МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Математический факультет

Кафедра компьютерной безопасности и прикладной алгебры

**ДИПЛОМНАЯ РАБОТА**

Реализация системы распознавания 3D лиц с использованием нейронных сетей

|  |  |
| --- | --- |
| ДОПУСТИТЬ К ЗАЩИТЕ  Заведующий кафедрой,  Кандидат физико-математических наук,  Доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ручай А.Н.    «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г. | Выполнил студент  Токарев Игорь Вячеславович  группы МК-601  очной формы обучения  направления подготовки 10.05.01 компьютерная безопасность  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

|  |  |
| --- | --- |
| Нормоконтроль  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шалагинов Л.В.    «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г. | Научный руководитель  Ручай Алексей Николаевич  Заведующий кафедры КБиПА  Кандидат физико-математических наук  Доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г. |

Челябинск

2020

Оглавление

[Введение 3](#_Toc32864697)

[1. Теоретическая часть 4](#_Toc32864698)

[1.1 Наборы данных 4](#_Toc32864699)

[1.2. Аугментация данных 10](#_Toc32864700)

[1.3. Задача идентификации по 3D изображению лица. 12](#_Toc32864701)

[1.4. Выбор архитектуры нейронной сети. 13](#_Toc32864702)

[2. Практическая часть 18](#_Toc32864703)

[2.1. Выполнение процесса идентификации 18](#_Toc32864704)

[2.2. Реализация процесса дообучения 19](#_Toc32864705)

[2.3. Создание размеченных данных. 20](#_Toc32864706)

[2.4. Тестирование 22](#_Toc32864707)

[Заключение 24](#_Toc32864708)

[Список литературы 25](#_Toc32864709)

[Приложение 27](#_Toc32864710)

# **Введение**

Задача идентификации по биометрическим признакам сегодня имеет большое значение в задачах компьютерной безопасности. Особенно, важным является при разграничении доступа к важным хранилищам информации или субъектам, которые должны быть доступны лишь ограниченному кругу лиц.

Целью данной работы является создание системы идентификации по 3D изображению лица человека.

Для реализации системы были поставлены следующие задачи:

1. Сбор данных;
2. Обработка полученных данных;
3. Дообучение нейронной сети на данных;
4. Обработка результата.

# **Теоретическая часть**

## **Наборы данных**

Для реализации систем компьютерного зрения на основе первостепенной задачей является сбор подходящих наборов данных. Для применения в поставленной задаче были выбраны следующие наборы:

1. 3DTEC
2. BU-3DFE
3. Bosphorus
4. FaceWarehouse

Опишем каждый из наборов.

**3DTEC**

В базе данных 3DTEC имеется 440 изображения в формате ABS.

Чтобы иметь возможность работать с изображениями, сначала необходимо изменить формат каждого изображения на PLY. Что было получено с использованием языка программирования Python и библиотеки NumPy.

Данная база данных была разработана для использования в задачах по распознаванию близнецов.

3DTEC имеет собственную лицензию.

**BU-3DFE**

В данной базе располагается 2500 изображений 3D лиц более чем 100 человек, которые были созданы для задач распознавания и анимации.

Соотношение изображений мужчин и женщин составляет 56% к 44%. Возраст актёров от 18 до 70 лет. Содержит множество рас:

* Европейцы;
* Африканцы;
* Среднеазиатская;
* Индийцы.

Каждый субъект исполнил 7 различных выражений лица перед 3D-сканером

Изображены следующие виды эмоций:

* Счастье
* Отвращение
* Страх
* Злость
* Удивление
* Грусть
* Нейтральная

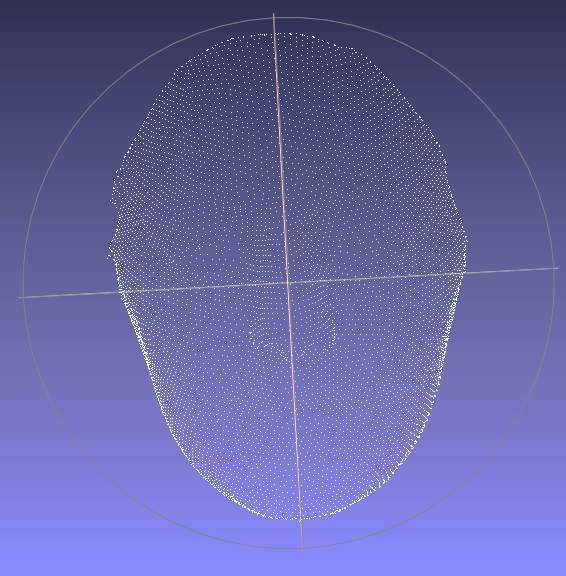
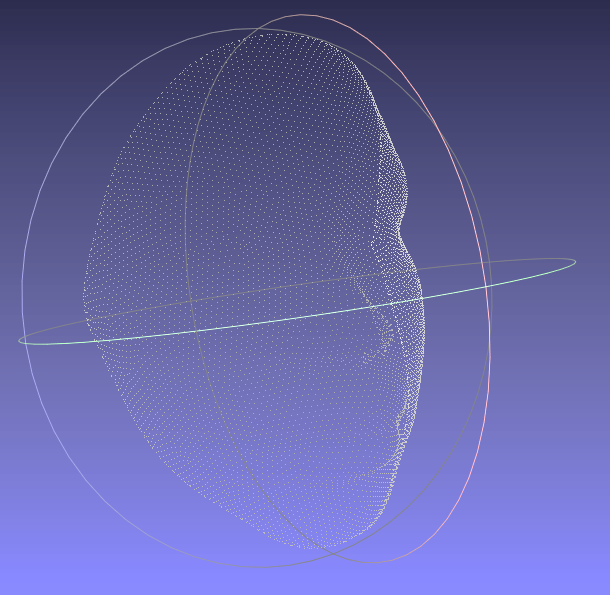


Рис 1. Пример облаков точек, полученных из файлов 3D-изображений библиотеки 3B-FU.

**FaceWarehouse**

Cao Chen, Yanlin Weng, Shun Zhou, Yiying Tong, Kun Zhou: "FaceWarehouse: a 3D Facial Expression Database for Visual Computing", IEEE Transactions on Visualization and Computer Graphics, 20(3): 413-425, 2014

База данных 3D изображений лиц с различными выражениями для создания приложений. Содержит изображения 150 субъектов в возрасте от 7 до 80 лет разных этнических групп. Для получения данных использовалась RGBD камера Kinect.

Данные для каждого человека хранятся в отдельной директории. В каждой директории "Tester\_xx" хранятся RGBD данные, для каждого изображения отмечены 74 особые точки, созданы меши, также представлены некоторые дополнительные данные.

* Tester\_1: folder name with person 1
  + TrainingPose
    - pose\_0.pose - pose\_20.pose: данные от Kinect RGBD камеры
    - pose\_0.obj - pose\_20.obj:реконструированные меши.
    - pose\_0.land - pose\_20.land: отмеченные 2D точки для первого кадра в директории.
  + Blendshape
    - shape.bs: двоичный файл с выражением blendshape
    - shape\_0.obj - shape\_46.obj: выражения blendshapes

Примеры возможных применений:

* перенос эмоций.
* создание анимаций изображений в реальном времени.
* создание новых эмоций для отдельного персонажа используя совмещение имеющихся

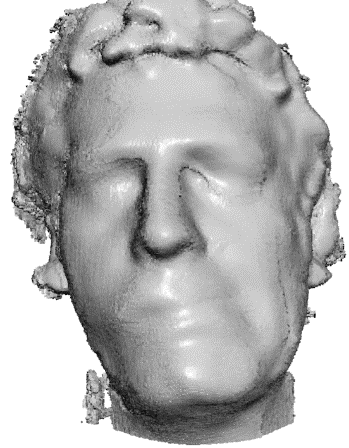
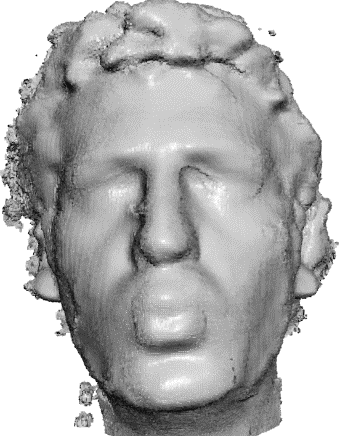


Рис 2. Примеры изображений набора данных FaceWarehouse

**Bosphorus**

База данных BOSPHORUS специально создана чтобы проводить исследования в области 3D и 2D обработки изображений лиц людей, а также 3D реконструкции лиц. База состоит из изображений 105 человек, и всего 4666 изображений 3D облаков. В базе содержатся не только нормализованные изображения, но также:

1. Выражение эмоций – до 35 для отдельного человека.
2. Подсчитанные лицевые движения.
3. Третья часть базы создана с привлечением профессиональных актёров.

Различные уровни поворота лица.

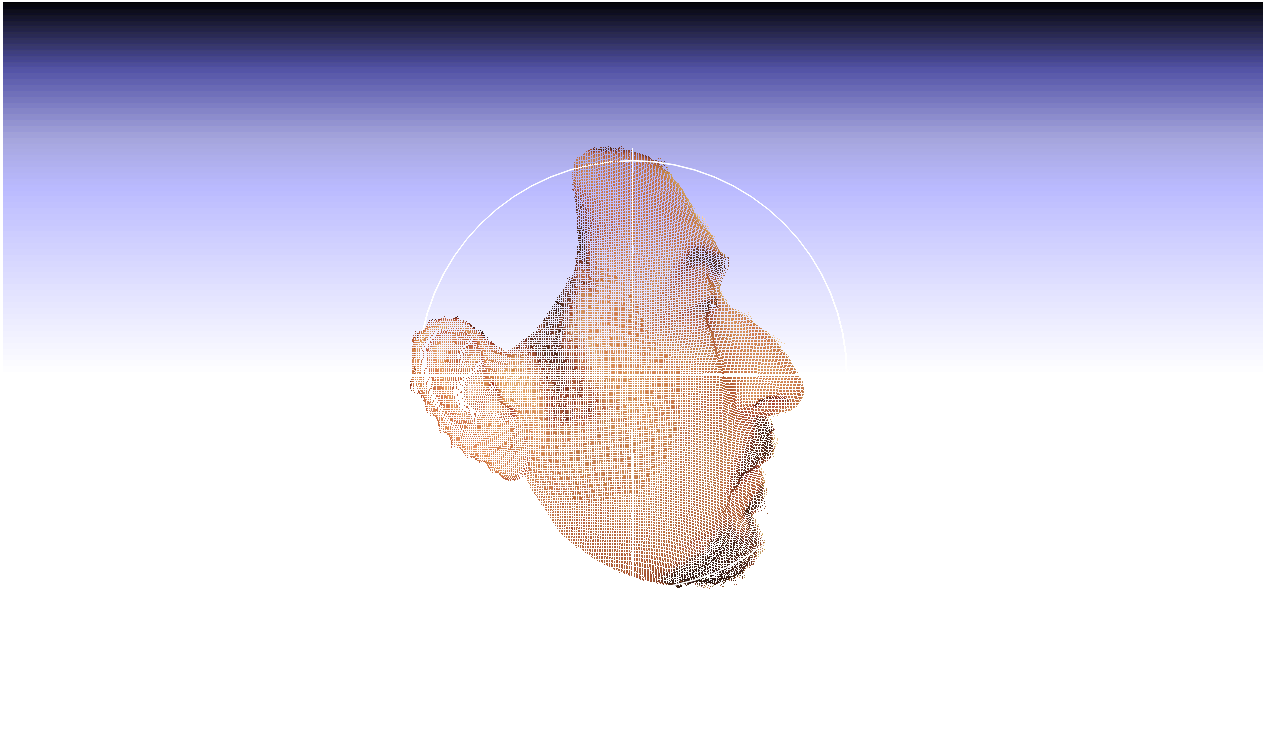


Рис 3. Пример изображения bs002\_YR\_L90\_0.ply до процедуры поворота.

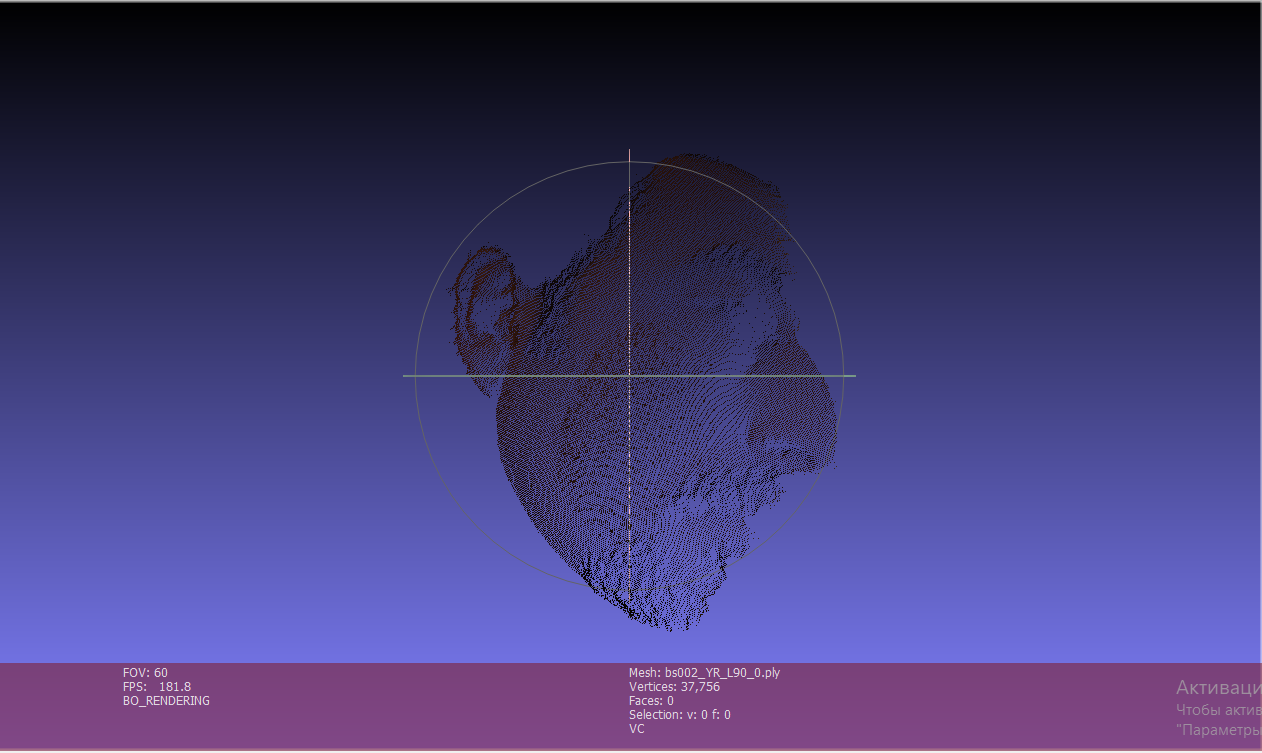


Рис 4. Пример изображения bs002\_YR\_L90\_0.ply после процедуры поворота.

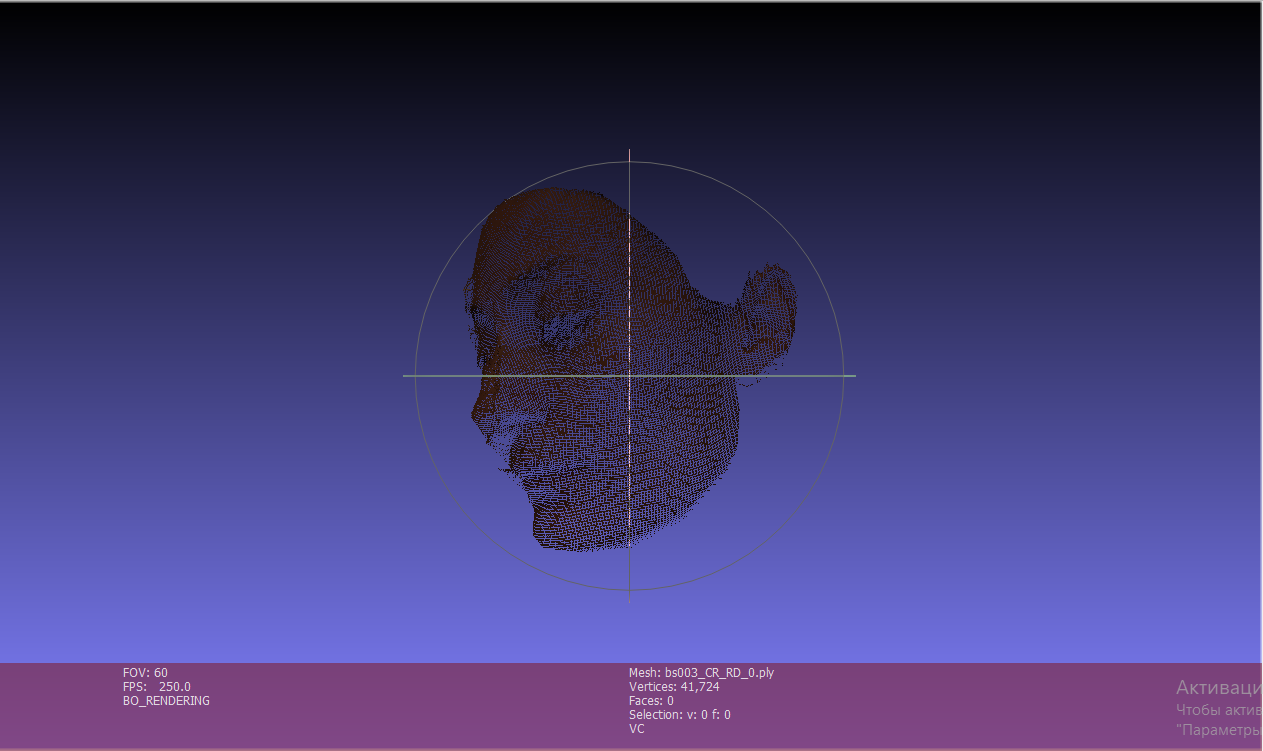


Рис 5. Пример изображения bs003\_CR\_RD\_0.ply после процедуры поворота.

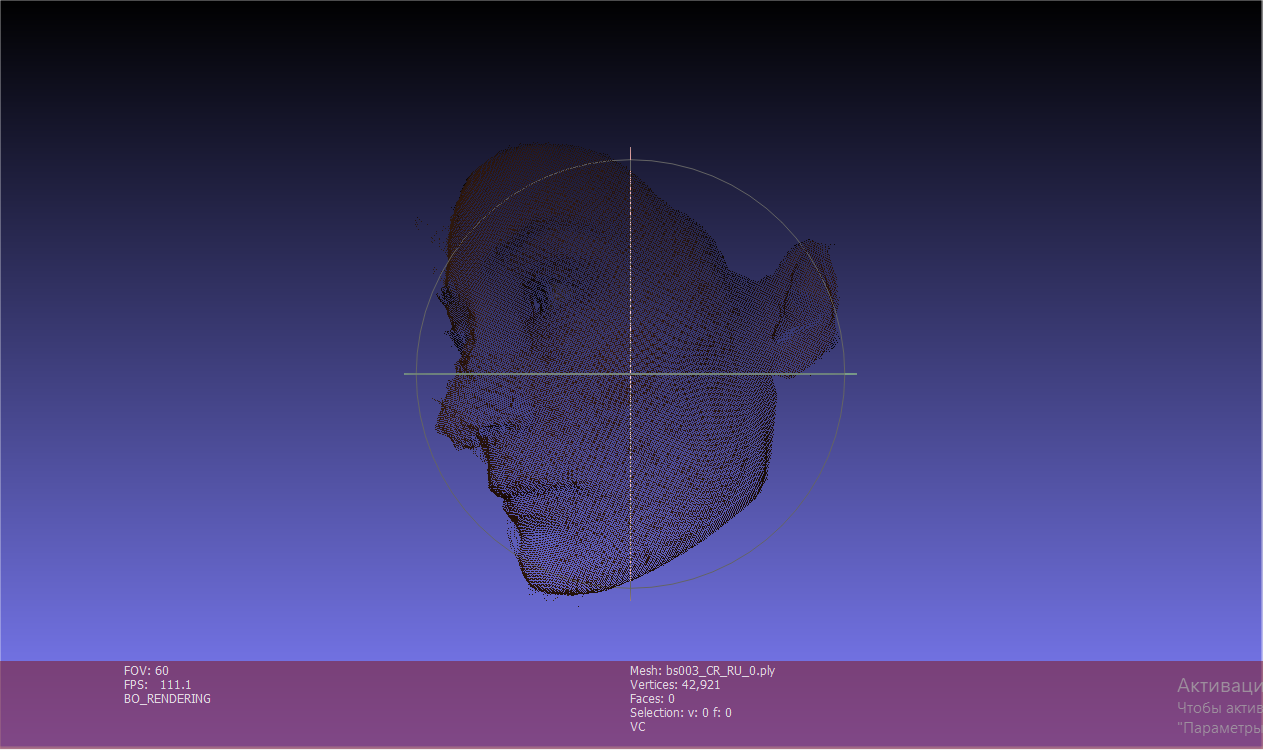


Рис 6. Пример изображения bs003\_CR\_RD\_0.ply после процедуры поворота.

По результатам предварительного исследования представленных наборов, можно заключить, что наиболее подходящим является Bosphorus, так как обладает изначально впечатляющей вариативностью изображений, со множеством дополнительных параметров.

Приведём сводную таблицу с характеристиками каждого из наборов:

|  |  |  |  |
| --- | --- | --- | --- |
| Набор | Качество представленных данных | Количество изображений | Необходимый уровень предобработки |
| Bosphorus | Высокий | 2622 | Низкий |
| 3DTEC | Низкий | 440 | Высокий |
| BU-3DFE | Средний | 5000 | Средний |
| FaceWarehouse | Высокий | 2850 | Высокий |

**Таблица 1.**

**VGG Face 2**

В качестве исходного набора данных для протестированных архитектур нейронных сетей используется VGG Face 2.

VGG Face 2 собран научной группой Оксфордского Университета (University of Oxford). Набор содержит более 3.3 миллионов изображений лиц более чем 9000 субъектов. Размеры изображений варьируются. Всем изображениями соответствуют различные методы аугментации, такие как:

* различные положения лица,
* присутствие посторонних лиц,
* использование дополнительных средств сокрытия элементов лица (очки, шляпы и т.д.),
* изменение фона,
* различие тона, яркости изображения.

** **

*Рис. 7 Примеры изображения лица субъекта в наборе VGG Face 2.*

** ** 

*Рис. 8 Примеры изображения лица субъекта в наборе VGG Face 2.*

## **Аугментация данных**

Под аугментацией будем понимать совокупность методов искусственного увеличения набора исходных данных для обучения нейронной сети.

Изначально мы имеем 10912 изображений. Это очень небольшое количество для задачи переобучения такой большой нейронной сети как ResNet-50. В связи с чем возникает необходимость в проведении аугментации исходных данных.

В задачах компьютерного зрения чаще всего используются следующие методы аугментации:

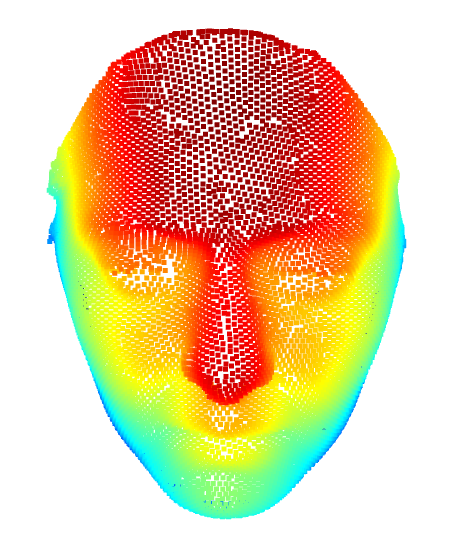
* афинные преобразования (такие, как повороты);
* нанесение шумов на 2D-изображение;
* нанесение шумов на 3D-изображения облака точек;
* нелинейные преобразования;
* искусственное нанесение на облака точек искажений лица.

В данной работе решено применить методы афинных преобразований.

Для каждого исходного облака точек применяется поворот вдоль каждой из осей координат с шагом в 2 градусf, в дапазоне от -30, до 30 градусов.

Таким образом размерность набора данных можно повысить в 90 раз.

Примеры полученных изображений:



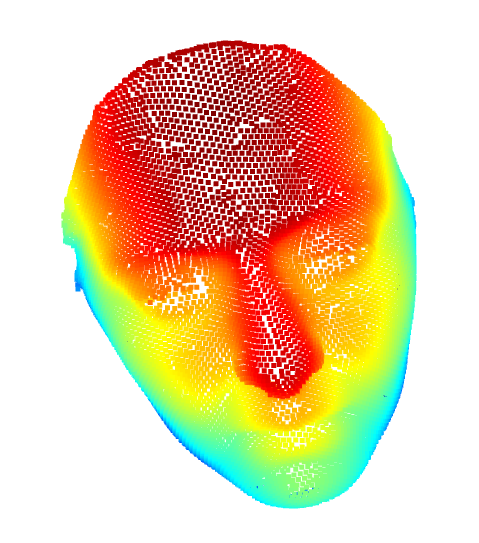


Рис. 9 Примеры агументированных изображений.

## **Задача идентификации по 3D изображению лица.**

Назовём режимом идентификации следующий эксперимент. Создадим два набора данных – Gallery и Probe.

Стандартным способом проведения идентификации является расположения в Gallery нормализованных изображений, а в Probe произвольного набора изображений.

На выходе нейронной сети получается набор признаков. Набор признаков представляется в виде вектора, длина которого определяется конкретной архитектурой нейронной сети.

Например:

* для сети VGG16 набор признаков слоя fc7 имеет длину 4096;
* для сети ResNet-50 вектор признаков слоя classifier имеет длину 8631.

Для того, чтобы сравнивать похожесть изображений в задаче идентификации мы используем косинусное расстояние, вычисляемое для двух векторов признаков, получаемых на выходе нейронной сети.

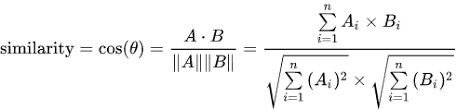


Рис 10. Косинусное расстояние между векторами A и B.

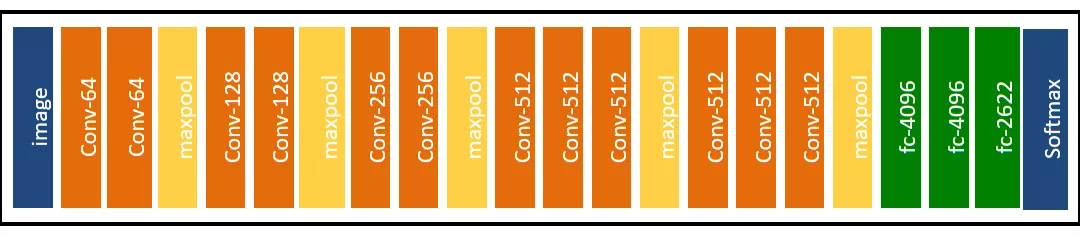
Подобный подход позволяет освободиться от создания классификатора для фиксированного набора изображений и вычислять сходство между произвольным количеством субъектов идентификации.

В нормализованной группе находились изображения нормальные изображения для каждого из объектов идентификации. Затем в 105 экспериментах для каждого отдельного объекта выбирались оставшиеся, кроме нормального, изображения и проводилось их сравнение нейронной сетью. На выходе нейронной сети мы получаем вероятности совпадения изображения из нормализованной и тестовой групп.

## **Выбор архитектуры нейронной сети.**

В предыдущих работах была исследована нейронная сеть с VGG16 обученнная на наборе данных VGGFace.

Данная нейронная сеть является сверхглубокой. Имеет 22 слоя, включая полносвязные и свёрточные слои. На вход сети подаётся изображение размерности 224\*224 пикселей.



Архитектура ResNet также является сверхглубокой нейронной сетью. Различные модификации могут включать различные наборы слоёв. Отличием ResNet от других свёрточных нейронных сетей является наличие специальных Residual слоёв.

Рис. 4. Архитектура нейронной сети VGG-Face.

В данной работе используется архитектура ResNet-50. Д

Схематиченое изображение Residual слоя изображено на рисунке Рис. 2. В задачах классификации изображений архитектуры ResNet неоднократно доказывали свою эффективность и показывая state-of-art результаты.

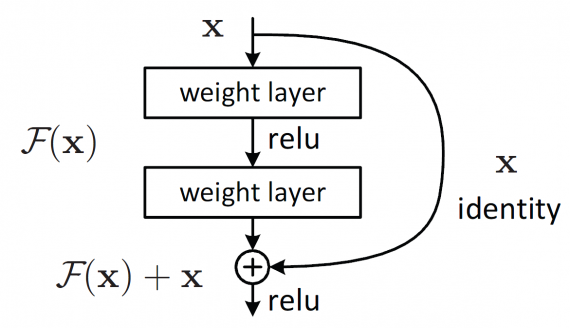


Рис. 11. Схематичное изображение Residual блока. Такой блок состоит из суммы вектора данных X и вектора обработанных данных преобразованием F(X).

Использование ResNet-50 было определено тем, что существуют несколько свободно распространяемых предобученных на VGG Face 2 нейронных сетей данной архитектуры.

Ниже представлена таблица с описанием предобученных архитектур нейронных сетей ResNet-50.

| **Архитектура** | **Размер выходного слоя** | **TAR@FAR = 0.001** | **TAR@FAR = 0.01** | **Фреймворк** |
| --- | --- | --- | --- | --- |
| ResNet-50 | 8631 | 0.891 | 0.947 | Caffe, PyTorch |
| ResNet-50-256D | 256 | 0.898 | 0.956 | Caffe, PyTorch |
| ResNet-50-128D | 128 | 0.904 | 0.956 | Caffe, PyTorch |

Таблица 2.

Каждая из представленных выше моделей имеет 228 слоёв. На рисунке Рис 12 изображена архитектура нейронной сети ResNet-50. За исключением того, что на последнем слое имеется 8631 выход, вместо 1000.

Различия заключаются лишь в количестве нейронов выходного слоя.

Для реализации системы выбрана сеть с наибольшим количеством параметров последнего слоя для обеспечения возможности наиболее точного распознавания.

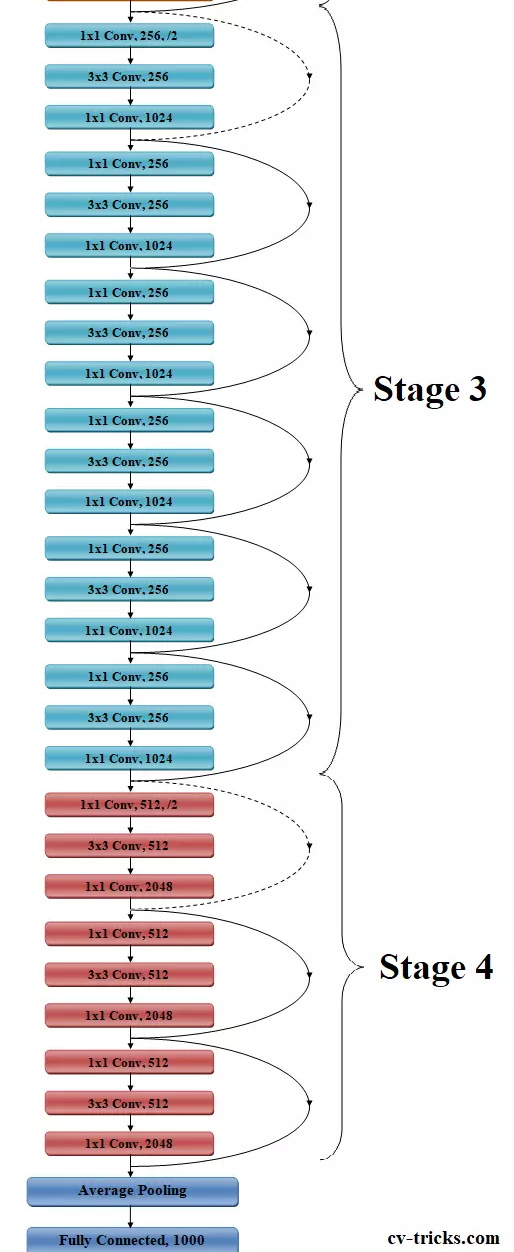
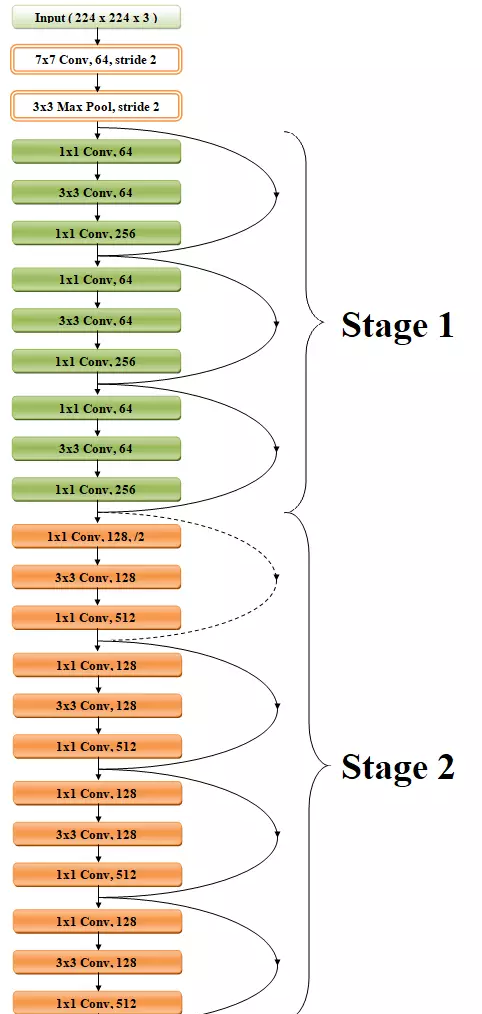


Рис. 12 Архитектура нейронной сети ResNet-50.

# **Практическая часть**

## **Выполнение процесса идентификации**

Конфигурация тестовой системы: CPU = intel i3 9100F, GPU = NVIDIA RTX 2060, RAM = 16 GB.

Транслирование наборов данных из различных форматов в облако точек.

В связи с тем, что программные системы применяют разные форматы хранения 3D изображений, вначале их следует приводить в один формат данных.

Для этих целей разработан набор скриптов. На последнем шаге приводящие данные в формат PLY. Применяется язык программирования Python версии 3.7 с библиотеками для научных вычислений (NumPy, SciKit-learn) и реконструирования 3D-изображений (PCL).

**Шаг 1.** Применение набора скриптов для преобразования данных в формате PLY из других форматов, таких как (OBJ, WRL, ABS).

**Шаг 2**. Использование скрипта augment\_здн.py к набору данных для применения аугментации и искусственного увеличения набора данных.

Для применения преобразований используются библиотеки scipy и open3d. Библиотека scipy позволяет эффективно строить матрицу поворотов. При этом методы библитеки open3d позволяют применять преобразования к облакам точек.

**Шаг 3.** Преобразование PLY облаков точек в двоичный формат NPY, содержащий представление 2D проекции облака точек на плоскость XY.

Для этого используются файлы Preprocessing.py и XYProjection.py

Для каждого изображения происходит вычитание значения средних параметров значения красной компоненты (R), зелёной компоненты (G), синей компоненты (B) значения цвета.

На вход нейронной сети передаются данные в двоичном формате, получаемые из файлов облаков точек PLY.

**Шаг 4.** Выполнение идентификации с помощью файла Recognition.py

* Считываются значения из всех бинарных данных, представленных в директориях Probe и Gallery, разделяющие в соответствующие группы.
* Данные для каждого изображения подаются на вход нейронной сети.
* Выходы нейронных сети обеих групп сравниваются друг с другом с использованием меры косинусного расстояния.
* Наиболее близкое к единице значение выдаётся за результирующее для данного файла из Gallery.

Из приведённых опытов, можно заключить, что не обученная на 3D-изображениях нейронная сеть производит идентификацию правильно лишь, в среднем, в 21% случаев.

## **Реализация процесса дообучения**

Использование нейронной сети с фреймворком глубинного обучения Caffe предполагает возможность реализации процесса дообучения (transfer learning) модели.

Учитывая, что дообучение производится для улучшения результатов предсказания в задаче идентификации (в которой нейронной сети нужно уметь правильно соотносить изображения одного человека к одному классу), производилась следующая разметка данных. Из каждого набора данных для изображений одного объекта идентификации присваивался уникальный идентификатор. Идентификаторы, при этом, не должны пересекаться между различными наборами данных.

В связи с тем, что процесс дообучения нейронной сети ResNet-50 является ресурсоёмким, были разработаны и адаптированы скрипты и конфигурационнные файлы для сервиса облачных вычислений Google Colab. Также скрипты могут использоваться для обучения сети и на персональном компьютере с поддержкой обучения лишь на CPU или GPU.

Следующие файлы обеспечивают обучение:

* resnet\_50\_solver.prototxt. Это конфигурационный файл, который описывает параметры обучения.
* generate\_datasets.py Данный файл работает в двух режимах – генерация разметки и создание базы данных в формате H5. Использует библиотеку для обработки больших наборов данных pandas.
* train\_h5\_list.txt. Содержит список файлов, которые используются в обучении.
* train.h5 - файл с данными, которыми оперирует Caffe, во время обучения.
* train\_val.prototxt – файл задающий архитектуру нейронной сети, которая обучается.

Для использования в Google Colab создан Jupyter Notebook файл caffe\_colab\_gpu.ipynb.

## Создание размеченных данных.

На данном шаге используется файл generate\_datasets.py

Каждый из имеющихся наборов данных имеет уникальную структуру. Чтобы разметить каждый из них для получения размеченных наборов требует предобработки.

Как уже описывалось ранее, каждому объекту идентификации в каждом из наборов приписывается уникальный идентификатор.

Вызывается команда: python generate\_dataset.py –type 1 –npy\_dir NPY\_DIR –save\_name SAVE\_NAME

* --type 1 указывает, что требуется создать файл разметки (пары label и путь до файла)
* --npy\_dir указывает, где находится директория с NPY файлами.
* --save\_name указывает имя, под которым сохраняется файл раметки.

После выполнения данной команды будет создан файл SAVE\_NAME, в котором будут содержаться строки, сопоставляющие идентификатор класса и путь до файла, которому данный идентификатор сопоставляется. (пример в таблице Таблица 3)

Следующей вызывается команда python generate\_dataset.py –type 2 –train\_name TRAIN\_NAME –save\_name SAVE\_NAME

* --train\_name указывает на файл, который
* –save\_name указывает имя, под которым сохраняется размеченных данных в формате H5.

После вызова данной команды создаются два файла: один содержит размеченные данные в формате H5, втором с именем train\_h5\_list.txt в котором указываются все файлы H5, которые должны быть использованы в процессе обучения.

|  |  |
| --- | --- |
| Идентификатор класса | Путь до файла |
| 1 | PLY\F0001\_AN01WH\_F3D.npy |
| 1 | PLY\F0001\_AN02WH\_F3D.npy |
| 1 | PLY\F0001\_AN03WH\_F3D.npy |
| 1 | PLY\F0001\_AN04WH\_F3D.npy |
| 2 | PLY\F0002\_DI02BL\_F3D.npy |
| 2 | PLY\F0002\_DI03BL\_F3D.npy |
| 3 | PLY\F0004\_DI01LA\_F3D.npy |

Таблица 3. Пример файла разметки train\_bu\_3dfe.txt, в котором соотносятся идентификаторы и пути до файлов NPY.

Для использования данных при тренировке нейронной сети используется формат данных Hierarchical Data Format и его современная версия HDF5. Данный формат позволяет эффективно хранить очено большие по объёму наборы данных и использовать его в любых приложения на языке программирования Python.

Отдельно стоит описать файлы конфигурации resnet\_50\_solver.prototxt . Данный файл конфигурационный файл описыват параметры, которые будут применены Caffe для оптимизации процесса обучения. Опишем некоторые параметры:

* base\_lr – это коэффициент градиентного спуска, применяемый для мсодификации весов нейронной сети в процессе обучения.
* lr\_policy – описывает политику, в соответствии с которой происходит обновление base\_lr.
* net – файл, описывающий архитектуру нейронной сети.
* gamma – параметр, определяющий величину, на которую изменяется base\_lr при срабатывании lr\_policy.
* solver\_mode – указывает, что должно использовать для обучения – CPU или GPU.
* snapshot – показывает, как часто должны создаваться копии обученной модели.

## **Тестирование**

Ниже приведём некоторые результаты тестирования обученной на некоторых данных:

|  |  |
| --- | --- |
| Количество изображений | Количество правильных ответов |
| 51 | 21 |
| 105 | 59 |

Таблица 4. Результаты тестирования нейронной сети ResNet-50 на базе данных Bosphorus.

Тестирование нейронной сети после 9000 итераций, для следующих конфигураций:

|  |  |
| --- | --- |
| Количество итераций | 9000 |
| Размер Batch | 32 |
| Обучаемый набор данных | BU-3DFE |

Таблица 5. Параметры обучения ResNet-50 для 9000 итераций.

Результат

|  |  |
| --- | --- |
| Количество изображений | Количество правильных ответов |
| 51 | 21 |
| 105 | 61 |

Таблица 6. Результаты тестирования нейронной сети ResNet-50 на базе данных Bosphorus.

# **Заключение**

В процессе выполнения дипломной работы был выработан план и реализованы компоненты системы распознавания 3D лиц людей с помощью нейронной сети. Удалось выполнить сбор данных из открытых источников, провести предварительную обработку части данных, а также адаптировать для применения в обучении нейронной сети.

Возможно продолжение работы в виде более полного обучения нейронной сети и проведении дополнительной оптимизации процесса.

# **Список литературы**

1. Николенко С., Кадурин А., Архангельская Е. – Глубокое обучение погружение в мир нейронных сетей. — СПб.: Питер, 2018. — 480 с. ISBN 978-5-496-02536-2
2. Рутковская Д., Пилиньский М., Л. Рутковский – Нейронные сети, генетические алгоритмы и нечеткие системы: Пер. с польск. И. Д. Рудинского. – М.: Горячая линия – Телеком, 2006. – 452 с.: ил. ISBN 5-93517-103-1.
3. К. В. Воронцов - Лекции по искусственным нейронным сетям.
4. Р. В. Шамин - Практическое руководство по методам машинного обучения. Москва, 2019. -
5. Parkhi O. M., Vedaldi A., Zisserman, A, Deep face recognition, 2015.
6. Kim D., Hernandez M., Choi J., Medioni G., Deep 3D Face Identification, 2017.
7. Zhou X., Seibert H., Busch C., Funk W., A 3D Face Recognition Algorithm Using Histogram-based Features, 2008.
8. Ruchay A.N., Dorofeev K.A., Kober A., Kolpakov V., Kalschikov V.V., Accuracy analysis of 3D object shape recovery using depth filtering algorithms, 2018.
9. Ruchay A.N., Dorofeev K.A., Kalschikov V.V., A novel switching bilateral filtering algorithm for depth map, 2018.
10. Diebel, J. and Thrun, S., An Application of Markov Random Fields to Range Sensing, 2005.
11. Abate A. F., Nappi M., Riccio M., Sabatino G., 2D and 3D face recognition: A survey, 2007.
12. <https://caffe.berkeleyvision.org/>
13. <https://cse.nd.edu/labs/cvrl/data-sets/und-license-agreement-3d-tec>
14. <http://kunzhou.net/zjugaps/facewarehouse/>
15. <https://github.com/ox-vgg/vgg_face2>
16. <https://www.hdfgroup.org/>
17. <https://colab.research.google.com/>

# **Приложение**

Исходные коды скриптов.

augment\_ply.py

import numpy as np

import open3d as o3d

from scipy.spatial.transform import Rotation as R

import glob

import os

from pathlib import Path

import shutil

#open3d==0.9.0

#scipy==1.4.1

save\_dir = "AUG\_PLY"

class Augmenter:

def \_\_init\_\_(self):

return

def rotate\_pcd\_by\_angle\_on\_axis(self, pcd, angle, axis = [1,0,0]):

rot\_vec = np.array(axis)

matrix = R.from\_rotvec(angle \* rot\_vec).as\_matrix()

rot\_pcd = pcd.rotate(matrix)

return rot\_pcd

def rotate\_pcd\_by\_rotation\_matrix(self, pcd, matrix):

rot\_pcd = pcd.rotate(matrix)

return rot\_pcd

def save\_ply\_by\_path(self, ply, path):

o3d.io.write\_point\_cloud(path, ply)

def load\_ply\_by\_path(self, path):

return o3d.io.read\_point\_cloud(path)

def get\_rotate\_range(self, a\_1, a\_2, steps):

return np.linspace(a\_1, a\_2, steps)

def get\_all\_ply\_files\_in\_dir\_and\_subdirs(self, in\_dir):

return Path(in\_dir).rglob('\*.ply')

def getRotationMatrixX(self, angle):

return np.array([

[1, 0, 0],

[0, np.cos(angle), -np.sin(angle)],

[0, np.sin(angle), np.cos(angle)],

])

def getRotationMatrixY(self, angle):

return np.array([

[np.cos(angle), 0, np.sin(angle)],

[0, 1, 0],

[-np.sin(angle), 0, np.cos(angle)],

])

def getRotationMatrixZ(self, angle):

return np.array([

[np.cos(angle), -np.sin(angle), 0],

[np.sin(angle), np.cos(angle), 0 ],

[0, 0, 1],

])

if \_\_name\_\_ == "\_\_main\_\_":

augm = Augmenter()

all\_plys = augm.get\_all\_ply\_files\_in\_dir\_and\_subdirs("./PLY")

#rot\_range = augm.get\_rotate\_range(np.pi/6, -np.pi/6, 30)

rot\_range = augm.get\_rotate\_range(np.pi/10, -np.pi/10, 30)

print(rot\_range)

#exit()

#new\_rot\_range = rot\_range / np.pi \* 180

new\_rot\_range = rot\_range

#rot\_range /= np.pi\*2

# x\_postfixes = [f"\_x\_{int(el)}" for el in np.round(new\_rot\_range)]

# y\_postfixes = [f"\_y\_{int(el)}" for el in np.round(new\_rot\_range)]

# z\_postfixes = [f"\_z\_{int(el)}" for el in np.round(new\_rot\_range)]

x\_postfixes = [f"\_x\_{el}" for el in new\_rot\_range]

y\_postfixes = [f"\_y\_{el}" for el in new\_rot\_range]

z\_postfixes = [f"\_z\_{el}" for el in new\_rot\_range]

x\_map = {x\_postfixes[i]: rot\_range[i] for i in range(len(rot\_range))}

y\_map = {y\_postfixes[i]: rot\_range[i] for i in range(len(rot\_range))}

z\_map = {z\_postfixes[i]: rot\_range[i] for i in range(len(rot\_range))}

print(x\_map)

#creating save\_dir for augmented ply files

flag = False

while not flag:

try:

os.makedirs(save\_dir)

flag = True

except FileExistsError:

save\_dir = save\_dir + "1"

for f\_full\_path in all\_plys:

ply = augm.load\_ply\_by\_path(str(f\_full\_path))

for postf, angle in x\_map.items():

save\_name = os.path.basename(f\_full\_path)[:-4] + postf + ".ply"

save\_path = os.path.join(save\_dir, save\_name)

#rot\_ply = augm.rotate\_pcd\_by\_angle\_on\_axis(ply, angle, [1,0,0])

matr = augm.getRotationMatrixX(angle)

rot\_ply = augm.rotate\_pcd\_by\_rotation\_matrix(ply, matr)

augm.save\_ply\_by\_path(rot\_ply, save\_path)

print(f"saved: {save\_path}")

for postf, angle in y\_map.items():

save\_name = os.path.basename(f\_full\_path)[:-4] + postf + ".ply"

save\_path = os.path.join(save\_dir, save\_name)

#rot\_ply = augm.rotate\_pcd\_by\_angle\_on\_axis(ply, angle, [0,1,0])

matr = augm.getRotationMatrixY(angle)

rot\_ply = augm.rotate\_pcd\_by\_rotation\_matrix(ply, matr)

augm.save\_ply\_by\_path(rot\_ply, save\_path)

print(f"saved: {save\_path}")

for postf, angle in z\_map.items():

save\_name = os.path.basename(f\_full\_path)[:-4] + postf + ".ply"

save\_path = os.path.join(save\_dir, save\_name)

#rot\_ply = augm.rotate\_pcd\_by\_angle\_on\_axis(ply, angle, [0,0,1])

matr = augm.getRotationMatrixZ(angle)

rot\_ply = augm.rotate\_pcd\_by\_rotation\_matrix(ply, matr)

augm.save\_ply\_by\_path(rot\_ply, save\_path)

print(f"saved: {save\_path}")

# for f\_full\_path in all\_plys:

# ply = augm.load\_ply\_by\_path(str(f\_full\_path))

# for postf, angle in x\_map.items():

# save\_name = os.path.basename(f\_full\_path)[:-4] + postf + ".ply"

# save\_path = os.path.join(save\_dir, save\_name)

# rot\_ply = augm.rotate\_pcd\_by\_angle\_on\_axis(ply, angle, [1,0,0])

# augm.save\_ply\_by\_path(rot\_ply, save\_path)

# print(f"saved: {save\_path}")

# for postf, angle in y\_map.items():

# save\_name = os.path.basename(f\_full\_path)[:-4] + postf + ".ply"

# save\_path = os.path.join(save\_dir, save\_name)

# rot\_ply = augm.rotate\_pcd\_by\_angle\_on\_axis(ply, angle, [0,1,0])

# augm.save\_ply\_by\_path(rot\_ply, save\_path)

# print(f"saved: {save\_path}")

# for postf, angle in z\_map.items():

# save\_name = os.path.basename(f\_full\_path)[:-4] + postf + ".ply"

# save\_path = os.path.join(save\_dir, save\_name)

# rot\_ply = augm.rotate\_pcd\_by\_angle\_on\_axis(ply, angle, [0,0,1])

# augm.save\_ply\_by\_path(rot\_ply, save\_path)

# print(f"saved: {save\_path}")

resnet\_50\_solver.prototxt

net: "train\_val.prototxt"

test\_iter: 1

test\_interval: 10000

base\_lr: 0.01

momentum: 0.5

weight\_decay: 0.0005

lr\_policy: "inv"

gamma: 0.1

power: 0.75

display: 10000

max\_iter: 1000000

snapshot: 1000

snapshot\_prefix: "resnet50\_"

solver\_mode: GPU