качества полученных результатов.

В главе 4 представлена подготовка модели для её использования на мобильном устройстве, описывается разработка мобильного приложения.

# **ПОДХОДЫ К СРАВНЕНИЮ ИЗОБРАЖЕНИЙ**

## **1.1 Описание подходов к сравнению изображений**

Среди наиболее популярных подходов к сравнению изображений выделяют: совпадение ключевых точек (Keypoint matching), метод гистограмм, методы на основе хэша.

### **1.1.1 Keypoint matching**

Первый подход широко используется в компьютерном зрении и считается базовым. Суть алгоритма заключается в следующем: сначала извлекаются и сохраняются ключевые точки объектов из набора контрольных изображений, затем производится сравнение нового изображения с контрольными, путём сравнивания каждого признака из нового изображения с сохранёнными контрольными признаками на основе расстояния между векторами признаков. В качестве метрики расстояния, как правило, используется евклидово расстояние. Далее из полученного набора соответствий выбираются поднаборы точек, имеющие наибольшую согласованность с эталонным объектом по местоположению, ориентации и другим признакам. Резко отклоняющиеся поднаборы отбрасываются. Затем вычисляется вероятность того, что определённый набор признаков свидетельствует о присутствии объекта. Так, например, широко используемым алгоритмом является масштабно-инвариантная трансформация признаков (scale-invariant feature transform, SIFT). Алгоритм был опубликован в 1999 году Дэвидом Лоу [[5](#Sift)]. Приложения алгоритма включают распознавание объектов, роботизированную навигацию, трёхмерное моделирование, распознавание жестов, трекинг.

Особенность SIFT заключается в том, что выделяемые признаки являются локальными и основаны на проявлении объекта в конкретных контрольных точках. Они неизменны по отношению к изменению масштаба и вращению изображения, стабильны к шумам. Это позволяет проводить сравнение изображений, отличающихся масштабом, углом поворота, освещением. Более того, такие признаки высоко различимы и относительно легко извлекаются.

Существуют альтернативные алгоритмы для распознавания объектов инвариантно по масштабу. Среди них можно выделить инвариантное относительно вращения обобщение SIFT (RIFT), использующее концентрические кольца для выделения признаков, PCA-SIFT (использующий метод главных компонент для сокращения размерности признакового пространства) и многие другие методы, в основе которых используется идея того, что внешний вид и форма объекта на участке изображения могут быть описаны распределением градиентов интенсивности или направлением краёв (Рисунок 1.1).

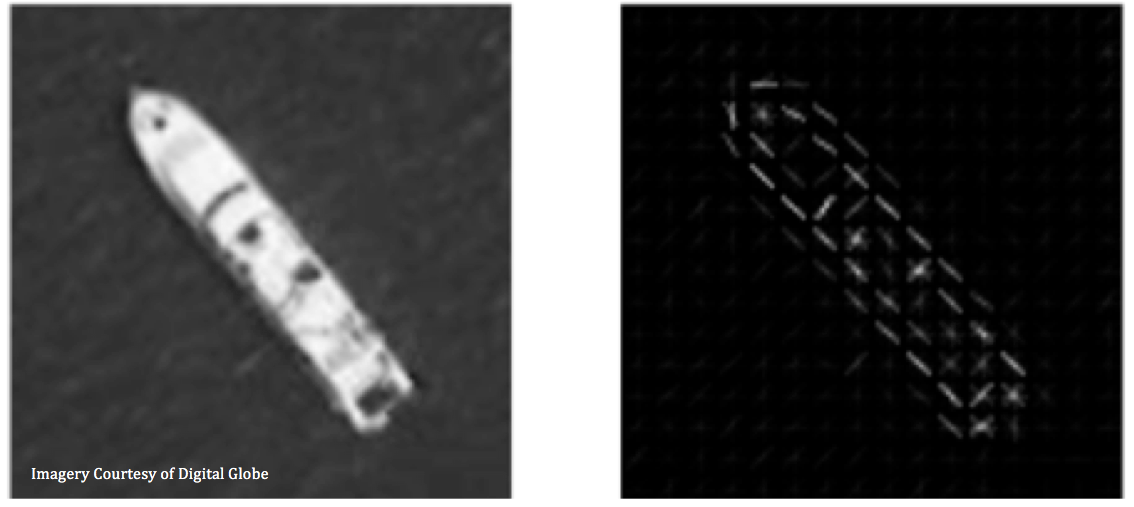
[](https://www.google.com/url?sa=i&url=https%3A%2F%2Fmedium.com%2Fthe-downlinq%2Fhistogram-of-oriented-gradients-hog-heading-classification-a92d1cf5b3cc&psig=AOvVaw3MgPyzYytNOiPmxdmEe7Ae&ust=1575285352900000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCKCanLKplOYCFQAAAAAdAAAAABAK)

Рисунок 1.1 — Гистограмма направленных градиентов на примере лодки

Основным же недостатком алгоритмов этого семейства является время работы. Так, наивная реализация базового алгоритма потребует O(n^(2m)) операций, где n — количество контрольных точек для каждого изображения, m — количество эталонных изображений.

### **1.1.2 Метод гистограмм**

Метод гистограмм является менее стойким, но в то же время потенциально более быстрым алгоритмом. Однако в этом случае накладываются определённые ограничения на сравниваемые изображения. Алгоритм был предложен в 1991 году Даном Баллардом и Майклом Свэйном [[4](#Color_indexing)].

Идея заключается в следующем: для каждого изображения строится гистограмма признаков, а затем происходит поиск эталонного изображения, гистограмма которого наиболее близка к гистограмме входного изображения. Как правило, используются 3 гистограммы цвета (RGB, red green blue) и две дополнительные диаграммы признаков, отвечающих за направление и масштаб. Для построения цветовых диаграмм, для каждого цвета палитры выделяется определенное количество ячеек-диапазонов, затем для каждого пикселя значение соответствующего цветового канала заносится в нужную ячейку гистограммы. После этого выполняется нормализация полученных значений, путём их шкалирования (деления на общее количество пикселей изображения). Гистограмма направлений строится аналогичным образом, однако в качестве значений уже выступают углы наклона векторов-направлений граничных точек. Для построения гистограммы масштаба, для каждой граничной точки высчитывается расстояние до ближайшей точки, имеющей такое же направление вектора. Далее происходит вычисление похожести гистограмм изображений.

Обозначим гистограмму входного изображения — I, эталонного изображения — M, каждая из этих гистограмм содержит одинаковое число ячеек — n.

Вычислим

|  |  |  |
| --- | --- | --- |
|  |  | (1.1) |

где функция min принимает два аргумента и возвращает наименьший из них. В результате мы получим количество пикселей эталонного изображения, которые соотносятся с пикселями такого же цвета/признака входного изображения. Для того, чтобы получить вероятность совпадения изображений, необходимо пронормировать полученное значение на количество пикселей эталонного изображения.

Но, как уже говорилось ранее, существуют определенные ограничения. Данный метод хорошо применим только для изображений, цвет которых является значимым предиктором идентичности объекта. В то же время, метод гистограмм не требует точного отделения объекта от фона, он устойчив к сокрытию объектов на переднем плане.

### **1.1.3 Метод хэшей**

Ещё одним популярным алгоритмом является сравнение хэшей изображений. Основная идея заключается в том, что исходное изображение сокращается в размере до небольшого хэш-кода («отпечатка») путём выявления характерных особенностей в исходном изображении и их хэшировании. Далее, для полученных хэшей изображений рассчитывается мера их различия, например расстояние Хэмминга, по величине которого можно судить о совпадении или несовпадении изображений. Изображения, которые кажутся нам похожими, могут очень сильно отличаться, если проводить их простое побайтовое сравнение. Это может быть вызвано рядом особенностей, таких как: разный масштаб, формат изображения, наличие шумов, различные оттенки цветов, поворот изображения. Поэтому использование хэш-функций, широко применяющихся в криптографии, не пригодно для данного метода сравнения изображений.

Здесь нужно использовать такие хэш-функции, которые генерировали бы похожий хэш для похожих объектов (изображений). В этом случае можно использовать вероятностный метод понижения размерности данных (Locality-sensitive hashing) [[3](#lsh)]. Основная идея данного метода состоит в подборе таких хэш-функций, чтобы похожие объекты с высокой степенью вероятности попадали в одну группу. LSH на сегодняшний день является одним из наиболее популярных алгоритмов поиска ближайшего соседа. В отличие от традиционных хэшей, он обладает свойством локальной чувствительности, что позволяет помещать соседние точки в одну группу (бин — аналог ячейки гистограммы).

Помимо этого, существует так называемое перцептивное хэширование (Perceptual hashing). В этом случае используются функции для генерации сравнимых хэшей. Характеристики изображений используются для создания отпечатков, которые затем можно сравнивать между собой. При этом отпечаток каждого изображения не является уникальным.

Один из наиболее простых алгоритмов является a-hash (average hash), который отображает средние значения низких частот. В изображении низкие частоты отвечают за структуру картинки, а высокие обеспечивают детализацию. Большое, детализированное изображение содержит много высоких частот, маленькая картинка имеет не так много деталей, а следовательно, и состоит из низких частот. Базовый алгоритм заключается в следующем: сначала мы уменьшаем размер изображения, тем самым избавляясь от высоких частот, затем изображение переводится в оттенки серого (greyscale), таким образом мы уменьшаем размер хэша от количества пикселей n (по трём каналам) до n значений цвета. Далее вычисляется среднее значение для всех цветов, затем значение каждого цвета сравнивается со средним и кодируется 0 или 1 (1 — если значение больше среднего, 0 — в противном случае). На выходе получается n-битное хэш-значение. Итоговый хэш не изменится, если картинку масштабировать, сжать, растянуть. Даже изменение яркости, контрастности или цветов не сильно повлияет на итоговый результат. В дополнение к этому, данный алгоритм выполняется очень быстро. Для того, чтобы сравнить две картинки нужно вычислить расстояние Хэмминга для полученных хэшей.

Однако в некоторых случаях, a-hash может давать сбои. Например, в случае изменения цветовой гистограммы, многие биты поменяют свои значения, что отразится на хэш-значении. Тогда целесообразнее применить другие алгоритмы, например p-hash, использующий дискретное косинусное преобразование, для устранения высоких частот. Изображение, как и в a-hash, сначала уменьшают в размере, затем убирают цветовые каналы, потом применяют дискретное косинусное преобразование, которое разбивает картинку на набор частот и векторов. После этого высчитывается среднее и каждое значение кодируется 0 или 1, аналогично a-hash. P-hash работает медленнее a-hash, однако он более стабилен к манипуляции над самой картинкой.

### **1.1.4 Общие особенности**

Все алгоритмы для сравнения изображений, так или иначе используют извлеченные из картинки признаки. Затем полученные наборы признаков сравниваются между собой. В зависимости от алгоритма для сравнения могут использоваться различные метрики, такие как, например, евклидово расстояние для векторов-признаков. На основе полученного значения, учитывая порог (margin), делается вывод о совпадении или несовпадении исходных изображений.

ВЫВОДЫ

1. Рассмотрены наиболее популярные подходы к сравнению изображений.
2. Выявлены преимущества и недостатки применяемых алгоритмов.
3. Выделены ключевые особенности, которые, в той или иной степени, присущи всем алгоритмам сравнения изображений.

# **СИАМСКИЕ НЕЙРОННЫЕ СЕТИ ДЛЯ СРАВНЕНИЯ ИЗОБРАЖЕНИЙ**

## **2.1. Введение в нейронные сети**

Нейронная сеть — это математическая модель, а также её программная реализация, которая построена по принципу организации и функционирования биологических нейронных сетей [[1](#network), c. 1]. Искусственная нейронная сеть представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Как правило, нейроны такой сети имеет довольно простую структуру, каждый из них работает с сигналами, которые он периодически получает, и сигналами, которые он периодически отправляет другим нейронам.

Объединение таких простых процессоров в большие связные сети, позволяет решать довольно сложные задачи. Нейронные сети используются для задач классификации (распределение объектов по классам, с заранее известным количеством классов и признаков объектов), кластеризации (разбиение множества входных сигналов на классы, при том, что ни количество, ни признаки классов заранее не определены), прогнозирование (возможность предсказывать следующий шаг), аппроксимации непрерывных функций, распознавания образов. Как и в биологических нейронных сетях составными компонентами искусственной нейронной сети являются нейроны и связи между ними (синапсы).

Нейрон представляет собой вычислительную единицу нейронной сети, которая получает информацию, производит над ней некоторые простые вычисления и передаёт её дальше. Существует три основных типа нейронов: входной (input), скрытый (hidden), выходной (output) (Рисунок 2.1).

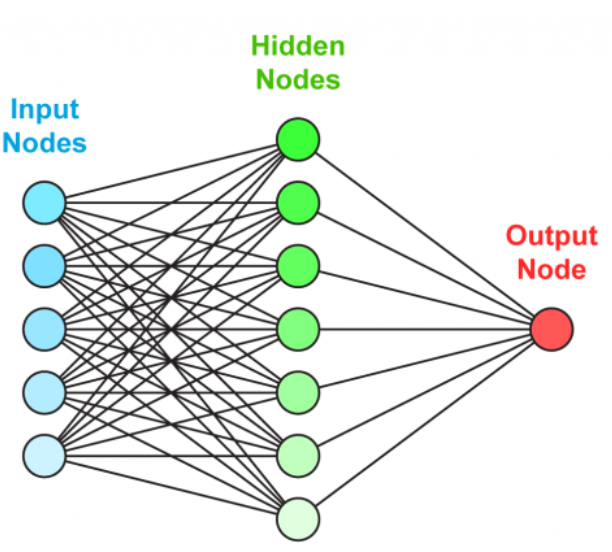


Рисунок 2.1 — Виды нейронов

В случае, когда нейронная сеть состоит из большого числа нейронов, нейроны одного типа объединяют в слои. Такие нейронные сети могут иметь несколько скрытых слоёв. Нейроны входного слоя получают информацию на вход, затем эту информацию обрабатывают нейроны скрытых слоёв, передавая её на вход выходному слою, который выводит конечный результат. Каждый нейрон имеет 2 основных параметра: входные данные (input data) и выходные данные (output data). Чаще всего на вход нейрону (скрытого слоя) подаётся суммарная информация от всех нейронов предыдущего слоя, после чего она нормализуется с помощью специальной функции (функции активации) и передаётся по сети далее. Это так называемая сеть прямого распространения (feedforward), в которой все связи направлены строго от входных нейронов к выходным. По характеру связей существуют также рекуррентные нейронные сети, сигналы с выходных нейронов или нейронов скрытого слоя которых частично передаются обратно на входы нейронов выходного слоя. Такая архитектура позволяет организовать ассоциативную память и широко используется для задач распознавания речи или распознавания рукописных символов.

Синапс представляет собой связь между двумя нейронами, он имеет один параметр — вес. Благодаря весу, изменяется входная информация, которая передается от одного нейрона в другому. В случае если у нас три нейрона передают информацию следующему, мы получаем три веса для каждого из синапсов (Рисунок 2.2). Важнейшей частью нейронной сети является матрица весов, которая представляет собой мозг нейронной сети.

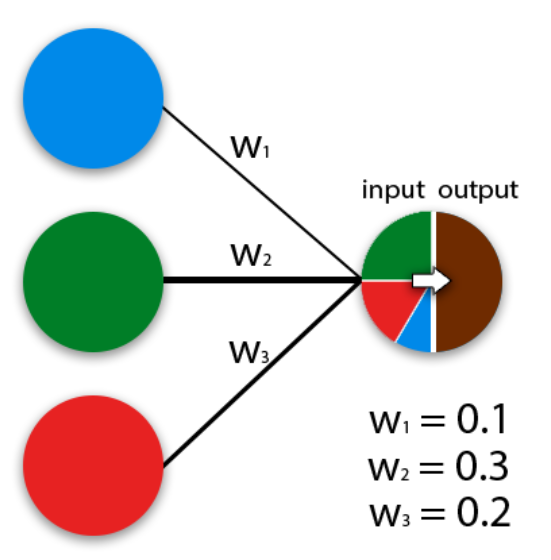


Рисунок 2.2 — Нейроны и веса

Обозначим буквой H — нейрон скрытого слоя, буквой I — нейрон входного слоя, w — веса, f — функция активации. Получим следующую простую архитектуру (Рисунок 2.3).

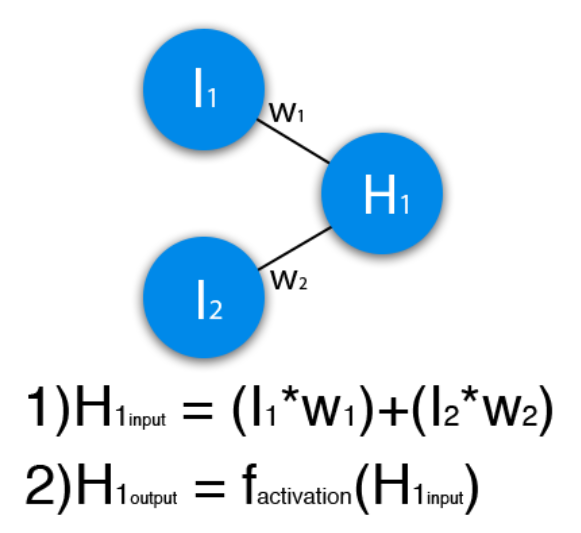


Рисунок 2.3 — Составная часть нейронной сети

Входная информация для нейрона H представляет собой сумму выходных данных нейронов I, умноженных на соответствующие веса. Далее для получения выходных данных нейрона H, мы рассчитываем значение функции активации и передаём его дальше по сети, пока не дойдём до выходного слоя.

Функция активации — это способ нормализации входных данных (т.к. нейроны оперируют числами в диапазоне от 0 до 1 или от -1 до 1). Она также частично определяет то, какие нейроны будут активированы, то есть какая информация будет передаваться последующим слоям. Существует множество функций активации, вот лишь некоторые из них (Рисунок 2.4):

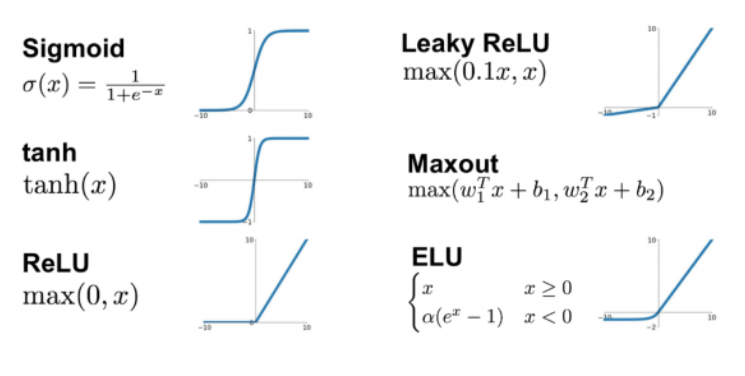


Рисунок 2.4 — Простейшие функции активации

В общем случае желательными свойствами для функции активации являются: нелинейность (тогда можно показать, что двухуровневая нейронная сеть будет универсальным аппроксиматором), непрерывная дифференцируемость (для обеспечения методов оптимизации на основе градиентного спуска), монотонность (поверхность ошибок будет выпуклой). Область значений функции активации также влияет на процесс обучения. В случае ограниченного множества значений, методы обучения на основе градиентного спуска будут более стабильны. Если же область значений бесконечна, то, как правило, обучение более эффективно, так как в этом случае мы будем иметь более существенное влияние на большинство весов. Выбор функции активации может зависеть от конкретной задачи. Так, например, сигмоида хорошо показывает себя в задачах классификации, ReLU работает как хороший аппроксиматор. Важно также подбирать функцию таким образом, чтобы процесс обучения и сходимость были быстрее. В это случае можно поэкспериментировать с разными функциями и найти наиболее подходящую.

Однако такая архитектура кажется статичной и вряд ли может справиться с серьёзными задачами. Суть обучения нейронной сети заключается в том, что мы будем настраивать параметры весов, минимизируя функцию потерь так, чтобы нейроны реагировали на нужные сигналы. Допустим, что у нас есть размеченные данные, то есть примеры входных и правильных выходных сигналов (таких, которых мы бы хотели добиться получить от сети). В начале обучения делается проход по сети от входного слоя к выходному, таким образом на выходе мы получаем некоторое предсказанное значение. Сравнив предсказанное значение с желаемым, мы начинаем двигаться по сети в обратном направлении, от выходного слоя к входному, и настраивать синаптические веса таким образом, чтобы максимально приблизить выходной сигнал к желаемому. Данный подход называется «Методом обратного распространения ошибки». Он впервые был описан в 1974 году А.И. Галушкиным, а также независимо и одновременно Полом Дж. Вербосом. В настоящее время данный алгоритм широко используется на практике, для обучения многослойных нейронных сетей прямого распространения.

Для расчёта ошибки между реальными и полученными ответами используется функция потерь (loss function). Главной задачей является минимизация этой функции. Она оценивает, насколько хорошо работает нейронная сеть в целом. Среди функций потерь выделяют MAE (mean absolute error), MSE (mean squared error), RMSE (root mean squared error). Каждый из этих методов рассчитывает ошибку по-разному. Как правило, выбор функции потерь определяется решаемой задачей. Так, например, в задачах классификации наиболее естественным выбором является пороговая функция потерь (точнее её непрерывные аппроксимации). Для борьбы с переобучением (когда предсказания алгоритма слишком близко или точно соответствуют конкретному набору данных и поэтому не подходят для применения алгоритма к новым данным) используются регуляризаторы, которые могут как штрафовать за получение больших весов модели, так и обнулять очень маленькие веса. Всё это делается с целью получения самой простой модели.

## **2.2. Свёрточные нейронные сети**

Особое место среди нейронных сетей занимают свёрточные нейронные сети. Свёрточная нейронная сеть представляет собой специальную архитектуру искусственной нейронной сети, предложенную в 1988 году и нацеленную на эффективное распознавание объектов [[2](#conv), c. 1]. В основе сверточных сетей лежат некоторые особенности зрительной коры мозга, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток.

Работа свёрточной нейронной сети обычно описывается как переход от конкретных особенностей изображения к более абстрактным деталям, и далее к ещё более абстрактным вплоть до выделения понятий высокого уровня. Свёрточные нейронные сети обеспечивают частичную устойчивость к изменениям масштаба, смещениям, поворотам и прочим искажениям.

Свёрточная нейронная сеть состоит из слоёв трёх видов: свёрточные слои (convolutional), слои субдискретизации (subsampling) и слои обычной нейронной сети (персептрона) (Рисунок 2.5). Первые два типа слоёв формируют входной вектор признаков для персептрона.

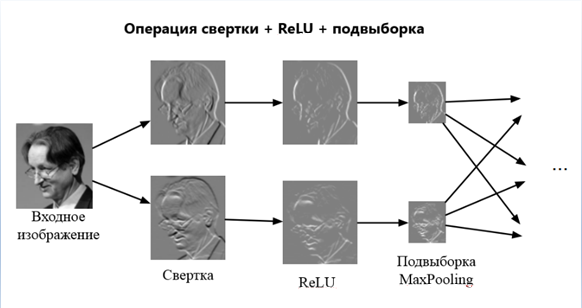


Рисунок 2.5 — Свёрточная нейронная сеть

Предположим, что мы решаем задачу классификации, и на вход свёрточной сети подаются изображения в определённом формате. Если размер таких изображений будет велик, то вычислительная сложность значительно повысится, если же выбрать слишком маленький размер, то сеть может не выявить ключевые признаки. Учитываю двумерную топологию изображений, на вход сети будет подаваться несколько матриц (карт), количество которых зависит от количества каналов. Входные данные каждого пикселя нормализуются в диапазон от 0 до 1.

Свёрточный слой представляет собой набор карт (карт признаков), каждая из которых содержит синаптическое ядро (фильтр). Увеличение количества карт приводит к повышению точности распознавания, однако вместе с этим увеличивается и вычислительная сложность. Стоит также отметить, что процесс свёртки может быть успешно распараллелен по каждой карте, как и процесс обратной свёртки во время обратного распространения ошибки.

Ядро представляет собой окно (фильтр), который перемещается по всей области предыдущей карты и находит определённые признаки. Если размер ядра слишком маленький, то оно с меньшей вероятностью сможет выделить какие-то признаки, если же ядро будет слишком большим, то мы увеличим количество связей в сети, а значит повысится и вычислительная сложность (Рисунок 2.6). Ядро представляет собой систему разделяемых весов, что значительно ускоряет процесс детектирования по сравнению с обычной многослойной сетью.

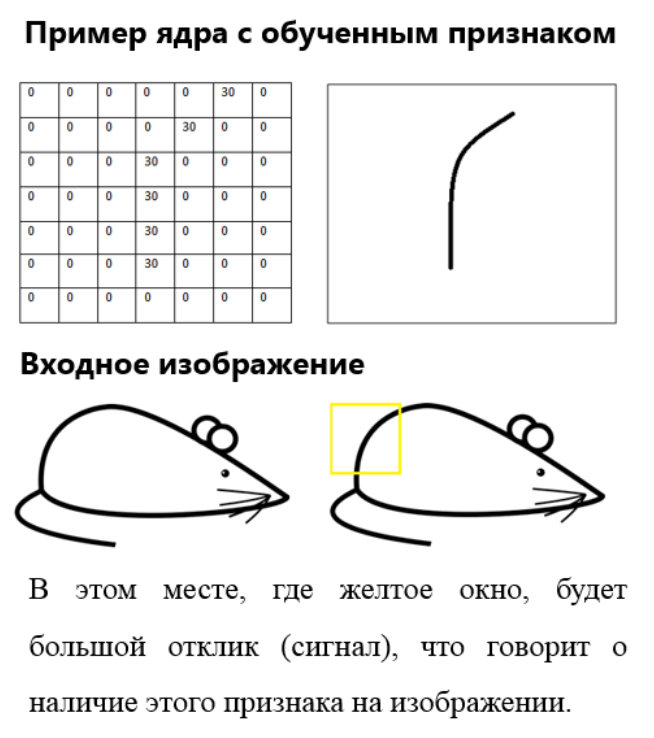


Рисунок 2.6 — Пример ядра свёртки

Изначально значения каждой карты свёрточного слоя равны 0, однако они могут быть проинициализированы и другими значениями, например, единицами, константами или случайными величинами из заданного распределения.

Ядро скользит по предыдущей карте и производит операцию свёртки:

|  |  |  |
| --- | --- | --- |
|  |  | (2.1) |

где f — исходная матрица изображения, g — ядро свёртки.

Окно размером ядра g проходит с заданным шагом всё изображение f, на каждом шаге поэлементно умножается содержимое окна на ядро g, результат суммируется и записывается в матрицу результата (Рисунок 2.7).

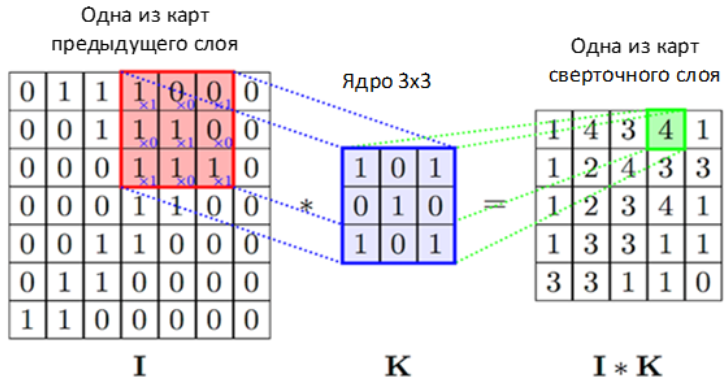


Рисунок 2.7 — Операция свёртки

При этом в зависимости от метода обработки краёв исходной матрицы, результат может иметь размерность меньше, чем размерность исходного изображения, такую же или большую размерность.

Подвыборочный слой также имеет карты, однако их количество совпадает с предыдущим слоем. Основная цель этого слоя — уменьшение размерности карт предыдущего слоя. Если на предыдущем шаге уже были выявлены некоторые признаки, то для последующей обработки подробное изображение уже не нужно, и оно уплотняется до менее подробного. Это также позволяет избежать процесса переобучения. В подвыборочном слое, при обработке изображения карты не пересекаются, обычно каждая карты имеет ядро 2x2, что позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Карта разделяется на ячейки 2x2 элемента, из которых выбираются элементы максимальные по значению (Рисунок 2.8).

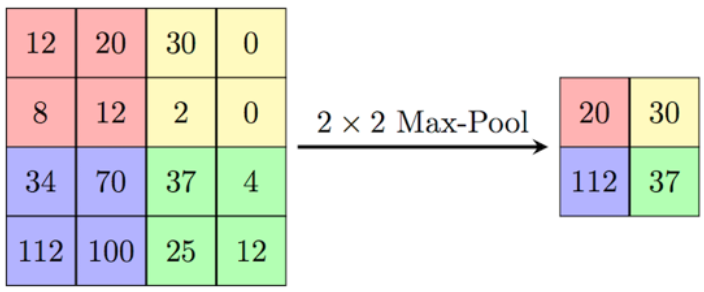


Рисунок 2.8 — Подвыборочный слой

Последний слой представлен обычным многослойным персептроном, задача которого состоит в классификации. Путём оптимизации функции этого слоя улучшается качество распознавания.

Нейроны каждой карты предыдущего подвыборочного слоя связаны с одним нейроном скрытого слоя. Число нейронов скрытого слоя равно числу карт подвыборочного слоя. Выходной слой связан со всеми нейронами предыдущего слоя, количество нейронов равно количеству распознаваемых классов.

## **2.3. Архитектура сиамских нейронных сетей**

В классических задачах классификации на вход нейронной сети подаются изображения, принадлежащие различным классам, далее происходит их обработка, для которой используются свёрточные и подвыборочные слои. На выходе мы получаем вероятность принадлежности объекта тому или иному классу. Однако для такого подхода характерны определённые недостатки. Во-первых, для хорошего обучения свёрточной сети требуется большое количество данных. Выборка также должна быть репрезентативной, то есть содержать приблизительно равное количество изображений различных классов. Более того, модель должна предоставлять возможность обобщать полученные результаты применительно к новым данным. Во-вторых, множество классов может динамически изменятся, что вызовет необходимость переобучения модели, с использованием новых данных, которые в свою очередь нужно отобрать. Однако цена сбора новых данных и переобучения модели может быть очень высока.

Одним из возможных решений является использование One-shot обучения. Такой подход позволяет использовать для обучения сети один эталонный пример для каждого класса (или небольшое количество примеров) [[9](#oneshot), c. 1]. В свою очередь, построенная таким образом модель позволяет делать точные предсказания для многих неизвестных примеров в будущем.

Наиболее ярким примером использования One-shot обучения является распознавание лиц. Как правило, в этом случае нам доступно только небольшое количество изображений человеческих лиц. Мы должны иметь возможность точно распознавать лица (а следовательно, и идентифицировать человека) с учётом изменений выражения лица, причёски, освещения и многих других факторов.

В области нейронных сетей одной из самых популярных архитектур для проблемы One-Shot обучения являются сиамские нейронные сети. Впервые они были представлены в 2005 году Джейн Брумли. Сиамская нейронная сеть состоит из двух идентичных подсетей, соединённых на своих выходах. Одна из сетей принимает известное (эталонное) изображение, а другая - изображение-кандидат. Выходные данные обеих сетей объединяются и оцениваются, чтобы сказать принадлежит ли изображение-кандидат тому же классу, что и эталонное изображение (в случае человеческих лиц, имеем ли мы дело с одним и тем же человеком). Проверка заключается в сравнении извлечённых векторов признаков. Векторы, которые отличаются друг от друга (для сравнения близости используются различные метрики) менее, чем некоторый порог принимаются как представители изображений одного класса, все остальные представители отклоняются (Рисунок 2.9).

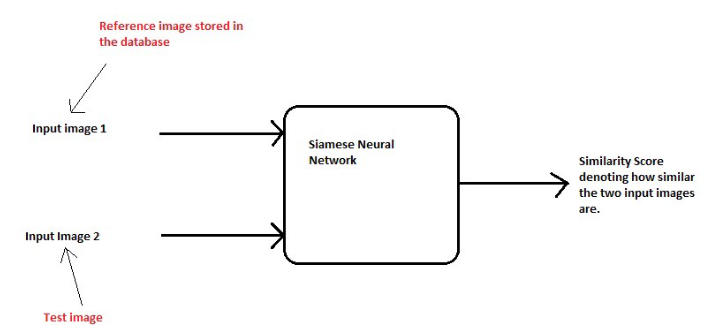


Рисунок 2.9 — Общая схема One-shot обучения с использованием сиамской нейронной сети

Одной из основных задач такого подхода является стремление к тому, чтобы научить модель распознавать абстрактные признаки и на их основе формировать вектора признаков. Далее модель может быть эффективно использована для проверки соответствия новых примеров шаблонам (эталонам) для каждого класса.

Для оценки похожести выходных векторов могут использоваться distance-функции, например, L1 distance, где p, q – выделенные вектора признаков

|  |  |  |
| --- | --- | --- |
|  |  | (2.2) |

то есть сумма модулей разности элементов векторов признаков.

Далее это значение подается на вход полносвязного слоя нейронной сети. В качестве функции активации обычно используется сигмоида или softmax.

Построение архитектуры сиамской нейронной сети можно провести следующим образом: сначала обучить модель для классификации (используя обычную свёрточную нейронную сеть), а затем удалить слой, отвечающий за классификацию, и использовать для активации вектора признаков, которые извлекаются из входных изображений и сравниваются друг с другом.

Сиамские нейронные сети не нацелены на классификацию изображений непосредственно, они используются для того, чтобы определить, насколько входные изображения похожи друг на друга. Это свойства позволяет эффективно использовать сиамские нейронные сети для сравнения изображений.

Такой подход также позволяет решать описанные выше проблемы. Во-первых, теперь для обучения сети нам не требуется большое количество экземпляров от каждого класса, достаточно лишь небольшого количества изображений для построения хорошей модели. Проблема динамического изменения количества классов также решается. Теперь, если нам необходимо обеспечить распознавание, например, лица нового человека, достаточно получить одно соответствующее изображение, которое будет принято за эталонное для добавленного человека. Получаемая модель довольна устойчива, что позволяет использовать её в различных системах проверки подписей (например, на основе человеческого лица). Так, например, в сканере объёмно-пространственной формы лица человека, разработанным компанией Apple, используются сиамские нейронные сети.

На вход сиамской нейронной сети, как правило, подаётся пара изображений, которая помечена 0 или 1, если изображения в паре относятся к разным классам или к одному классу соответственно. Таким образом нейронная сеть обучается на так называемых «положительных» и «отрицательных» парах изображений. Количество таких пар обычно совпадает. Для того, чтобы определить принадлежность (похожесть) нового изображения к уже существующим классам, необходимо подать на вход сети это изображение и по одному эталонному изображению для каждого класса (или несколько). Затем необходимо вычислить вероятность совпадения изображений для каждой пары и среди полученных вероятностей выбрать наибольшую. Пара с наибольшей вероятностью совпадения и будет указывать тот класс (по эталонному изображению в паре), которому принадлежит входное изображение.

Важной частью работы сети является выделение векторов признаков. Данная задача заключается в уменьшении размерности входных данных, она направлена на преобразование многомерных данных (например, изображений) в низкоразмерное представление таким образом, чтобы подобные объекты оставались близки в новом признаковом пространстве. Это делается в первую очередь для того, чтобы постараться избежать «проклятия размерности», что в первую очередь касается экспоненциального роста необходимых экспериментальных данных в зависимости от размерности пространства при решении задач вероятностно-статистического распознавания объектов. Также это позволяет более экономно расходовать ресурсы, отбирать наиболее важные признаки и может значительно упростить задачи визуализации многомерных данных.

Низкоразмерное представление должно сохранять структуру входных данных таким образом, чтобы расстояния между выходными векторами признаков отражали различия во входных данных. Тем не менее, векторы также должны захватывать и неизменные (инвариантные) признаки.

Как уже отмечалось ранее, устойчивым и эффективным инструментом для выделения признаков являются свёрточные нейронные сети. Однако возникает вопрос выбора оптимальной функции потерь, которую нужно будет минимизировать в ходе обучения. В 2006 году Райа Хадсел в своей работе «Dimensionality Reduction by Learning an Invariant Mapping» исследовала сиамские нейросети (со свёрточными сетями внутри) для уменьшения размерности изображений и предложила использовать для обучения таких сетей функции контрастных потерь (contrastive loss function)

|  |  |  |
| --- | --- | --- |
|  |  | (2.3) |

где у — метка пары (0 или 1); r0, r1 — векторы признаков для изображений пары, m — пороговое значение.

В отличие от других функций потерь, которые могу оценивать предсказательную способность модели по всем входным наборам данных, contrastive loss рассчитывается только между парами входов (например, парой изображений), которые подаются на вход сиамской сети. Размеченные пары изображений подаются на вход сети, и функция потерь по-разному исправляет модель в зависимости от того, как помечена соответствующая пара. Если в паре изображения являются представителями одного класса, то функция потерь корректирует модель таким образом, чтобы выделенные векторы признаков были похожи, тогда как, если классы отличаются, то функция побуждает модель выводить векторы признаков, которые менее похожи. Данная функция потерь требует определения некоторого предельного значения (порога), которое используется как критическое значение для определения совпадения или несовпадения изображений пары. Это в свою очередь является недостатком данного подхода. Ведь выбор значения порога влияет на чувствительность (способность распознавать изображений одного класса) и специфичность (способность не принимать представителей одного класса за изображения, принадлежащие разным классам). Оптимальное значение порога для конкретной задачи может быть найдено экспериментально.

Стоит отметить, что в настоящее время сиамские нейронные сети тренируют и с использованием троек изображений. В состав такой тройки входит само проверяемое изображение (входное) и пара изображений, одно из которых является «позитивным», т.е. принадлежит тому же классу, что и входное изображения, а другое — «негативным», т.е. класс его принадлежности отличается от класса входного изображения. Обучение происходит путём минимизации различий между представителями одного класса и максимизации различий между представителями разных классов. Для этого используется специальная функция потерь, которая называется triplet loss.

Сиамские нейронные сети также используются в задачах отслеживания объектов, благодаря своей способности к измерению подобия. При отслеживании объектов одним входом сиамской сети является то изображение, которое необходимо отследить (образец), другим входом является поисковое изображение. Задача здесь заключается в поиске образца внутри поискового изображения. Путём измерения сходства между образцом и каждой частью поискового изображения, сеть позволяет получить оценки сходства. Использование свёрточных сетей позволяет организовать быструю независимую обработку каждой части изображения.

ВЫВОДЫ

1. Рассмотрена общая архитектура нейронных сетей, область их применения.
2. Проанализирована структура свёрточных нейронных сетей.
3. Изучена технология One-shot обучения.
4. Описана возможность использования сиамских нейронных сетей в рамках технологии One-shot обучения.

# **РЕАЛИЗАЦИЯ ПРОТОТИПА СИСТЕМЫ ОЦЕНКИ КАЧЕСТВА НАПИСАНИЯ РУКОПИСНЫХ СИМВОЛОВ.**

## **3.1 Проектирование системы**

Прототип системы оценки качества написания рукописных символов был реализован на языке Python, c использованием открытой нейросетевой библиотеки Keras (как надстройкой над TensorFlow), c использованием сторонних библиотек, таких как numpy, sklearn и других. За основу набора данных была выбрана база образцов рукописного написания символов MNIST. База данных MNIST содержит 60000 изображений для обучения и 10000 изображений для тестирования.

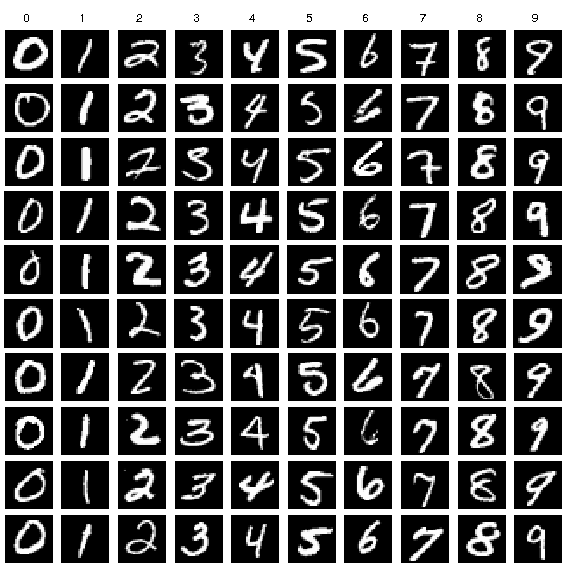
[](https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FExample-images-from-the-MNIST-dataset_fig1_306056875&psig=AOvVaw2m6xrBMx5A7-iA2FiPKEef&ust=1575404196219000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCOjpqJnkl-YCFQAAAAAdAAAAABAD)

Рисунок 3.1 — Набор данных MNIST

Каждое изображение было нормализовано и приведено к серому полутоновому изображению (grayscale) размером 28x28 пикселей. Таким образом удалось уменьшить количество каналов, а нормализация данных позволила значительно снизить вычислительную нагрузку, а также уменьшить значение цветов, которые сильно отклоняются от основного тона.

Формирование тренировочного и тестового набора данных происходило следующим образом: в начале был загружен исходный набор данных MNIST, размерностью 60000x784 и 10000x784 точек для тренировочной и тестовой выборки соответственно.

Далее происходило формирование «позитивных» и «негативных» пар изображений. Из исходного набора данных выбиралось определённое количество совпадающих цифр, пары которых помечались как идентичные с пометкой 1, и такое же количество пар несовпадающих символов, которые помечались 0.

Тренировочная и тестовая выборка представляли собой многомерный массив размерности Nx2x1x28x28, где N — размер выборки, 2 — количество изображений, 1 — количество фильтров (для свёрточного слоя), 28x28 — размер изображения в пикселях

Сиамская нейронная сеть была построена на основе двух идентичных свёрточных сетей следующей архитектуры (Рисунок 3.2).

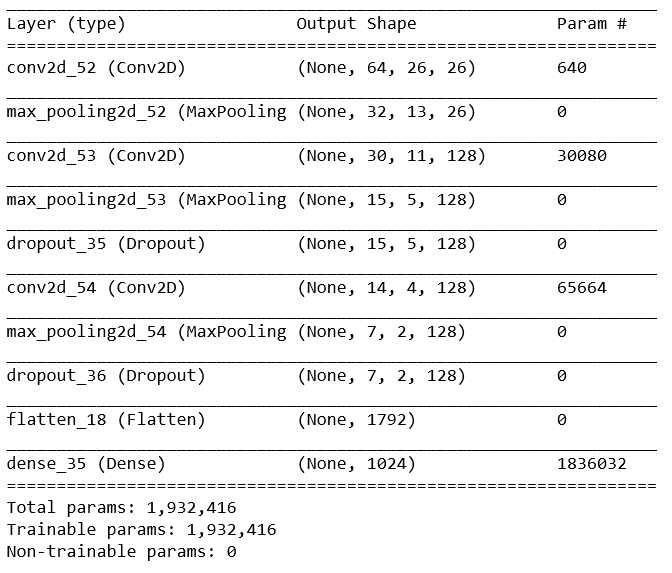


Рисунок 3.2 — Архитектура свёрточной сети

Параметры сетей (количество нейронов, используемые слои, порядок следования слоёв) были выбраны экспериментально, путём достижения высоких показателей точности (99,6 % на тестовой выборке) для классической задачи классификации рукописных цифр (с использованием обычной свёрточной нейронной сети). Также были использованы Dropout слои, для того чтобы предотвратить переобучение (Приложение А).

Далее в сиамской сети была высчитана L1-норма для выделенных векторов признаков. Последним был добавлен полносвязный слой с одним нейроном и сигмоидой в качестве функции активации.

Для тренировки модели была использована функция потерь contrastive loss, с порогом равным 1. В качестве оптимизатора (функция, которая минимизирует функцию потерь) был выбран Nesterov Adam optimizer.

Для «позитивных» пар с высокой вероятностью совпадения выходное значение сети стремилось к 0, для «негативных» пар — к 1. Соответственно, для получения вероятности совпадения необходимо было применить биективную трансформацию. Например, отняв от 1 полученное на выходе сети значение.

Для подсчёта точности модели (accuracy score) выходные значения сети сначала были округлены до соответствующей ближайшей метки (0 или 1), а затем была вычислена стандартная метрика accuracy

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

где tp — true positive (модель сработала правильно на «положительной» паре, tn — true negative (модель сработала правильно на «отрицательной» паре, fp — false positive (ошибка первого рода, модель приняла «отрицательную» пару за «положительную»), fn — false negative (ошибка второго рода, модель приняла «положительную» пару за отрицательную»).

## **3.2 Тестирование и оценка качества результатов**

При построении простой модели для классификации символов с использованием свёрточной нейронной сети была использована следующая архитектура (Рисунок 3.3) (Приложение Б).

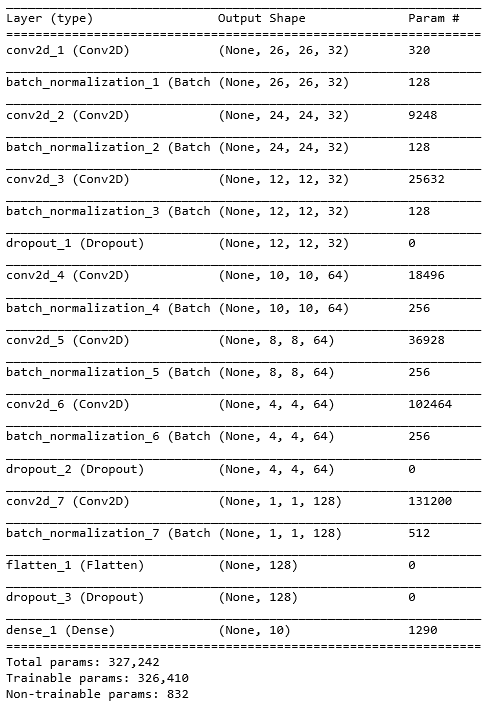


Рисунок 3.3 — Архитектура свёрточной нейронной сети для задачи классификации

Были получены очень высокие показатели точности модели: 99,54 % на тестовой выборке и 99,23 % на валидационной. Во многом это было обусловлено использованием во время обучения специального генератора, который позволял сдвигать изображения, масштабировать их, вращать. Всё это значительно влияет на качество модели, используемой для классификации.

Для сиамской нейросети более важным является именно оценка совпадения символов, а не качество распознавания. Однако стоит также понимать, что именно от точности распознавания зависят те признаки, которые мы выделяем из изображения и используем в дальнейшем для сравнения изображений внутри одной пары. На тестовой выборке удалось получить точность 98,51 %.

Стоит отметить, что модель может быть улучшена путём дообучения с использованием другого оптимизатора. Так, например, можно использовать в паре оптимизаторы Nadam и Adam с возможностью улучшить полученный результат.

## **3.3 Выбор оптимальной архитектуры сети**

Выбор оптимальной архитектуры сети проводился на основе показателей точности, получаемых на дополнительной (валидационной) выборке данных. Такой подход применяется для того, чтобы предотвратить переобучение модели, выполнить настройку гиперпараметров и выбрать наиболее подходящую модель среди всех рассматриваемых. При этом тестовая выборка используется исключительно для оценки качества уже выбранной и обученной модели.

Были добавлены дополнительные свёрточные слои, ответственные за выделение признаков. Для повышения вычислительной устойчивости в модель были включены подвыборочные слои, которые позволяют уменьшать размерность карт свёрточных слоёв.

Архитектура полученной сети представляет собой три последовательных блока свёртки, с количеством карт 64, 128, 256 соответственно, каждый из которых содержит подвыборочный слой. Далее следует полносвязный слой с сигмоидальной функцией активации, который содержит 1024 нейрона. Рассчитывается расстояние (L1 distance) между полученными векторами признаков, и результат передается на полносвязный слой с 1 нейроном и сигмоидальной функцией активации.

При обучении модели были использованы методы обратного вызова (callbacks). Они представляют собой набор функций, которые могут быть применены на различных этапах обучения. Так для того, чтобы остановить процесс обучения модели при отсутствии значимых изменений выбранной метрики качества, использовался EarlyStopping callback. ReduceLROnPlateau callback позволяет уменьшить длину шага градиентного спуска, когда алгоритм достигает пологой поверхности и метрика качества перестаёт улучшаться.

В результате удалось добиться точности 98,8% на тестовой выборке, уменьшив вычислительную сложность модели.

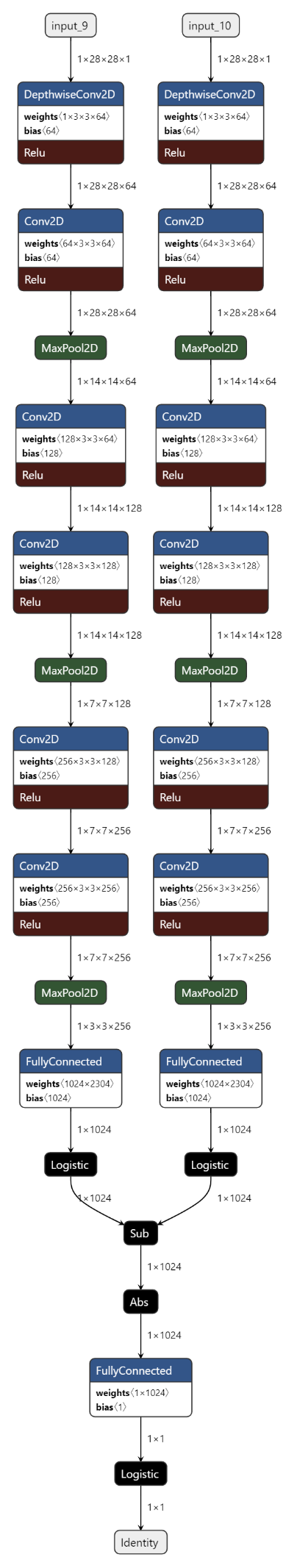


Рисунок 3.4 ­— Архитектура сиамской сети

ВЫВОДЫ

1. Сформированы тестовая, валидационная и тренировочная выборки данных.
2. Реализован прототип системы оценки качества написания рукописных символов.
3. Выбрана оптимальная архитектура сиамской нейронной сети.
4. Проведена оценка качества полученных результатов.

# **РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПЛАТФОРМЫ ANDROID.**

## **4.1 Подготовка разработанной модели**

Для того, чтобы использовать обученную модель на мобильном устройстве, её необходимо сохранить в файле определённого формата в память устройства и затем открыть файл с использованием сторонних интерфейсов.

В качестве библиотеки для работы с моделью была выбрана разработанная Google библиотека для мобильного машинного обучения TensorFlow Lite [[11](#tflite)]. Она позволяет осуществлять быстрый запуск моделей машинного обучения на мобильных устройствах, поддерживает большое количество мобильных платформ с возможностью оптимизации и аппаратного ускорения.

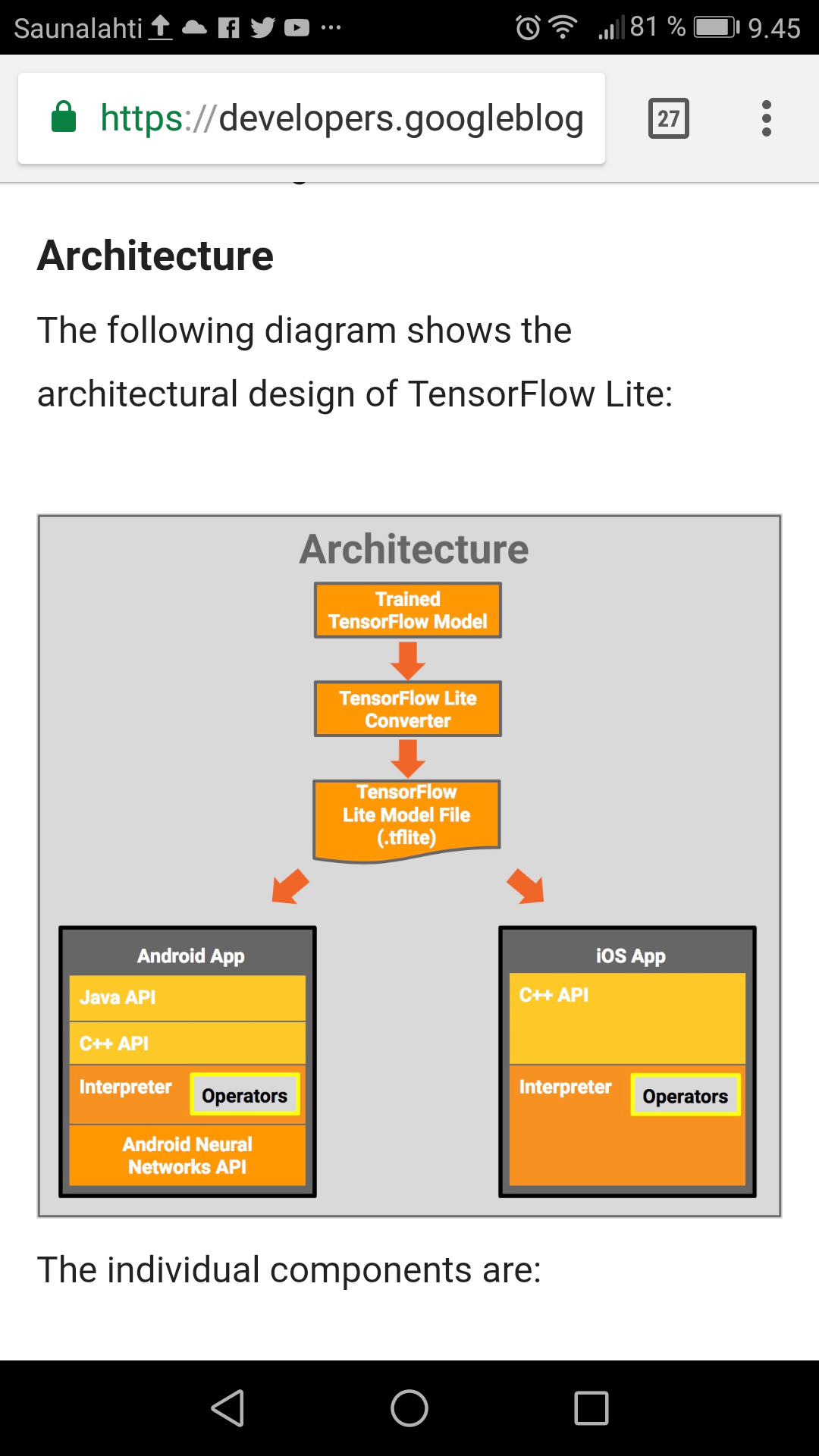


Рисунок 4.1 ­— TensorFlow Lite

Сохранив обученную модель (веса, архитектуру, состояние оптимизатора, метрики качества) в файле формата HDF5, c помощью специального конвертера (TensorFlow Lite Converter) можно перевести модель в формат .tflite и использовать файл с таким расширением для разворачивания на мобильном устройстве. Так как исходные данные обычно не соответствуют формату входных данных, ожидаемых моделью, следующий шаг заключается в преобразовании исходных данных. Для изображений может понадобиться изменить их размер или формат, чтобы они были совместимы с моделью. Далее для выполнения модели используется TensorFlow Lite API, создаётся интерпретатор и выделяется память для входных данных модели (тензоров) (Приложение В). Заключительный шаг связан с вызовом функции, генерирующей предсказание для входных данных, и интерпретацией полученного результата. В случае использование сиамской нейронной сети, для входной пары изображений выходными данными будет вещественное число из интервала (0, 1), которое указывает на вероятность принадлежности входных изображений одному классу (0 – высокая вероятность, 1 – низкая).

В разработанной системе для каждого из 10 входных классов изображений (цифры от 0 до 9) было выбрано по 5 эталонных изображений. Для определения качества написания рукописного символа, формируется пять предсказаний для каждой из пар, состоящий из входного изображений и одного из эталонных изображений выбранного класса, затем предсказания усредняются, и на выходе получается число от 0 до 10 включительно, которое соответствует качеству написания введённого символа (0 – самая низкая оценка, 10 – самая высокая).

## **4.2 Экраны приложения**

Разработанное приложение имеет три экрана. Первый — главное меню, экран на котором можно использовать модель для предсказания качества написания рукописных символов. Второй — история попыток, которая будет содержать в себе попытки пользователя по улучшению качества написания. Третий — сведения о модели, предоставляет краткую информацию об архитектуре сиамской нейронной сети, использованной для предсказания.

### **4.2.1 Главное меню**

Запустив приложение, пользователь переходит в главное меню (Рисунок 4.2).

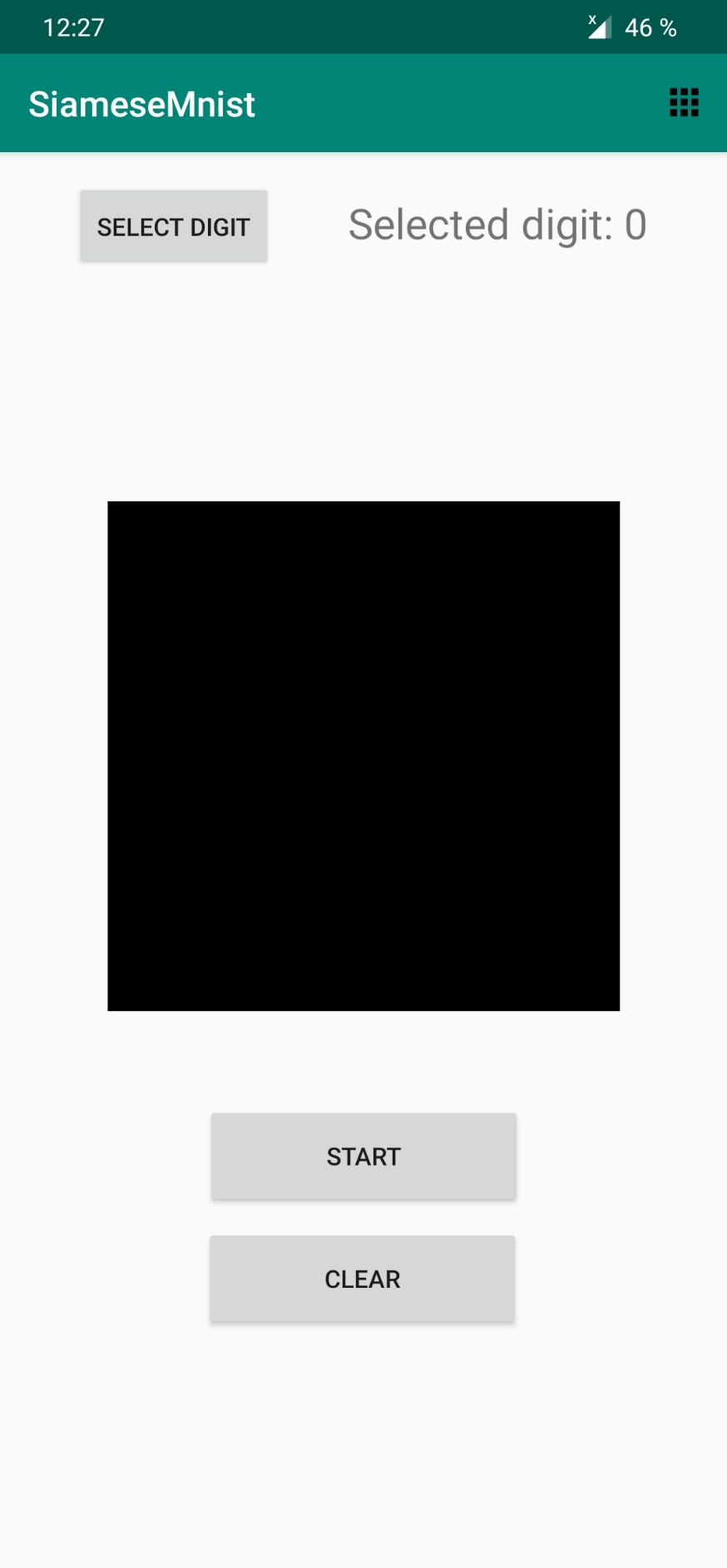


Рисунок 4.2 — Главное меню

При нажатии на кнопку «Настройки» (в правом верхнем углу) появляются два всплывающих пункта («History» и «About»), которые позволяют перейти на соответствующие экраны.

Кнопка «Select digit» позволяет выбрать из всплывающего диалога интересующую цифру для проверки качества написания. По умолчанию выбрана цифра 0. После выбора нужной цифры и нажатия кнопки «Ок», автоматически обновиться значение поля «Selected digit».

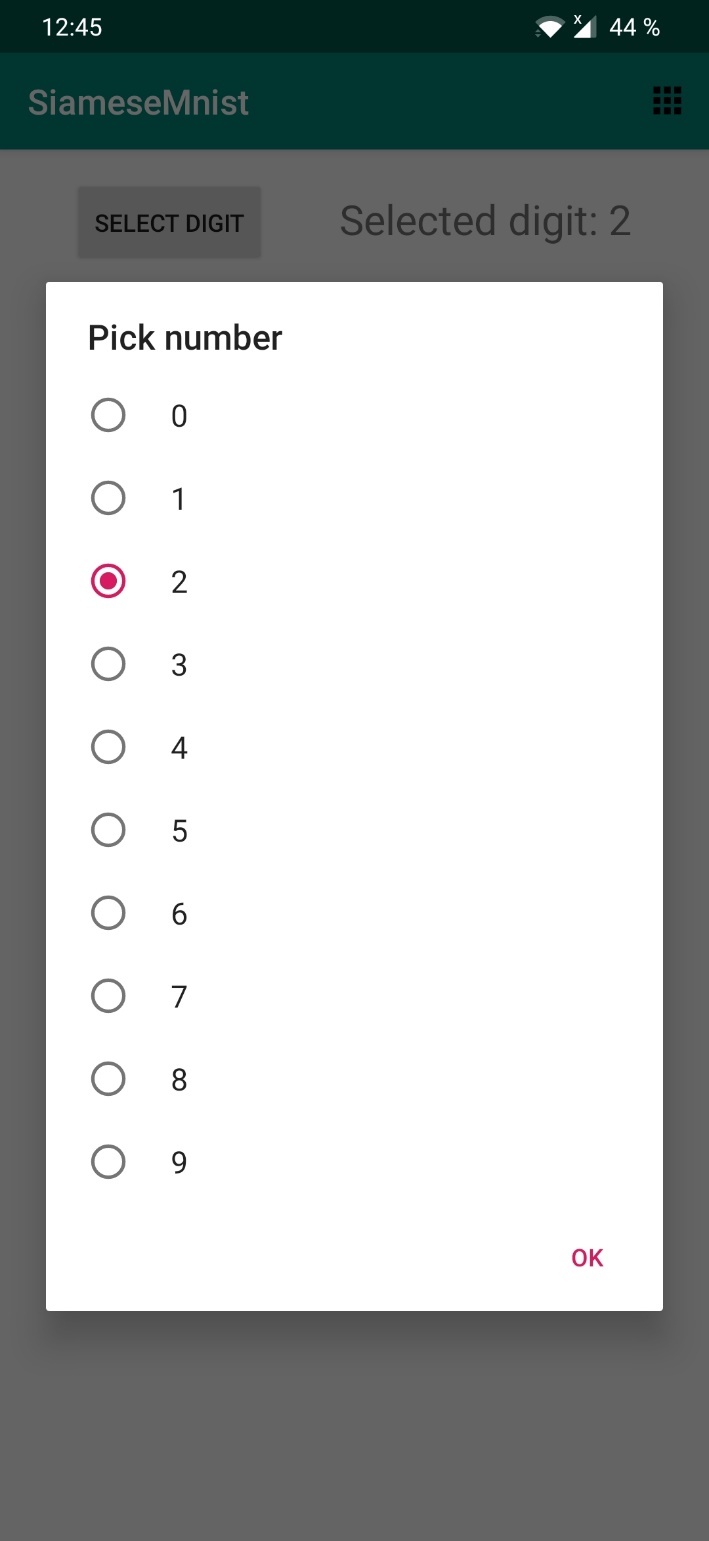


Рисунок 4.3 — Выбор цифры

По середине экрана расположен элемент CustomView, который наследуется от View (Приложение Г). Он позволяет с помощью пальца нарисовать цифру и вернуть полученное изображение в формате, подходящем для модели.

При нажатии на кнопку «Start» изображение с View-элемента вместе с выбранными эталонными изображениями передается в метод predict модели, для формирования предсказания. Результат (оценка качества написания) будет выведен в поле «Score», расположенном над элементом View.

Полученный результат вместе с рукописным символом сохраняется в базу данных SQLite, в которой имеется таблица History для хранения результатов предсказания модели.

Кнопка «Clear» предназначена для очистки View-элемента.



Рисунок 4.4 — Оценка качества написания рукописного символа

### **4.2.2 История попыток**

На данном экране используется RecyclerView, который отображает список сохранённых в базе данных попыток. Каждая запись в базе данных содержит рукописный символ, выбранную цифру и оценку качества написания. С помощью утилиты ItemTouchHelper реализована возможность удаления элементов из списка и базы данных по свайпу вправо [[14](#itemtouch)].

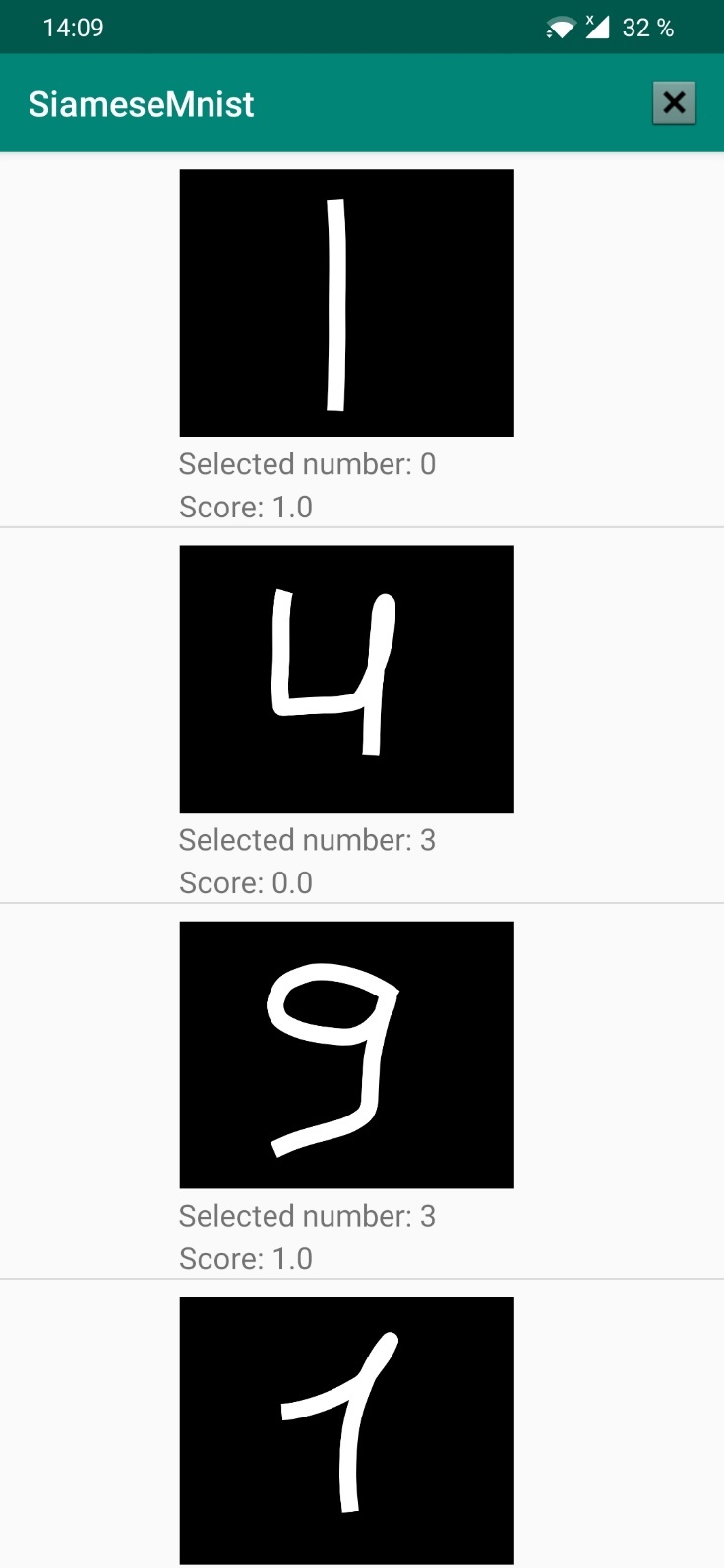


Рисунок 4.5 — История попыток

### **4.2.3 Сведения о модели**

На этом экране расположен заголовок с названием разработанной системы, а также ImageView-элемент, который отображает архитектуру использованной модели.

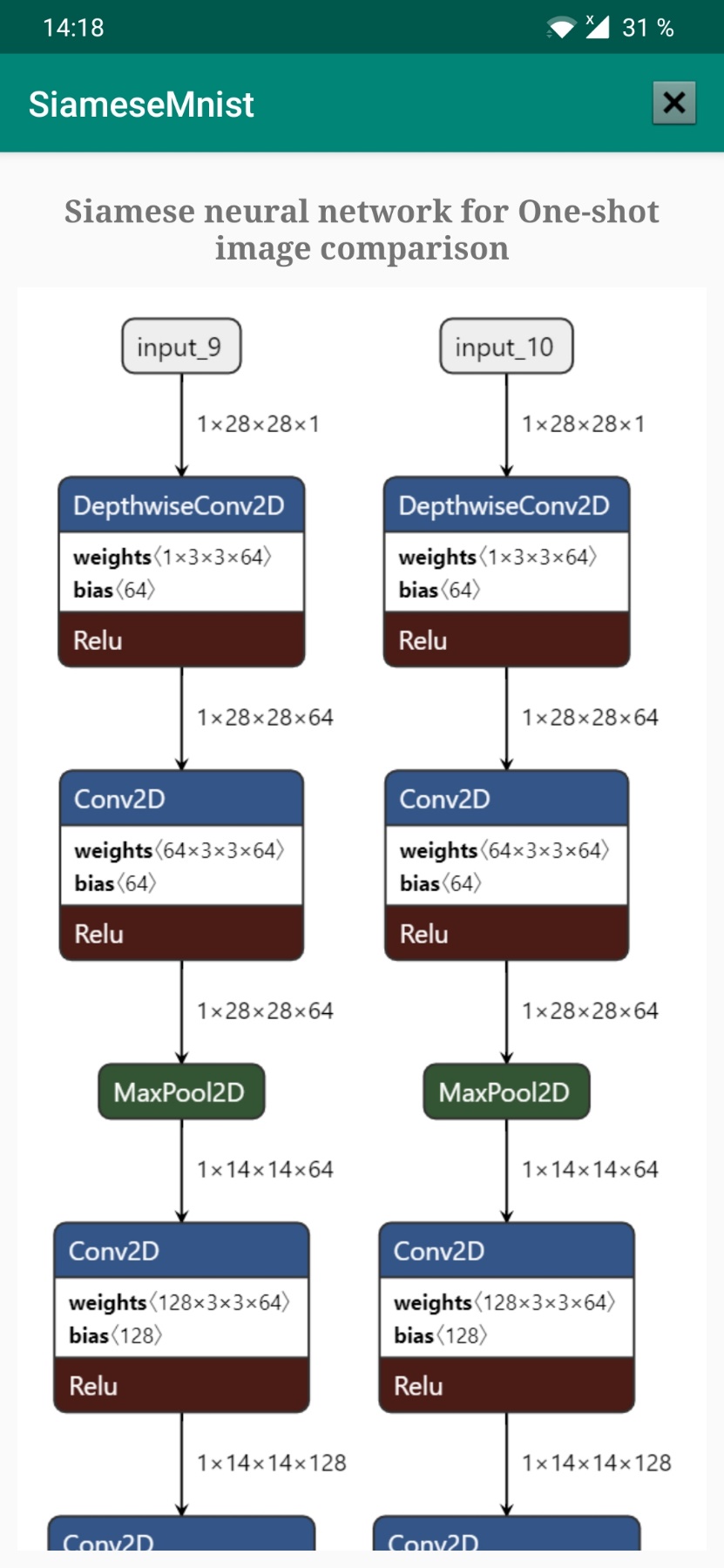


Рисунок 4.6 — Сведения о модели

ВЫВОДЫ

1. Обученная модель сконвертирована в формат, пригодный для использования на мобильном устройстве.
2. В приложение добавлены файл модели и необходимые для формирования предсказаний изображения-эталоны.
3. Разработана архитектура Android приложения.
4. Реализована система оценки качества написания рукописных символов для платформы Android.

# **ЗАКЛЮЧЕНИЕ**

Для оценки качества написания рукописных символов рассмотрена технология One-shot обучения, позволяющая производить сравнение изображений и оценку соответствия эталону. В основу указанной технологии был положен принцип использования сиамской нейронной сети, составной частью которой является свёрточная нейронная сеть, ответственная за извлечение признаков из изображений. Использованный подход позволяет получать более точные результаты по сравнению со многими классическими методами, а особенности архитектуры, такие как параллельная свёртка, дают возможность организовать вычисления более эффективно.

Разработанное мобильное приложение может быть успешно применено на практике для обучения. Оно позволяет развивать зрительную память, тренировать мелкую моторику, отрабатывать навыки написания рукописных символов.

В ходе выполнения работы были рассмотрены и решены следующие задачи:

1. Изучены подходы к сравнению изображений.
2. Описаны преимущества и недостатки наиболее распространённых методов сравнения изображений.
3. Описана архитектура нейронных сетей.
4. Рассмотрена возможность использования нейронных сетей для сравнения изображений.
5. Описана технология One-shot обучения, а также использование сиамских нейронных сетей в рамках этой технологии.
6. Разработана архитектура сиамской нейронной сети.
7. Реализован прототип системы оценки качества написания рукописных символов.
8. Выбрана оптимальная архитектура сиамской нейронной сети.
9. Проведены тестирование полученного прототипа и оценка качества результатов.
10. Обученная модель преобразована в формат, который позволяет использовать её на мобильном устройстве.
11. Разработана архитектура Android приложения.
12. Реализована система оценки качества написания рукописных символов для платформы Android.

Для подготовки модели использовался язык Python.

Для реализации приложения использовалась интегрированная среда разработки Android Studio.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

* 1. Как работает нейронная сеть [Electronic resource]: *https://neurohive.io/ru/osnovy-data-science/osnovy-nejronnyh-setej-algoritmy-obuchenie-funkcii-aktivacii-i-poteri/*
  2. Свёрточная нейронная сеть [Electronic resource]: [*https://habr.com/ru/post/348000/*](https://habr.com/ru/post/348000/)
  3. Piotr Indyk, Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality — STOC ’98 Proceedings of thirtieth annual ACM symposium on Theory of computing. — 1998. — c. 604-613.
  4. Michael J. Swain, Dana H. Ballard. Color Indexing — International Journal of Computer Vision — 1991. — с. 11-32.
  5. David G. Lowe. Object recognition from local scale-invariant features — Proceedings of the Seventh IEEE International Conference on Computer Vision — 1999. — c. 1150-1157.
  6. One-Shot Learning for Face Recognition [Electronic resource]: [*https://machinelearningmastery.com/one-shot-learning-with-siamese-networks-contrastive-and-triplet-loss-for-face-recognition/*](https://machinelearningmastery.com/one-shot-learning-with-siamese-networks-contrastive-and-triplet-loss-for-face-recognition/)
  7. One Shot learning, Siamese networks and Triplet Loss with Keras [Electronic resource]: [*https://medium.com/@crimy/one-shot-learning-siamese-networks-and-triplet-loss-with-keras-2885ed022352*](https://medium.com/@crimy/one-shot-learning-siamese-networks-and-triplet-loss-with-keras-2885ed022352)
  8. Siamese Neural Networks for One-shot Image [Electronic resource]: *https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf*
  9. Similarity Learning with Convolutional Neural Networks [Electronic resource]: *http://slazebni.cs.illinois.edu/spring17/lec09\_similarity.pdf*
  10. Understanding Neural Networks [Electronic resource]: [*https://towardsdatascience.com/understanding-neural-networks-19020b758230*](https://towardsdatascience.com/understanding-neural-networks-19020b758230)
  11. TensorFlow Lite inference [Electronic resource]:[*https://www.tensorflow.org/lite/guide/inference*](https://www.tensorflow.org/lite/guide/inference)
  12. TensorFlow Lite converter [Electronic resource]: [*https://www.tensorflow.org/lite/convert*](https://www.tensorflow.org/lite/convert)
  13. From Keras to Android with TensorFlow Lite [Electronic resource]:[*https://medium.com/@vvalouch/from-keras-to-android-with-tensorflow-lite-7581368aa23e*](https://medium.com/@vvalouch/from-keras-to-android-with-tensorflow-lite-7581368aa23e)
  14. ItemTouchHelper [Electronic resource]: [*https://developer.android.com/reference/androidx/recyclerview/widget/ItemTouchHelper.SimpleCallback*](https://developer.android.com/reference/androidx/recyclerview/widget/ItemTouchHelper.SimpleCallback)

# **ПРИЛОЖЕНИЯ**

## *Приложение А*

**class** SiameseNetwork:  
 margin = 1  
  
 **def** \_\_init\_\_(self, input\_shape):   
 img1 = Input(shape=input\_shape)  
 img2 = Input(shape=input\_shape)   
   
 model = Sequential()  
   
 model.add(Conv2D(64, (3,3),

activation=**'relu'**,

input\_shape=input\_shape,   
 data\_format=**'channels\_first'**,

kernel\_initializer=**'random\_uniform'**,   
 bias\_initializer=**'zeros'**))

model.add(MaxPooling2D((2,2)))  
 model.add(Conv2D(128, (3,3),

activation=**'relu'**,

kernel\_initializer=**'random\_uniform'**,

bias\_initializer=**'zeros'**))  
   
 model.add(MaxPooling2D())  
 model.add(Dropout(.25))  
 model.add(Conv2D(128, (2,2),

activation=**'relu'**,

kernel\_initializer=**'random\_uniform'**,

bias\_initializer=**'zeros'**))  
   
 model.add(MaxPooling2D((2,2)))  
 model.add(Dropout(.25))  
 model.add(Flatten())  
 model.add(Dense(1024, activation=**'sigmoid'**))   
   
 features1 = model(img1)  
 features2 = model(img2)  
   
 L1\_layer = Lambda(**lambda** vectors:K.abs(vectors[0] - vectors[1]))   
 L1\_distance = L1\_layer([features1, features2])  
 prediction = Dense(1,activation=**'sigmoid'**)(L1\_distance)  
 self.model = Model(inputs=[img1,img2],outputs=prediction)  
   
 @staticmethod  
 **def** contrastive\_loss(y\_true, y\_pred, margin=margin):  
 **return** K.mean(y\_true \* K.square(y\_pred) + (1 - y\_true) \* K.square(K.maximum(margin - y\_pred, 0)))  
  
 **def** make\_prediction(self, x\_test):  
 **return** self.model.predict([x\_test[:,0], x\_test[:,1]])  
  
 **def** fit(self, x\_train, y\_train, batch\_size, epochs):  
 ndm = Nadam()  
 self.model.compile(loss=self.contrastive\_loss, optimizer=ndm)  
 img\_1 = x\_train[:,0]  
 img\_2 = x\_train[:,1]  
 self.model.fit([img\_1, img\_2], y\_train, validation\_split=.25, batch\_size=batch\_size, verbose=1, epochs=epochs)

## *Приложение Б*

**def** build\_model():  
   
 model = Sequential()  
 model.add(Conv2D(32, kernel\_size=3,

activation=**'relu'**,

input\_shape = (28,28,1)))  
 model.add(BatchNormalization())   
 model.add(Conv2D(32, kernel\_size=3,

activation=**'relu'**))  
 model.add(BatchNormalization())   
 model.add(Conv2D(32, kernel\_size=5,

strides=2,

padding=**'same'**,

activation=**'relu'**))  
 model.add(BatchNormalization())  
 model.add(Dropout(0.25))  
   
 model.add(Conv2D(64, kernel\_size=3,

activation=**'relu'**))  
 model.add(BatchNormalization())  
 model.add(Conv2D(64, kernel\_size=3,

activation=**'relu'**))  
 model.add(BatchNormalization())  
 model.add(Conv2D(64, kernel\_size=5,

strides=2,

padding=**'same'**,

activation=**'relu'**))  
 model.add(BatchNormalization())   
 model.add(Dropout(0.25))  
   
 model.add(Conv2D(128, kernel\_size=4, activation=**'relu'**))  
 model.add(BatchNormalization())  
 model.add(Flatten())  
 model.add(Dropout(0.5))  
 model.add(Dense(10, activation=**'softmax'**))  
  **return** model

## *Приложение В*

**class** Model {  
  
 **static** MappedByteBuffer loadModelFile(Context context) **throws** IOException {  
 String MODEL\_ASSETS\_PATH = **"23\_new\_model.tflite"**;  
 AssetFileDescriptor assetFileDescriptor = context.getAssets().openFd(MODEL\_ASSETS\_PATH) ;  
 FileInputStream fileInputStream = **new** FileInputStream( assetFileDescriptor.getFileDescriptor() ) ;  
 FileChannel fileChannel = fileInputStream.getChannel() ;  
 **long** startoffset = assetFileDescriptor.getStartOffset() ;  
 **long** declaredLength = assetFileDescriptor.getDeclaredLength() ;  
 **return** fileChannel.map( FileChannel.MapMode.***READ\_ONLY*** , startoffset , declaredLength ) ;  
 }  
  
 **static float** predict(Interpreter interpreter, Context context, **int** selectedNumber, ByteBuffer inputImage1, ByteBuffer inputImage2, Bitmap scaledBitmap){  
 scaledBitmap = Bitmap.*createScaledBitmap*( scaledBitmap , 28 , 28 , **false** );  
 **int**[] imagePixels1 = **new int**[28 \* 28];  
 **int**[] imagePixels2 = **new int**[28 \* 28];  
 ImageUtils.*convertBitmapToByteBuffer*(scaledBitmap, inputImage2, imagePixels2);  
 **float** result = 1;  
 Bitmap sample = BitmapFactory.*decodeResource*(context.getResources(),R.drawable.***image00***);  
  
 **for** (**int** i=0; i<5;i++){  
 String sampleName = **"image"**+String.*valueOf*(selectedNumber)+String.*valueOf*(i)+**".png"**;  
 **int** resId = context.getResources().  
 getIdentifier(sampleName.split(**"\\."**)[0], **"drawable"**, context.getApplicationInfo().**packageName**);  
 sample = BitmapFactory.*decodeResource*(context.getResources(), resId);  
 sample = Bitmap.*createScaledBitmap*( sample , 28 , 28 , **false** ) ;  
 ImageUtils.*convertBitmapToByteBuffer*(sample, inputImage1, imagePixels1);  
 Object[] inputs = { inputImage1 , inputImage2};  
 @SuppressLint(**"UseSparseArrays"**) Map<Integer, Object> outputs = **new** HashMap<>();  
 outputs.put(0, **new float**[1][1] );  
 interpreter.runForMultipleInputsOutputs( inputs , outputs ) ;  
 **float**[][] similarity = ( **float**[][] )outputs.get( 0 ) ;  
 **if** (similarity[0][0]< result)  
 result = similarity[0][0];  
 }  
 **return** result;  
 }  
}

## *Приложение Г*

**public class** CustomView **extends** View {  
  
 **private** Canvas **canvas**;  
 **private** Paint **paint**;  
 **private** Path **path**;  
 **public** Bitmap **bitmap**;  
 **private** Context **context**;  
  
 **private float x**, **y**;  
 **private static final float *TOUCH\_TOLERANCE*** = 4;  
  
 **public** CustomView(Context context, AttributeSet attrs) {  
 **super**(context, attrs);  
 **this**.**context** = context;  
 init();  
 }  
  
 **private void** init() {  
 DisplayMetrics dm = **context**.getResources().getDisplayMetrics();  
 **bitmap**= Bitmap.*createBitmap*(dm.**widthPixels**, dm.**heightPixels**, Bitmap.Config.***ARGB\_8888***);  
 **canvas** = **new** Canvas(**bitmap**);  
 **path** = **new** Path();  
 **paint** = **new** Paint();  
  
 **paint**.setAntiAlias(**true**);  
 **paint**.setDither(**true**);  
 **paint**.setColor(Color.***WHITE***);  
 **paint**.setStyle(Paint.Style.***STROKE***);  
 **paint**.setStrokeJoin(Paint.Join.***ROUND***);  
 **paint**.setStrokeWidth(48f);  
 }  
  
 @Override  
 **protected void** onDraw(Canvas canvas) {  
 **super**.onDraw(canvas);  
 canvas.drawColor(Color.***BLACK***);  
 canvas.drawBitmap(**bitmap**, 0, 0, **paint**);  
 canvas.drawPath(**path**, **paint**);  
 }  
  
 @Override  
 **protected void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh) {  
 **super**.onSizeChanged(w, h, oldw, oldh);  
 **bitmap** = Bitmap.*createBitmap*(w, h, Bitmap.Config.***ARGB\_8888***);  
 **canvas** = **new** Canvas(**bitmap**);  
 }  
  
 @SuppressLint(**"ClickableViewAccessibility"**)  
 @Override  
 **public boolean** onTouchEvent(MotionEvent event) {  
 **float** x = event.getX();  
 **float** y = event.getY();  
  
 **switch** (event.getAction()) {  
 **case** MotionEvent.***ACTION\_DOWN***:  
 touch\_start(x, y);  
 invalidate();  
 **break**;  
 **case** MotionEvent.***ACTION\_MOVE***:  
 touch\_move(x, y);  
 invalidate();  
 **break**;  
 **case** MotionEvent.***ACTION\_UP***:  
 touch\_up();  
 invalidate();  
 **break**;  
 }  
 **return true**;  
 }  
  
 **private void** touch\_start(**float** x, **float** y) {  
 **path**.reset();  
 **path**.moveTo(x, y);  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
 **private void** touch\_move(**float** x, **float** y) {  
 **float** dx = Math.*abs*(x - **this**.**x**);  
 **float** dy = Math.*abs*(y - **this**.**y**);  
 **if** (dx >= ***TOUCH\_TOLERANCE*** || dy >= ***TOUCH\_TOLERANCE***) {  
 **path**.quadTo(**this**.**x**, **this**.**y**, (x + **this**.**x**)/2, (y + **this**.**y**)/2);  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
 }  
 **private void** touch\_up() {  
 **path**.lineTo(**x**, **y**);  
 **canvas**.drawPath(**path**, **paint**);  
 **path**.reset();  
 }  
  
 **public void** clear() {  
 **path**.reset();  
 **bitmap** = Bitmap.*createBitmap*(**bitmap**.getWidth(), **bitmap**.getHeight(), Bitmap.Config.***ARGB\_8888***);  
 **canvas** = **new** Canvas(**bitmap**);  
 invalidate();  
 }  
  
 **public** Bitmap getBitmap(**int** width, **int** height) {  
 **return** Bitmap.*createScaledBitmap*(**bitmap**, 28, 28, **false**);  
 }  
}