PUNE INSTITUTE OF COMPUTER TECHNOLOGY,
DHANKAWADI PUNE-43.

A Seminar Report
On
Similarity Search Algorithms on Embeddings in Face Recognition

SUBMITTED BY

NAME: Apoorv Dixit
ROLL NO: 31106
CLASS: TE-1

GUIDED BY

PROF. P. S. Vidap

COMPUTER ENGINEERING DEPARTMENT
Academic Year: 2019-20

PUNE INSTITUTE OF COMPUTER TECHNOLOGY,
DHANKAWADI PUNE-43.

CERTIFICATE

This is to certify that Mr. ***Apoorv Dixit*** , *R*oll No**. 31106** a student of T.E. (Computer Engineering Department) Batch 2019-2020, has satisfactorily completed a seminar report on "**Similarity Search Algorithms on Embeddings in Face Recognition**" under the guidance of Prof. P. S.Vidap towards the partial fulfillment of the third year Computer Engineering Semester II of Pune University.

Prof. P. S. Vidap	Prof. M.S.Takalikar
Internal Guide	**Head of Department,**
	Computer Engineering

Date:

Place:

ACKNOWLEDGEMENT

I would like to extend my sincerest gratitude to my guide Prof. P.S. Vidap for providing me with the necessary resources as well as her invaluable insights. This seminar would not have been possible without her constant support. She has been instrumental in making this seminar a grand success.

SIMILARITY SEARCH ALGORITHMS ON EMBEDDINGS IN FACE RECOGNITION

Abstract:

Face Recognition is a challenging task in computer vision in which researchers from all over the world have made great strides owing to the development of more robust and accurate deep learning models. These frameworks have come a long way from hand engineered systems to the end-to-end mapped models that are state of the art today. However, implementing faster similarity search of face features is an increasingly important problem. There is a dire need to address this issue in order to make the face recognition pipelines scalable and more suited for real time applications. This seminar aims to benchmark different similarity search algorithms on face embeddings generated from a standard face dataset. The benchmarking will help other users choose a nearest neighbour search method for their face recognition models.

***Keywords**: Face recognition, computer vision, deep learning model, nearest neighbor search, embeddings, similarity search, benchmark.*

Contents

List of Figures

List of Tables

1 INTRODUCTION

Face Recognition is the task of identifying a person from either a digital image or a frame from a video. This task is typically accomplished by obtaining facial features of a face from a given image and subsequently comparing it with other faces stored in a database. It finds its application in a wide variety of fields like robotics, access controls in security systems, commercial identification, marketing tool, human computer interaction, video surveillance, biometrics and many more. Face Recognition's contactless and non-invasive process is what sets it apart from other biometric technologies like iris recognition and fingerprint recognition.

Face Recognition is a task that can be performed trivially by humans, but can prove to be a daunting challenge for machines. It has remained so for computer vision problems until recently. There have been major strides in the development of more accurate and robust systems in face recognition owing to the advent of artificial intelligence in the past two decades. We have come a long way from the hand engineered models used in the beginning of the 21st century to the deep learning pipelines that are pretty much the state of the art today. It is only in recent times that these state of the art models have matched, if not surpassed, the human accuracy. The deep learning systems employed take into consideration various variable conditions like light textures, pose and orientation of the person's face, age of the person, the varying facial hair and much more.

Face Recognition Pipelines are primarily broken down into the following major subtasks – Face Detection, Face Alignment, Feature Extraction and finally, the recognition task. The Modern deep learning pipelines for facial recognition systems deploy deep learning models for the first three subtasks, and use an Approximate Nearest Neighbour Search Algorithm (ANN) for the recognition. This report will be focusing on face embeddings and various ANN, which will cater to Feature Extraction and Recognition respectively.

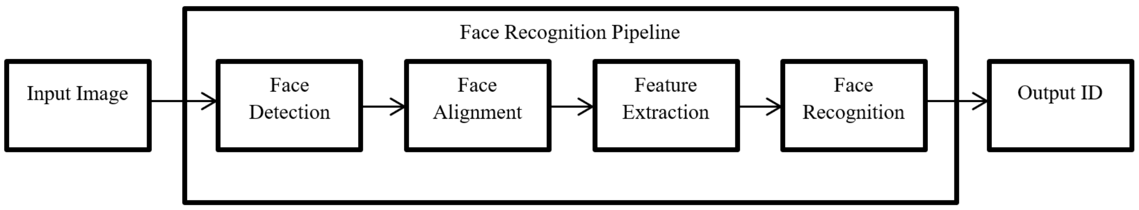


Figure 1: Face Recognition Pipeline Architecture

1.1 Motivation

While accuracy is not that big of a problem in Face Recognition Systems anymore, making them more suited for real time tasks is. The deep learning models that we see today have been trained on millions of faces. The memory footprint of these models is way too heavy to be used for real time systems. Moreover deep learning pipelines are composed of more than one deep learning model, to perform the different subtasks like Face Detection, Face Alignment and Feature Extraction. This makes them too slow for achieving the overall objective.

This issue can be tackled in two ways. The first way is to design a more light weight deep learning model for feature extraction. The second way is to select a fast ANN for the final recognition subtask. This final subtask involves comparing the extracted features of the face obtained with that of faces stored in a database. This subtask returns the ID of the person the face resembles the closest to, thereby recognizing the person. The database typically stores the extracted features of faces rather than the images of the faces themselves. The comparison is done by an ANN.

1.2 Literature Survey

1.2.1 ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms

This paper talks about ANN-Benchmarks [1] , a tool for evaluating the performance of in-memory approximate nearest neighbor algorithms (ANNs). It provides a standard interface for measuring the performance and quality achieved by ANNs on different standard data sets. It supports several different ways of integrating kNN algorithms, and its configuration system automatically tests a range of parameter settings for each algorithm. Algorithms are compared with respect to many different quality measures, and adding more is easy and fast. This tool aims to provide a constantly updated overview of the current state of the art of kNN algorithms.

1.2.2 FaceNet: A Unified Embedding for Face Recognition and Clustering

This paper present a system called FaceNet [2] , that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. Once this space has been produced, tasks such as face recognition, verification and clustering can be easily implemented using standard techniques with FaceNet embeddings [2] as feature vectors. FaceNet [2] uses a deep convolutional network trained to directly optimize the embedding itself. To train the deep learning model, the authors of the paper use triplets of roughly aligned matching / non-matching face patches generated using a novel online triplet mining method. This ensures a much greater representational efficiency. When this paper was released, it achieved state of the art face recognition performance on the standard face datasets using only 128-bytes per face. Most of the modern proprietary feature extraction models find their base in FaceNet [2] . It laid the foundation for the modern face recognition pipelines.

1.2.3 Disentangled representation learning GAN for pose invariant face recognition

Pose Discrepancy is a major issue in face recognition. To combat the same, traditional approaches either perform face frontalization of the face or learn a pose-invariant representation. This paper proposes Disentangled Representation learning-Generative Adversarial Network (DR-GAN) [3] , which performs both the approaches mentioned before in order to leverage the benefits of both. DR-GAN [3] sports three distinct novelties under its belt. First, the encoder-decoder structure of the generator allows it to learn a generative and discriminative representation. Second, this representation is pose invariant, through the pose code provided to the decoder and pose estimation in the discriminator. Third, DR-GAN [3] can take one or multiple images as the input, and generates one common representation. It has demonstrated its superiority over other face recognition models on face datasets with high pose variance. It might very well turn out to be a precursor to future state of the art face recognition feature extractors.

1.2.4 **VGGFace2: A dataset for recognising faces across pose and age**

This paper talks about VGGFace2 [4], a face dataset contains 3.31 million images of 9131 subjects, with an average of 362.6 images for each subject. Images are downloaded from Google Image Search and have large variations in pose, age, illumination, ethnicity and even image noise. DR-GAN [3] was trained on this dataset.

1.2.5 **Visualizing Data using t-SNE**

This paper describes a technique called “t-SNE” [5] that visualizes high-dimensional data by giving each data-point a location in a 2D or 3D map. The technique is a variation of Stochastic Neighbor Embedding but is much easier to optimize, and produces significantly better visualizations by reducing the tendency to crowd points together in the center of the map. t-SNE [5] is better than other existing techniques at creating a single map that reveals structure at many different scales. This is particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds, such as images of objects from multiple classes seen from multiple viewpoints. In this seminar we demonstrate clustering capabilities of the face embeddings used with t-SNE technique [5] .

1.3 Challenges

1.3.1 Proprietary Feature Extracting Models

Most of the proprietary deep learning models for embedding generation which are state of the art are proprietary. Their saved models are not readily available.

1.3.2 Older ANNs are implemented in Python 2

Many of the older ANNs have been implemented in Python 2. Due to which they cannot be benchmarked against Python 3 ANNs. Python 2.x is generally faster than Python 3.x, which gives the former an unfair advantage.

1.3.3 Need better GPUs for generating a larger embedding dataset

A larger embedding dataset will enable us to capture the performance of ANNs more effectively, thereby making the results more accurate.

2 FACE EMBEDDINGS

For Feature extraction, the deep learning model trains itself to extract features of a given dimension from millions of images. The input for these neural network models are face images which are cropped and aligned for better performance. The output of these models is a vector of a given dimension which encompasses the features of the face. These are called face embeddings. Thus, the embedding of the person in question is generated, which is then compared against the embeddings of the faces stored in the database. Needless to say, embeddings stored in the database are generated from the same deep learning model.

FaceNet [2] is the deep convolutional network trained to directly optimize the embedding itself. FaceNet [2] accelerated the research in Face Recognition and most of the State of the Art Face Embedding generators find their base in FaceNet [2] . These modern derivatives of FaceNet [2] are State of the Art albeit proprietary models.

DR-GAN [3] , on the other hand, approaches pose invariant face recognition with a novel representation. This representation is generative and discriminative in nature which leverages the face frontalization of DR-GAN [3] . This pose invariant representation shows promising results in face recognition research.

In this report, we have generated face embeddings for a dataset, created a database out of the same and then benchmarked various ANN on the database synthesized. We have generated embeddings using FaceNet [2] and DR-GAN [3] . The dataset used for this seminar is VGGFace2 [4] . VGGFace2 [4] is a large-scale face recognition dataset. This dataset has large variations in pose, age, illumination, ethnicity and profession. DR-GAN [3] was trained on VGGFace2 [4].

It is interesting to note when t-SNE technique [5] is applied on the FaceNet Embeddings [2] , the embeddings of the same person group together, since they are similar. Thus FaceNet embeddings [2] portray an impressive cluster analysis. DR-GAN [3] on the other hand does not show significant clustering under t-SNE [5] .

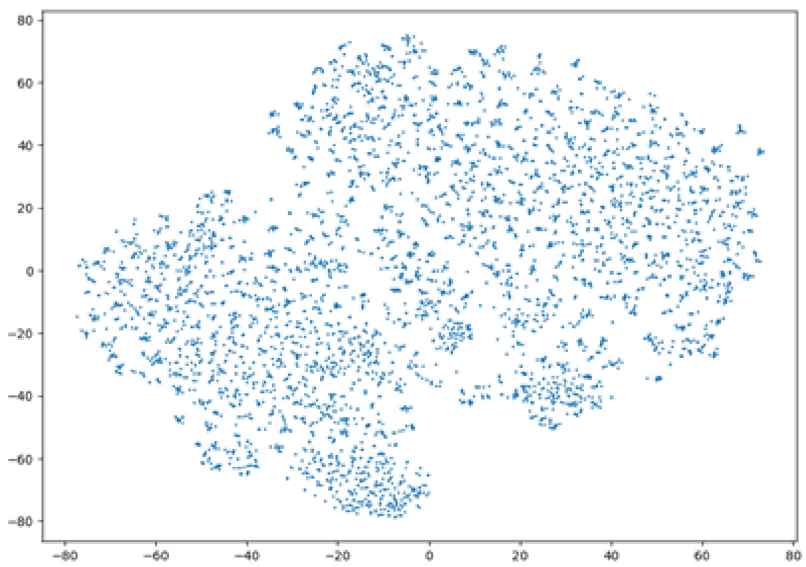


Figure 2: TSNE Analysis on FaceNet Embeddings

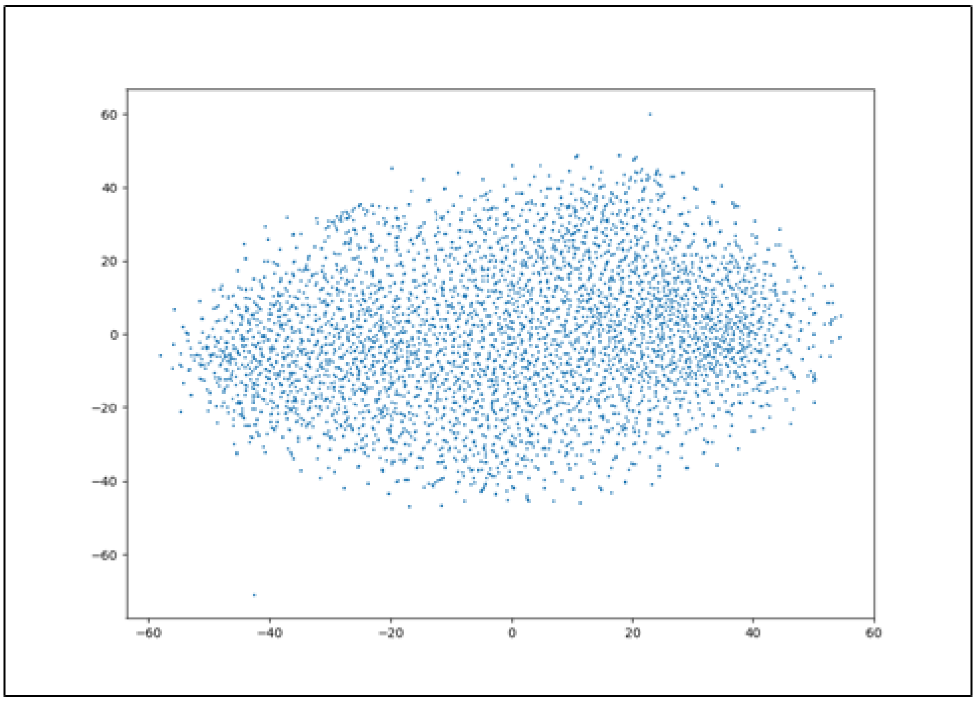


Figure 3: TSNE Analysis on DR-GAN Embeddings

3 APPROXIMATE NEAREST NEIGHBOURS

Nearest neighbor search (NNS) is the optimization problem of finding the point in a given set that is closest (or most similar) to a given point. Closeness is typically expressed in terms of a dissimilarity function: the less similar the objects, the larger the function values. In some applications it may be acceptable to retrieve a "good guess" of the nearest neighbor. In those cases, we can use an algorithm which doesn't guarantee to return the actual nearest neighbor in every case, in return for improved speed or memory savings. These algorithms are called Approximate Nearest Neighbour Search (ANN).

The recognition subtask involves comparing the embedding of the face obtained with that of faces stored in a database. This subtask returns the ID of the person the face resembles the closest to, thereby recognizing the person. The database typically stores the face embeddings rather than the images of the faces themselves. The comparison is done by an ANN.

In this report, I have benchmarked various ANN algorithms on the basis of queries executed versus Time. This benchmarking has been done on the FaceNet embeddings [2] . The ANN algorithms either have a python wrapper, a python package or were cloned from Github. Moreover, some ANN algorithms support batch processing.

4 PROPOSED ARCHITECTURE

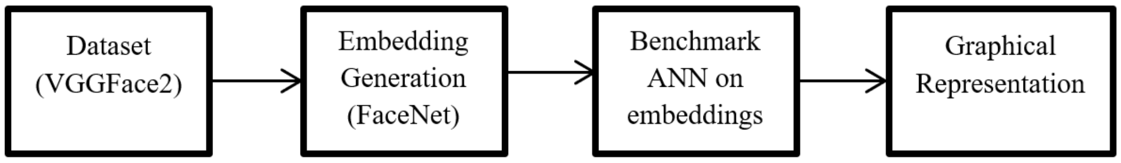


Figure 4: Report Implementation Overview

For the first part of this report two sets of embeddings have been generated for 5000 test images of VGGFace2 [4]. These 5000 images include 10 images each of 500 people. The two sets of embeddings are the FaceNet embeddings [2] and DR-GAN's pose invariant embeddings. FaceNet embeddings [2] have been generated using a Python library of the same. On the other hand, DR-GAN's embeddings have been generated using the demo test code on their official site.

After the demo of the embedding, we have benchmarked 20 ANN algorithms on the basis of queries executed versus Time. This benchmarking has been done on the FaceNet embeddings [2] generated in the previous part. The ANN algorithms either have a python wrapper, a python package or were cloned from Github. Moreover, some ANN algorithms support batch processing. Some of the older ANNs which were implemented in Python 2 were converted to equivalent Python 3 code using Python 2to3 refactoring tool. This benchmarking has been done on Google Colab.

5 IMPLEMENTATION

5.1 Platforms and Technologies

Google Colaboratory - It is a free Jupyter notebook environment that runs in the cloud and stores the notebooks on Google Drive. The implementation is done on Google Colaboratory (Colab) under Python 3 environment.

5.2 Python Libraries

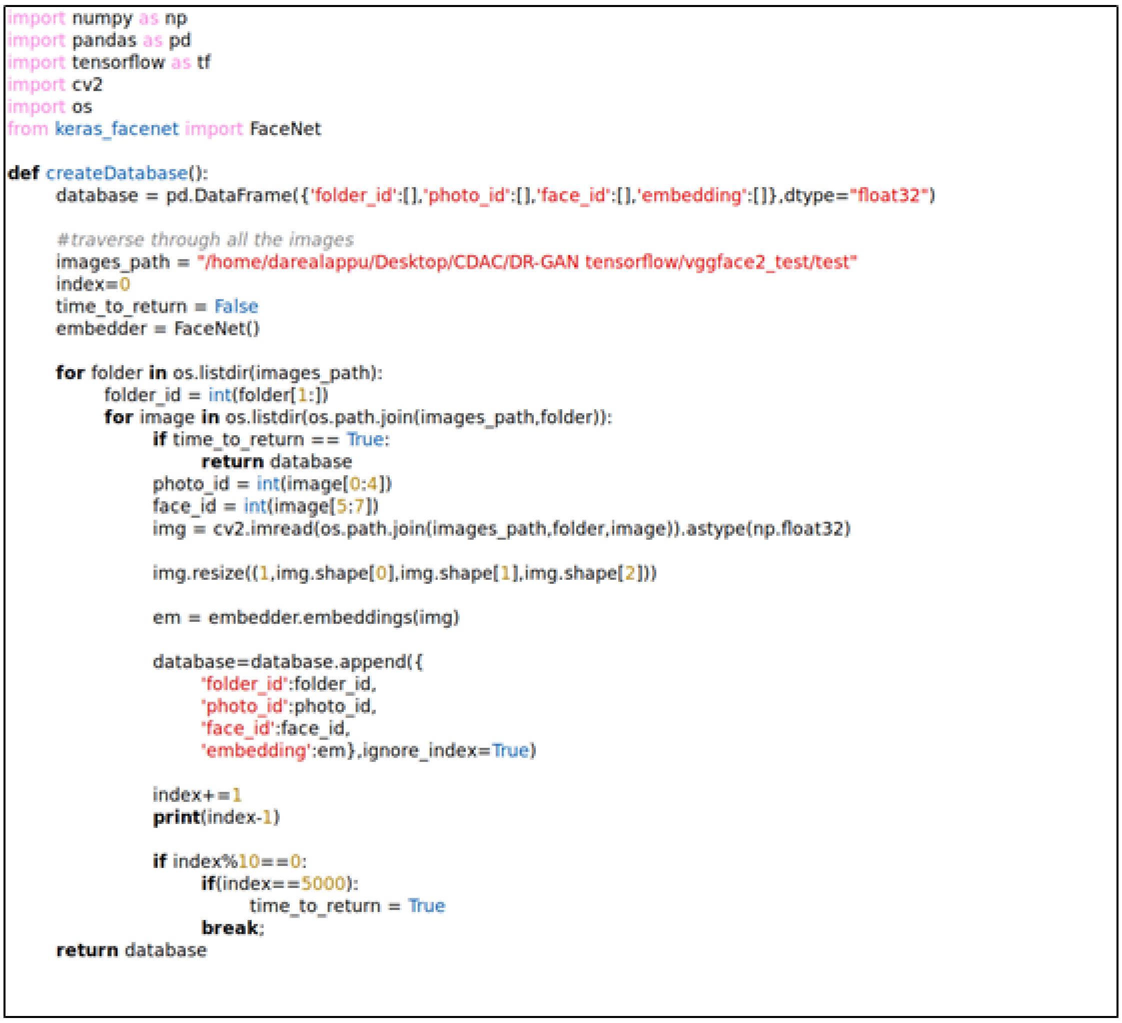
- **Core Libraries**
    - Time - To benchmark ANN libraries.
    - numpy - To store N-dimensional array objects.

  - pandas - To store face embeddings in dataframe object.
  - sklearn
    1) It has 3 NNS Algorithms - KDTree, Ball Tree and Brute Method.
    2) It has t-SNE Algorithm [5] .
  - 2to3 - It is a refactoring tool to convert Python 2 to Python 3 code.
  - keras_facenet - To generate FaceNet face embeddings from a face image.
  - Matplotlib - To visualize performance of ANN libraries.

- **ANN Python Libraries**
  - annoy [6]
  - nearpy [7]
  - ngt [8, 9]
  - pyflann [10]
  - mrpt [11]
  - pynndescent [12]
  - faiss [13]
  - rpforest [14]
  - hnswlib [15]
  - nmslib - hnsw [15], sw-graph, napp [16]
  - n2 [15]
  - panns [17]
  - kgraph [12]
  - DolphinnPy [18]
  - FALCONN [19]
  - nanopq [20, 21]
  - sklearn - KDTree [22], BallTree [23], Brute

- **Github**
  - DR-GAN - To generate DR-GAN embeddings of face images. [3]

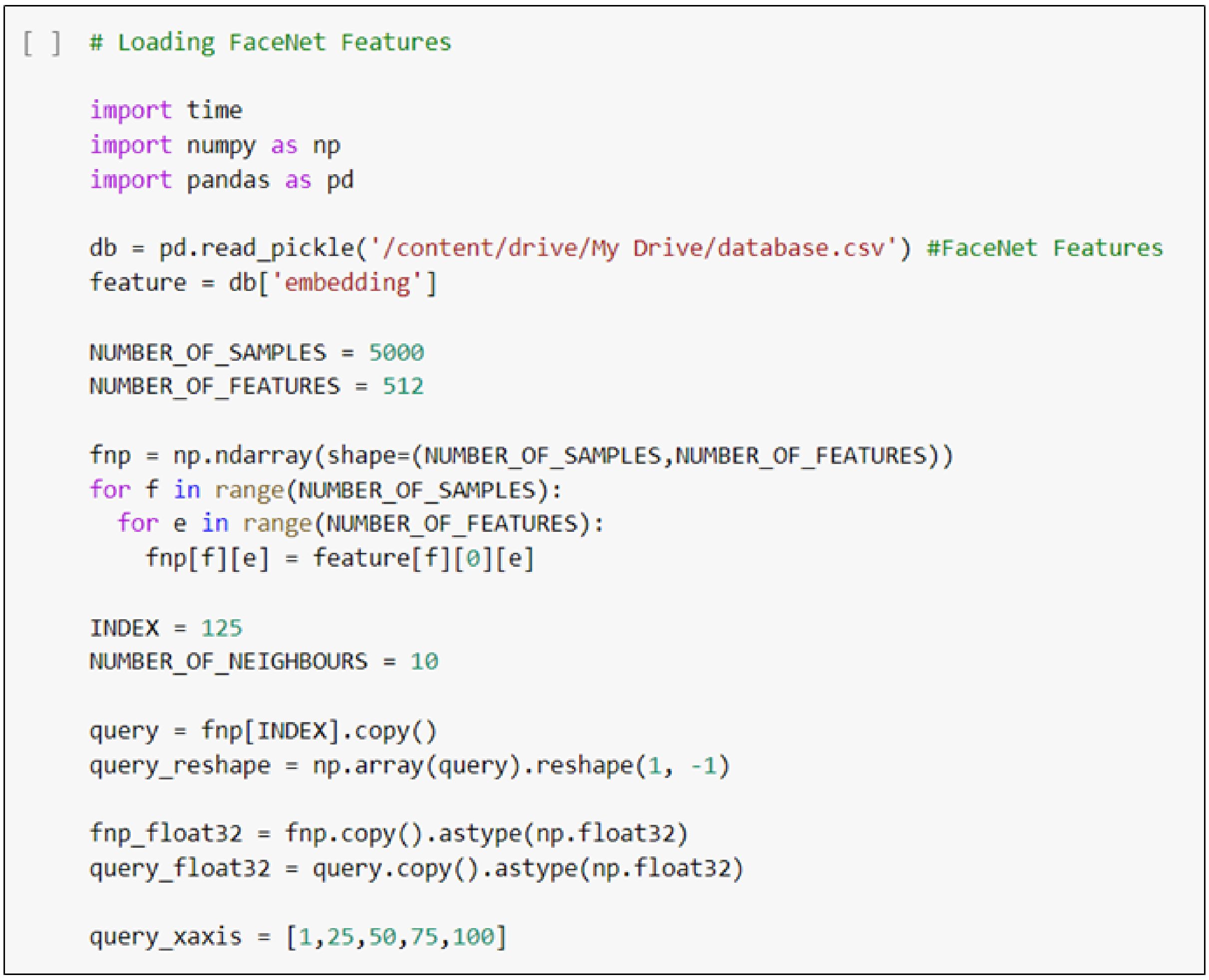
5.3 Source Code



```python
import numpy as np
import pandas as pd
import tensorflow as tf
import cv2
import os
from keras_facenet import FaceNet

def createDatabase():
    database = pd.DataFrame({'folder_id':[],'photo_id':[],'face_id':[],'embedding':[]},dtype="float32")

    #traverse through all the images
    images_path = "/home/darealappu/Desktop/CDAC/DR-GAN tensorflow/vggface2_test/test"
    index=0
    time_to_return = False
    embedder = FaceNet()

    for folder in os.listdir(images_path):
        folder_id = int(folder[1:])
        for image in os.listdir(os.path.join(images_path,folder)):
            if time_to_return == True:
                return database
            photo_id = int(image[0:4])
            face_id = int(image[5:7])
            img = cv2.imread(os.path.join(images_path,folder,image)).astype(np.float32)

            img.resize((1,img.shape[0],img.shape[1],img.shape[2]))

            em = embedder.embeddings(img)

            database=database.append({
                'folder_id':folder_id,
                'photo_id':photo_id,
                'face_id':face_id,
                'embedding':em},ignore_index=True)

            index+=1
            print(index-1)

            if index%10==0:
                if(index==5000):
                    time_to_return = True
                break;
    return database
```

Figure 5: Source Code for FaceNet Embedding Generation

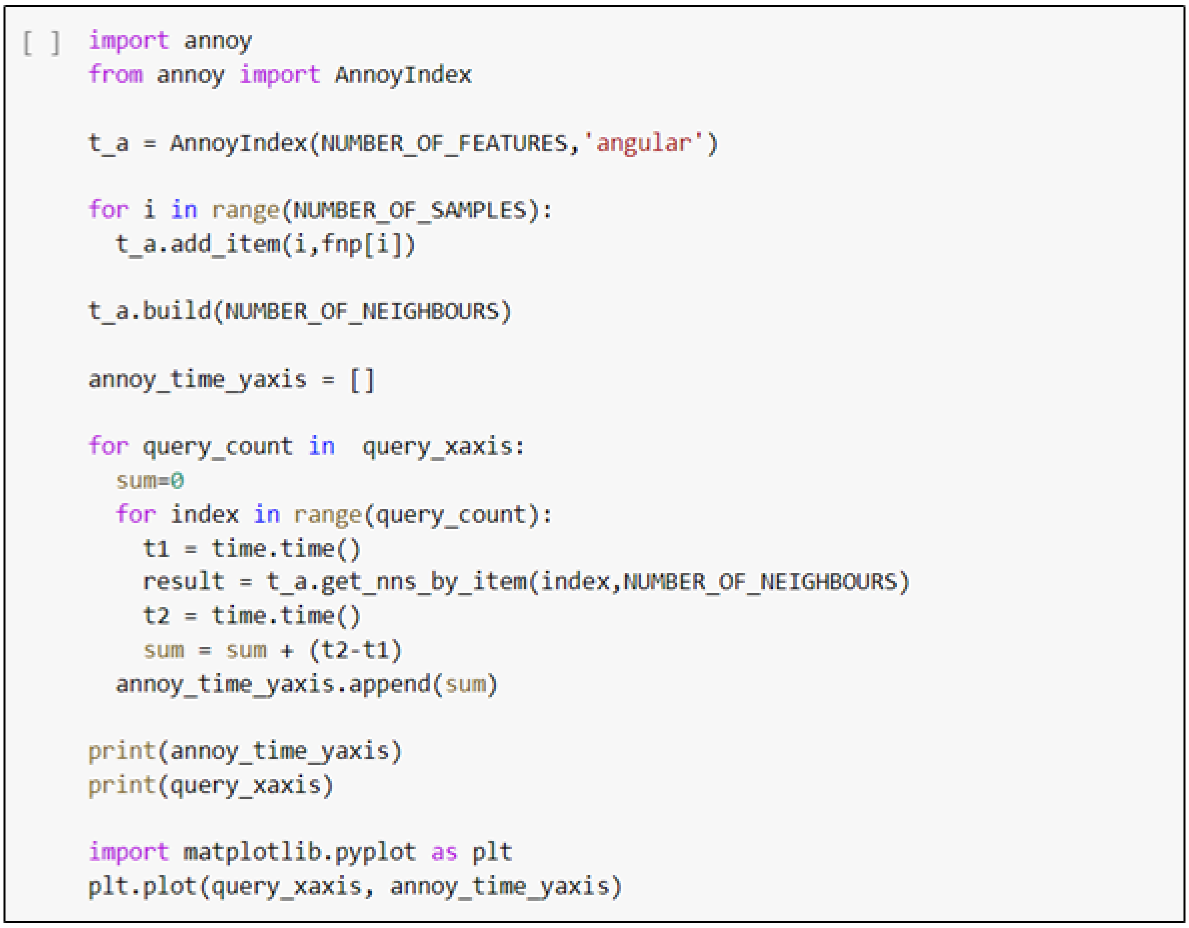
This Python 3 script was used to create a database which contains FaceNet Embeddings of 5,000 faces. 10 images each were selected for 500 persons. This database was subsequently saved as a CSV file as “database.csv” . A similar Python 3 script was used to create a database of DR-GAN Embeddings [3] as well



```
[ ]  # Loading FaceNet Features

     import time
     import numpy as np
     import pandas as pd

     db = pd.read_pickle('/content/drive/My Drive/database.csv') #FaceNet Features
     feature = db['embedding']

     NUMBER_OF_SAMPLES = 5000
     NUMBER_OF_FEATURES = 512

     fnp = np.ndarray(shape=(NUMBER_OF_SAMPLES,NUMBER_OF_FEATURES))
     for f in range(NUMBER_OF_SAMPLES):
       for e in range(NUMBER_OF_FEATURES):
         fnp[f][e] = feature[f][0][e]

     INDEX = 125
     NUMBER_OF_NEIGHBOURS = 10

     query = fnp[INDEX].copy()
     query_reshape = np.array(query).reshape(1, -1)

     fnp_float32 = fnp.copy().astype(np.float32)
     query_float32 = query.copy().astype(np.float32)

     query_xaxis = [1,25,50,75,100]
```

Figure 6: Source Code for Retrieving FaceNet Embeddings

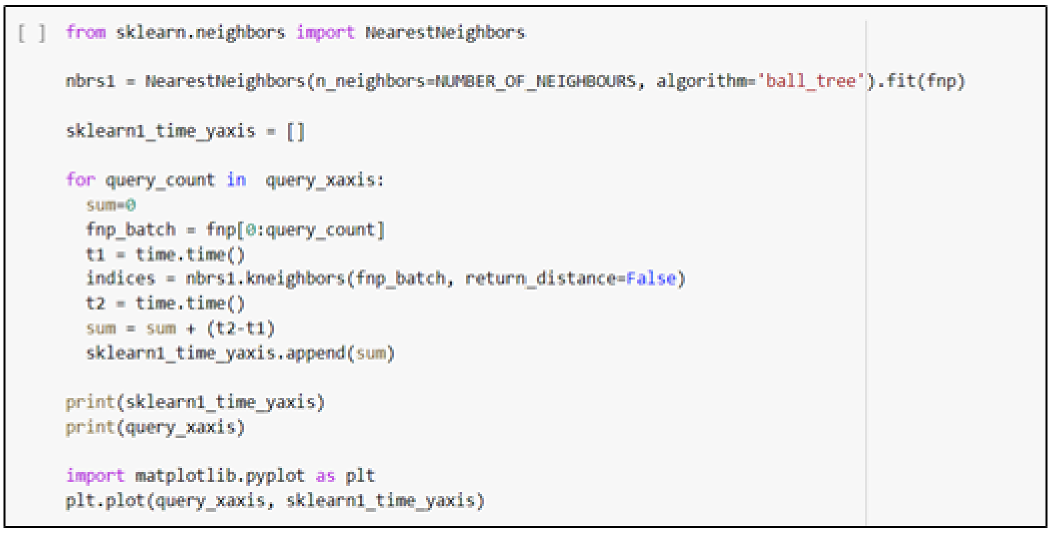
This Python 3 Cell in Google Colaboratory Notebook was used to load the FaceNet Embeddings [2] from “database.csv” and initialize various other parameters. A similar cell was used for DR-GAN Embeddings [3] as well.



```
[ ] import annoy
    from annoy import AnnoyIndex

    t_a = AnnoyIndex(NUMBER_OF_FEATURES,'angular')

    for i in range(NUMBER_OF_SAMPLES):
      t_a.add_item(i,fnp[i])

    t_a.build(NUMBER_OF_NEIGHBOURS)

    annoy_time_yaxis = []

    for query_count in  query_xaxis:
      sum=0
      for index in range(query_count):
        t1 = time.time()
        result = t_a.get_nns_by_item(index,NUMBER_OF_NEIGHBOURS)
        t2 = time.time()
        sum = sum + (t2-t1)
      annoy_time_yaxis.append(sum)

    print(annoy_time_yaxis)
    print(query_xaxis)

    import matplotlib.pyplot as plt
    plt.plot(query_xaxis, annoy_time_yaxis)
```

Figure 7: Benchmarking ANN with batch processing

This Python 3 Cell in Google Colaboratory Notebook is an example of the benchmarking of an ANN which does not facilitate batch processing.



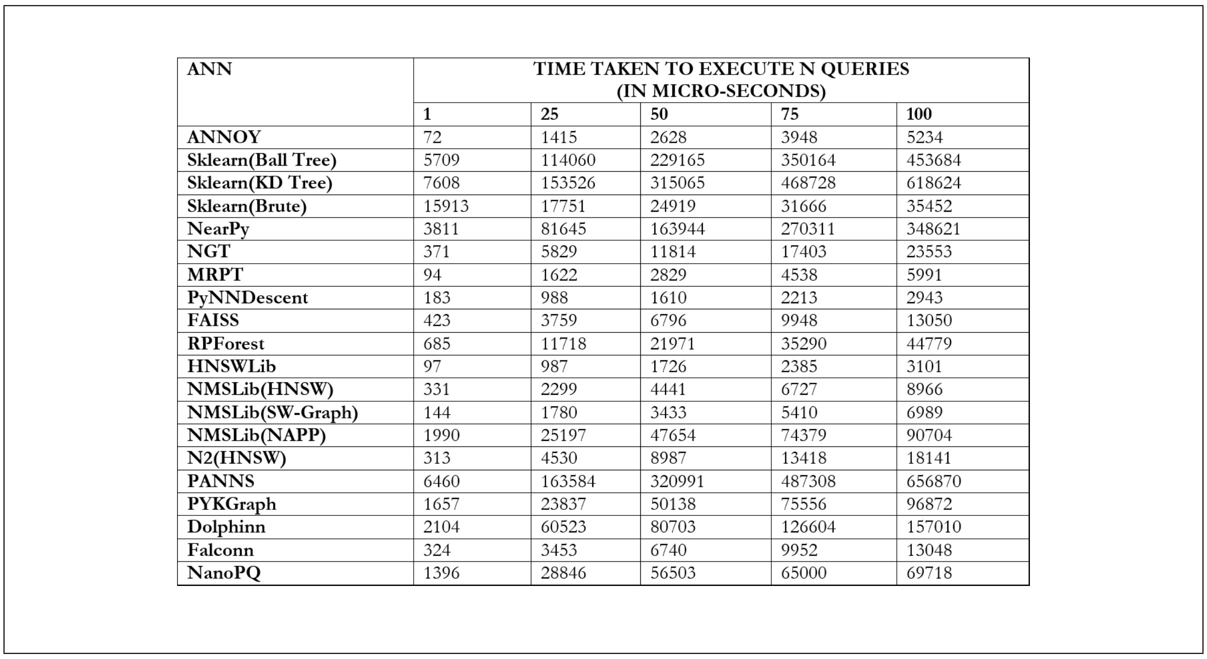
```
[ ] from sklearn.neighbors import NearestNeighbors

    nbrs1 = NearestNeighbors(n_neighbors=NUMBER_OF_NEIGHBOURS, algorithm='ball_tree').fit(fnp)

    sklearn1_time_yaxis = []

    for query_count in  query_xaxis:
      sum=0
      fnp_batch = fnp[0:query_count]
      t1 = time.time()
      indices = nbrs1.kneighbors(fnp_batch, return_distance=False)
      t2 = time.time()
      sum = sum + (t2-t1)
      sklearn1_time_yaxis.append(sum)

    print(sklearn1_time_yaxis)
    print(query_xaxis)

    import matplotlib.pyplot as plt
    plt.plot(query_xaxis, sklearn1_time_yaxis)
```

Figure 8: Benchmarking ANN without batch processing

This Python 3 Cell in Google Colaboratory Notebook is an example of the benchmarking of an ANN which facilitates batch processing.

6 RESULT AND ANALYSIS

20 ANN libraries in Python 3 have been benchmarked on FaceNet embeddings [2] on 5000 images of VGGFace2 dataset [4] successfully and the results are tabulated below:



ANN	TIME TAKEN TO EXECUTE N QUERIES (IN MICRO-SECONDS)				
	1	25	50	75	100
ANNOY	72	1415	2628	3948	5234
Sklearn(Ball Tree)	5709	114060	229165	350164	453684
Sklearn(KD Tree)	7608	153526	315065	468728	618624
Sklearn(Brute)	15913	17751	24919	31666	35452
NearPy	3811	81645	163944	270311	348621
NGT	371	5829	11814	17403	23553
MRPT	94	1622	2829	4538	5991
PyNNDescent	183	988	1610	2213	2943
FAISS	423	3759	6796	9948	13050
RPForest	685	11718	21971	35290	44779
HNSWLib	97	987	1726	2385	3101
NMSLib(HNSW)	331	2299	4441	6727	8966
NMSLib(SW-Graph)	144	1780	3433	5410	6989
NMSLib(NAPP)	1990	25197	47654	74379	90704
N2(HNSW)	313	4530	8987	13418	18141
PANNS	6460	163584	320991	487308	656870
PYKGraph	1657	23837	50138	75556	96872
Dolphinn	2104	60523	80703	126604	157010
Falconn	324	3453	6740	9952	13048
NanoPQ	1396	28846	56503	65000	69718

Table 1: ANN Benchmarks

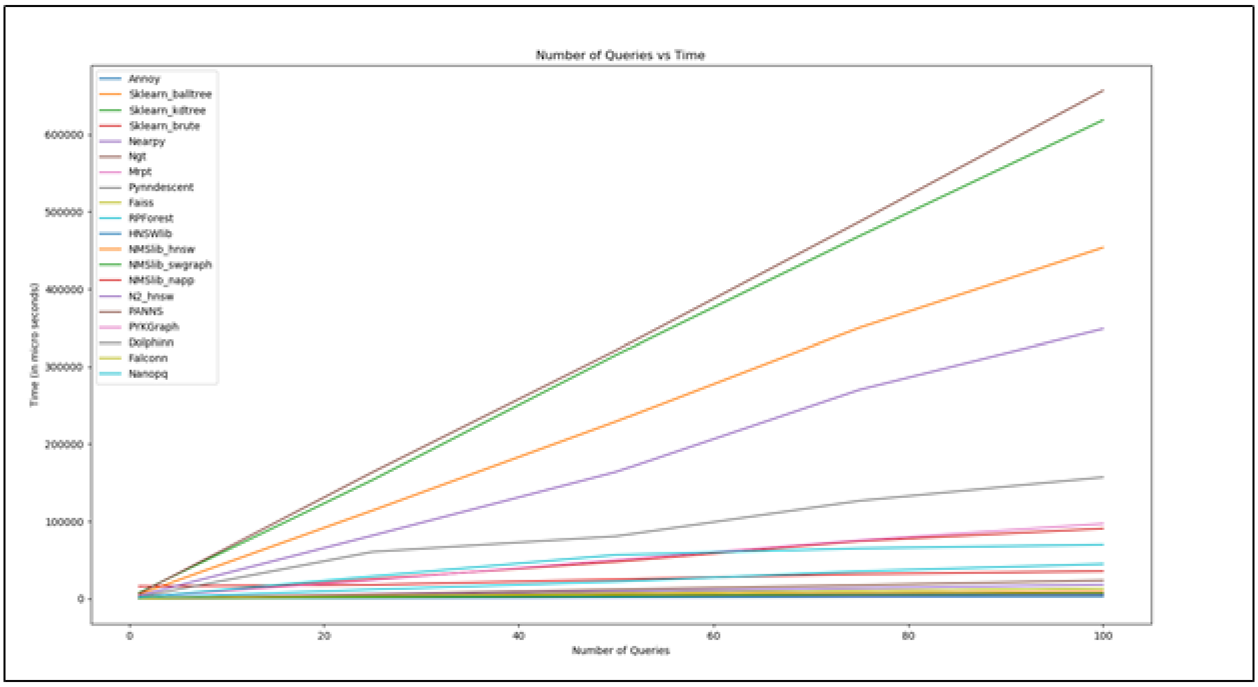


Figure 9: Queries vs Time graph

7 CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

The results generated establish that while ANNOY ANN is the fastest ANN for a single query, PyNNDescent delivers the fastest results for multiple queries, given that it can facilitate batch processing as well. From the results, it is noted that for single queries, ANNOY [6] (72 ms) gives us 23% boost over the immediate runner-up MRPT [11] (94 ms), and is 221 times faster than the slowest NNS, Brute force method of SKLearn. For Multiple Queries, Pynndescent [12] delivers the fastest results (2943 ms for 100 queries). It is followed closely by HNSWLib [15] benchmarked at 3101 ms for 100 queries.

Face Recognition is an active area of research and making it more suited for real time application still proves to be a challenging task. This report provides an insight on generation of face embeddings and analysis of performance of various Approximate Nearest Neighbour Libraries. The results generated will help developers to choose a suitable ANN for their Face Recognition Pipeline.

7.2 Future Scope

Even though the results generated are satisfactory, to make this project more comprehensive, the following points can be added.

- Include more Deep Learning Models for Embedding Generators.
- Include ANN algorithms written in Python2 by converting their code to Python3 and further optimise their performance with the new features of Python3.
- A larger Embedding Dataset can be generated to capture the performance of ANN more accurately.

References

[1] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. “ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms”. In: *International Conference on Similarity Search and Applications*. Springer. 2017, pp. 34–49.

[2] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[3] Luan Tran, Xi Yin, and Xiaoming Liu. “Disentangled representation learning gan for pose-invariant face recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1415–1424.

[4] Qiong Cao et al. “Vggface2: A dataset for recognising faces across pose and age”. In: *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE. 2018, pp. 67–74.

[5] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[6] *ANNOY library*. https://github.com/spotify/annoy.

[7] Keon Myung Lee. “Locality-Sensitive Hashing Techniques for Nearest Neighbor Search”. In: *International Journal of Fuzzy Logic and Intelligent Systems* 12 (Dec. 2012), pp. 300–307.

[8] Masajiro Iwasaki. “Pruned Bi-directed K-nearest Neighbor Graph for Proximity Search”. In: *SISAP*. 2016.

[9] Masajiro Iwasaki and Daisuke Miyazaki. “Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data”. In: *arXiv preprint arXiv:1810.07355* (2018).

[10] Marius Muja and David Lowe. “Flann-fast library for approximate nearest neighbors user manual”. In: ().

[11] Ville Hyvönen et al. “Fast k-nn search”. In: *arXiv preprint arXiv:1509.06957* (2015).

[12] Wei Dong, Charikar Moses, and Kai Li. “Efficient k-nearest neighbor graph construction for generic similarity measures”. In: *Proceedings of the 20th international conference on World wide web*. 2011, pp. 577–586.

[13] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billion-scale similarity search with GPUs”. In: *arXiv preprint arXiv:1702.08734* (2017).

[14] Donghui Yan et al. “K-nearest Neighbor Search by Random Projection Forests”. In: Dec. 2018.

[15] Yury A Malkov and Dmitry A Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* (2018).

[16] Eric Sadit Tellez, Edgar Chávez, and Gonzalo Navarro. “Succinct nearest neighbor search”. In: *Proceedings of the Fourth International Conference on SImilarity Search and APplications*. 2011, pp. 33–40.

[17] *Panns on Github*. https://github.com/ryanrhymes/panns.

[18] Georgia Avarikioti et al. “Practical linear-space Approximate Near Neighbors in high dimension”. In: (Dec. 2016).

[19] Alexandr Andoni et al. “Practical and optimal LSH for angular distance”. In: *Advances in neural information processing systems*. 2015, pp. 1225–1233.

[20] H. Jégou, M. Douze, and C. Schmid. “Product Quantization for Nearest Neighbor Search”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (2011), pp. 117–128.

[21] T. Ge et al. “Optimized Product Quantization”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.4 (2014), pp. 744–755.

[22] Rina Panigrahy. “An improved algorithm finding nearest neighbor using kd-trees”. In: *Latin American Symposium on Theoretical Informatics*. Springer. 2008, pp. 387–398.

[23] Mohamad Dolatshah, Ali Hadian, and Behrouz Minaei-Bidgoli. “Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces”. In: *arXiv preprint arXiv:1511.00628* (2015).

Appendix – D

Log Book

Roll No. **:- 31106**

Name of the Student **:- Apoorv Dixit**

Name of the Guide **:- Prof. P.S.Vidap**

Seminar Title **:- Similarity Search Algorithms on Embeddings in Face Recognition**

Sr. No.	**Date**	**Details of Discussion/ Remarks**	**Signature of guide / Seminar Incharge**
1.			
2.			
3.			
4.			
5.			
6.			
7.			
8.			
9.			
10.			

Student Signature **Guide Signature**