Karnataka Law Society’s

KLS GOGTE INSTITUTE OF TECHNOLOGY-590008

UDYAMBAG, BELAGAVI-59008



**An Autonomous Institute under Vishweshwaraya  Technological University, Belagavi**

Report

**Automatic Human Face Recognition and Segmentation**

**REPORT 2023-24**

**Semester:5**

**Submitted By**

Aum Pauskar - 2GI21CS037

**Under the guidance of**

**Prof. Anil Gavade**

**of Electronics and communications**

**Academic Year 2023 - 24(odd semester)**

**Institute Vision**

KLS Gogte Institute of Technology shall stand out as an institution of excellence in technical education and in training individuals for outstanding caliber, character coupled with creativity and entrepreneurial skills**.**

**Mission**

To train the students to become quality engineers with high standards of professionalism and ethics who have positive attitude, a perfect blend of           techno-managerial skills and problem-solving ability with an analytical and innovative mindset.

**Department Vision:**

To be a centre of excellence in Computer Science and Engineering education and research, producing globally competent professionals who can cater to the needs of industry and society.

**Mission:**

* To impart quality education in Computer Science and Engineering through effective teaching-learning process and state-of-the-art infrastructure.
* To provide a conducive environment for students and faculty to pursue research and innovation in emerging areas of Computer Science and Engineering.
* To inculcate professional ethics, social values, leadership skills and entrepreneurial spirit among the students and faculty.
* To foster industry-institute interaction and collaboration with reputed institutions and organizations for enhancing the knowledge and skills of the students and faculty.

# Karnatak Law Society’s

## GOGTE INSTITUTE OF TECHNOLOGY

# Udyambag Belagavi -590008 Karnataka, India.

# 

# 

# Department of Computer Science and Engineering

# https://lh7-us.googleusercontent.com/8MIWHIEVPHoBfqiPu39OgddNgiaB5Yr8exB72SX4AbAwIFKTyBHHQyaixf_LM3sZwiZOzJPgjPdAOr8te_oO4EcUEmErUcA8pL_6WAevqqG_74dw81taNqoOUOIDBxdI-o9rHZl3rLP2y5d6Cj6CAwg

# Certificate

# This is to certify that the Course Project work titled “Automatic Human Face Recognition and Segmentation” carried out by  Aum P ,bearing USNs: 2GI21CS037for Big Data Management (21CS542) course is submitted in partial fulfillment of the requirements for 5th semester B.E. in COMPUTER SCIENCE AND ENGINEERING. It is certified that all corrections/ suggestions indicated have been incorporated in the report. The course project report has been approved as it satisfies the academic requirements prescribed for the said degree.

# 

# Date:                                                                                                     Signature of Guide

# Place: Belagavi                                                                                     Prof. Anil Gavade

# KLS Gogte Institute of Technology, Belagavi

# Automatic Human Face Recognition and Segmentation

## Table of contents

|  |  |
| --- | --- |
| Section Number | Section Title |
| 1 | Introduction |
| 2 | Technoogy used |
| 3 | About the dataset |
| 4 | Code - based on the lfw dataset |
| 5 | Triplet loss functions |
| 6 | CNN’s |
| 7 | Role of CUDA in hardware acceleration |
| 8 | Applications of Face Recognition |
| 9 | Use of face recognition in the space of mobile phones |
| 10 | Running ML models within mobile devices |
| 11 | Future of running face recognition models |
| 12 | References |
| 13 | Conclusion |

## Introduction

Automatic human face recognition and segmentation is a very important task in computer vision. It has many applications in the real world, such as security, surveillance, and human-computer interaction. In this project, we will implement a face recognition and segmentation system using deep learning.

## Technology used

* Python
* TensorFlow
* Keras
* NumPy
* Pandas
* PIL (Python Imaging Library)
* Jupyter Notebook (indicated by the use of %matplotlib inline, %load\_ext autoreload, and %autoreload 2 commands)

## About the dataset

For this project, we will use the LFW dataset. The LFW dataset is a collection of more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the dataset. The only constraint on these faces is that they were detected by the Viola-Jones face detector. More details about the dataset can be found [here](http://vis-www.cs.umass.edu/lfw/).

## Code - based on the lfw dataset

# 1 - Importing libraries  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, ZeroPadding2D, Activation, Input, concatenate  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import BatchNormalization  
from tensorflow.keras.layers import MaxPooling2D, AveragePooling2D  
from tensorflow.keras.layers import Concatenate  
from tensorflow.keras.layers import Lambda, Flatten, Dense  
from tensorflow.keras.initializers import glorot\_uniform  
from tensorflow.keras.layers import Layer  
from tensorflow.keras import backend as K  
K.set\_image\_data\_format('channels\_last')  
import os  
import numpy as np  
from numpy import genfromtxt  
import pandas as pd  
import PIL  
  
%matplotlib inline  
%load\_ext autoreload  
%autoreload 2  
  
# 2 - Loading Pretrained Facenet Model  
from tensorflow.keras.models import model\_from\_json  
model = tf.keras.models.load\_model('/kaggle/input/files-for-face-verification-and-recognition/model')  
  
print(model.inputs)  
print(model.outputs)  
  
# 3 - Triplet Loss Function  
def triplet\_loss(y\_true, y\_pred, alpha = 0.2):  
  
 anchor, positive, negative = y\_pred[0], y\_pred[1], y\_pred[2]  
  
 pos\_dist = tf.reduce\_sum(tf.square(tf.subtract(anchor, positive)), axis=-1)  
 neg\_dist = tf.reduce\_sum(tf.square(tf.subtract(anchor, negative)), axis=-1)  
 basic\_loss = tf.add(tf.subtract(pos\_dist, neg\_dist), alpha)  
 loss = tf.reduce\_sum(tf.maximum(basic\_loss, 0))  
  
 return loss  
  
FRmodel = model  
  
# 4 - Function to preprocess images and predict them  
def img\_to\_encoding(image\_path, model):  
 img = tf.keras.preprocessing.image.load\_img(image\_path, target\_size=(160, 160))  
 img = np.around(np.array(img) / 255.0, decimals=12)  
 x\_train = np.expand\_dims(img, axis=0) # add a dimension of 1 as first dimension  
 embedding = model.predict\_on\_batch(x\_train)  
 return embedding / np.linalg.norm(embedding, ord=2)  
  
# 5 - Simulation of a Database  
database = {}  
database["danielle"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/danielle.png", FRmodel)  
database["younes"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/younes.jpg", FRmodel)  
database["tian"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/tian.jpg", FRmodel)  
database["andrew"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/andrew.jpg", FRmodel)  
database["kian"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/kian.jpg", FRmodel)  
database["dan"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/dan.jpg", FRmodel)  
database["sebastiano"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/sebastiano.jpg", FRmodel)  
database["bertrand"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/bertrand.jpg", FRmodel)  
database["kevin"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/kevin.jpg", FRmodel)  
database["felix"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/felix.jpg", FRmodel)  
database["benoit"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/benoit.jpg", FRmodel)  
database["arnaud"] = img\_to\_encoding("/kaggle/input/files-for-face-verification-and-recognition/images/arnaud.jpg", FRmodel)  
  
# loading the images of danielle and kian  
danielle = tf.keras.preprocessing.image.load\_img("/kaggle/input/files-for-face-verification-and-recognition/images/danielle.png", target\_size=(160, 160))  
kian = tf.keras.preprocessing.image.load\_img("/kaggle/input/files-for-face-verification-and-recognition/images/kian.jpg", target\_size=(160, 160))  
  
np.around(np.array(kian) / 255.0, decimals=12).shape  
kian  
  
np.around(np.array(danielle) / 255.0, decimals=12).shape  
danielle  
  
# 6 - Face Verification  
def verify(image\_path, identity, database, model):  
   
 encoding = img\_to\_encoding(image\_path, model)  
 dist = np.linalg.norm(tf.subtract(database[identity], encoding))  
 if dist < 0.7:  
 print("It's " + str(identity) + ", welcome in!")  
 door\_open = True  
 else:  
 print("It's not " + str(identity) + ", please go away")  
 door\_open = False  
 return dist, door\_open  
  
distance, door\_open\_flag = verify("/kaggle/input/files-for-face-verification-and-recognition/images/camera\_0.jpg", "younes", database, FRmodel)  
print("(", distance, ",", door\_open\_flag, ")")  
  
verify("/kaggle/input/files-for-face-verification-and-recognition/images/camera\_2.jpg", "kian", database, FRmodel)  
  
# 7 - Face Recognition  
def who\_is\_it(image\_path, database, model):  
   
 encoding = img\_to\_encoding(image\_path, model)  
  
 min\_dist = 100  
  
 for (name, db\_enc) in database.items():  
  
 dist = np.linalg.norm(tf.subtract(db\_enc, encoding))  
  
 if dist < min\_dist:  
 min\_dist = dist  
 identity = name  
  
 if min\_dist > 0.7:  
 print("Not in the database.")  
 else:  
 print ("it's " + str(identity) + ", the distance is " + str(min\_dist))  
  
 return min\_dist, identity  
  
# Test 1 with Younes pictures  
who\_is\_it("/kaggle/input/files-for-face-verification-and-recognition/images/camera\_0.jpg", database, FRmodel)  
  
# Test 2 with Younes pictures  
test1 = who\_is\_it("/kaggle/input/files-for-face-verification-and-recognition/images/camera\_0.jpg", database, FRmodel)  
  
  
# Test 3 with Younes pictures  
test2 = who\_is\_it("/kaggle/input/files-for-face-verification-and-recognition/images/younes.jpg", database, FRmodel)

## Triplet loss functions

The Triplet Loss function is a loss function used in machine learning and, more specifically, in the training of neural networks. It’s particularly useful in tasks that involve learning to compare items, such as recommendation systems or, as in this case, face recognition.

The Triplet Loss function is designed to learn useful embeddings by comparing an anchor example to a positive example (same class as anchor) and a negative example (different class from anchor). The goal is to learn embeddings such that the distance between the anchor and positive is less than the distance between the anchor and negative by some margin.

Here’s a detailed breakdown of the Triplet Loss function in the provided code:

def triplet\_loss(y\_true, y\_pred, alpha = 0.2):  
 anchor, positive, negative = y\_pred[0], y\_pred[1], y\_pred[2]  
 pos\_dist = tf.reduce\_sum(tf.square(tf.subtract(anchor, positive)), axis=-1)  
 neg\_dist = tf.reduce\_sum(tf.square(tf.subtract(anchor, negative)), axis=-1)  
 basic\_loss = tf.add(tf.subtract(pos\_dist, neg\_dist), alpha)  
 loss = tf.reduce\_sum(tf.maximum(basic\_loss, 0))  
 return loss

1. **Inputs**: The function takes three inputs: y\_true, y\_pred, and alpha. y\_true is the true labels, which is not used in this function because the labels are implicitly available in the y\_pred tensor. y\_pred is a list containing the embeddings for the anchor, positive, and negative examples. alpha is the margin parameter that is used to enforce a gap between the positive and negative distances.
2. **Positive and Negative Distances**: The function calculates the Euclidean distance between the anchor and the positive example (pos\_dist) and between the anchor and the negative example (neg\_dist). This is done by subtracting the embeddings, squaring the result, and then summing over the elements.
3. **Basic Loss**: The basic loss is calculated as the difference between the positive and negative distances plus the margin alpha. This encourages the network to push the positive and negative examples apart by at least alpha.
4. **Final Loss**: The final loss is the maximum of the basic loss and 0. This is done to ensure that the loss is always non-negative, which is a requirement for many optimization algorithms. The loss is then summed over the batch.

The goal during training is to minimize this loss. As the loss decreases, the network learns to produce embeddings where positive pairs are closer together and negative pairs are further apart.

## CNNs

The provided code is implementing a face recognition system using a Convolutional Neural Network (CNN) model, specifically a pre-trained FaceNet model.

**Convolutional Neural Networks (CNNs)** are a class of deep learning models that are primarily used for image processing tasks. CNNs are designed to automatically and adaptively learn spatial hierarchies of features from the input images. They are composed of one or more convolutional layers, often followed by pooling layers, and then followed by one or more fully connected layers as in a standard multilayer neural network.

The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

**FaceNet Model**: The FaceNet model is a specific type of CNN that was designed by Google for face recognition tasks. It works by learning a mapping of face images to a compact Euclidean space where distances correspond to a measure of face similarity. The model was trained using a triplet loss function that aims to ensure that an image of a person is closer to all other images of the same person than it is to any images of different people.

The provided code uses the FaceNet model to generate embeddings for face images, and then uses these embeddings to perform face verification and recognition tasks. The embeddings are generated by passing the face images through the model and normalizing the output vectors. The face verification task involves comparing the embedding of a new face image to the embedding of a known face image, and the face recognition task involves finding the closest embedding in a database of known face embeddings.

The code also includes a function for preprocessing the face images before they are input to the model. This involves resizing the image, normalizing the pixel values, and expanding the dimensions of the image array to match the input shape expected by the model.

The database of known face embeddings is simulated by storing the embeddings for a set of images in a Python dictionary. The keys in the dictionary are the names of the people, and the values are the corresponding embeddings.

## Role of CUDA in hardware acceleration

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by NVIDIA. It allows software developers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing, which is known as GPGPU (General-Purpose computing on Graphics Processing Units).

When it comes to accelerating Convolutional Neural Networks (CNNs) or any other deep learning models, CUDA plays a crucial role due to the following reasons:

1. **Parallel Processing**: CNNs involve a lot of matrix and vector operations. These operations are highly parallelizable - that is, many operations can be performed simultaneously. GPUs are designed to handle such parallel computations efficiently, and CUDA provides a way to program these parallel operations.
2. **High Memory Bandwidth**: GPUs have a much higher memory bandwidth than CPUs. This means they can read and write data much faster, which is crucial when training models on large datasets.
3. **Specialized Libraries**: NVIDIA provides libraries like cuDNN (CUDA Deep Neural Network library) which are specifically optimized for deep learning tasks. These libraries provide highly optimized implementations for primitive functions, such as activation layers, normalization, and forward and backward convolutional layers, that are commonly used in deep learning.
4. **Integration with Deep Learning Frameworks**: Most deep learning frameworks, including TensorFlow, PyTorch, and Keras, have support for CUDA. This means that when you train a CNN with these frameworks, the computations can be automatically run on the GPU, leading to a significant speedup.

In summary, CUDA allows CNNs to be trained much faster by leveraging the parallel processing power of GPUs. This can turn computations that would take days on a CPU into computations that take only hours on a GPU.

## Applications of Face Recognition

Face recognition technology has a wide range of applications in real life, from security and surveillance to social media and e-commerce. It works by comparing selected facial features from a given image with faces within a database.

Here are some real-life applications of face recognition:

1. **Security and Surveillance**: Face recognition is used in security systems to grant access to restricted areas. It’s also used in surveillance systems to identify individuals in crowds.
2. **Smartphones**: Many smartphones now use face recognition technology for unlocking the device, providing a higher level of security than traditional passwords or PINs.
3. **Social Media**: Platforms like Facebook use face recognition to suggest tags for photos.
4. **Healthcare**: Face recognition can be used in patient monitoring systems. For example, it can be used to identify patients and ensure they receive the correct medication.
5. **Marketing and Retail**: Companies can use face recognition to identify returning customers and provide personalized marketing messages. It can also be used in self-checkout systems.

As for Amazon, they are using face recognition in their Amazon Go stores, which are a new kind of store with no checkout required. They call it “Just Walk Out Shopping”. Here’s how it works:

1. **Customer Identification**: When a customer enters the store, they scan a unique QR code within the Amazon app at a turnstile. They then pick out the items they want and leave the store. No checkout is required.
2. **Item Tracking**: Hundreds of cameras in the store use computer vision, sensor fusion, and deep learning (similar to the technology used in self-driving cars) to track customers and the items they pick up. They can even track when items are returned to shelves.
3. **Payment Processing**: When a customer leaves the store, Amazon automatically debits their account for the items they’ve taken and sends a receipt to the app.

In this way, Amazon is using face recognition and other AI technologies to revolutionize the retail experience. It’s important to note that while Amazon Go uses a lot of AI, it’s not entirely clear how much of the technology is based on face recognition, as Amazon has been quite secretive about the specifics of the technology they use.

## Use of face recognition in the space of mobile phones

Today’s smartphones use face recognition extensively for applications like face unlock. While products like android use image based face recognation models primarily based on RGB values, iOS devices use a 3D face scanner with the use of a dot projector to ensure that the model will run anytime without much effect of the current ambient lighting.

## Running ML models within mobile devices

Machine learning models on mobile devices can operate in two main ways: on-device and on the cloud.

1. **On-device:**
   * **Pros:**
     + Enhanced privacy: Data doesn’t leave the device, addressing privacy concerns.
     + Reduced latency: Inference occurs locally, minimizing communication delays.
     + Functionality offline: The model can work even without an internet connection.
   * **Cons:**
     + Limited resources: Mobile devices have constraints in terms of processing power and memory, restricting model complexity.
     + Limited training: On-device models might have limited training data compared to large cloud-based models.
2. **On the cloud:**
   * **Pros:**
     + High computational power: Cloud servers can handle complex and resource-intensive models.
     + Extensive training data: Access to vast datasets can improve model accuracy.
   * **Cons:**
     + Privacy concerns: Data is sent to the cloud for processing, raising potential privacy issues.
     + Latency: Communication between the device and the cloud introduces latency, impacting real-time applications.
     + Dependency on connectivity: Requires a stable internet connection for model execution.

## Future of running face recognition models

The future of machine learning models, especially in image recognition on mobile devices, involves advancements in on-device processing, the integration of specialized hardware like tensor cores, and continued collaboration with cloud services.

1. **On-device Processing:**
   * Future models are expected to be more optimized for on-device processing, utilizing hardware acceleration to run efficiently on mobile devices.
   * Mobile processors with dedicated AI accelerators, such as tensor cores, will enhance the speed and efficiency of image recognition tasks.
2. **Tensor Cores:**
   * Tensor cores are specialized hardware components designed to accelerate tensor-based operations commonly used in deep learning models.
   * They significantly speed up matrix multiplication, a fundamental operation in neural networks, making image recognition tasks faster and more energy-efficient on devices equipped with tensor cores.
3. **Role of Cloud:**
   * Cloud services will continue to play a crucial role in the development and deployment of machine learning models on mobile devices.
   * While on-device processing provides benefits like privacy and offline functionality, cloud services can be used for training more complex models with large datasets.
   * Cloud-based models can also be leveraged for tasks requiring extensive computational resources, with the results sent back to the device.
4. **Edge-Cloud Integration:**
   * A hybrid approach, combining on-device processing with cloud services, is likely to become more prevalent. This approach allows devices to offload certain tasks to the cloud while maintaining the advantages of local processing.
   * Edge-Cloud integration ensures that devices can handle a variety of tasks efficiently, even those that might exceed the processing capabilities of the device alone.

Overall, the future of machine learning models for image recognition on mobile devices will involve a balance between on-device capabilities, specialized hardware like tensor cores, and strategic utilization of cloud services for training and processing tasks that demand extensive resources.

## References

1. [Face recognition based on lfw dataset](https://www.kaggle.com/datasets/atulanandjha/lfwpeople)
2. [Face recogition tech](https://aws.amazon.com/what-is/facial-recognition/)
3. [CNN’s](https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network)

## Conclusion

In conclusion, the advancements in machine learning, particularly Convolutional Neural Networks (CNNs), have significantly improved face recognition systems. The use of CUDA technology has further accelerated these models, enabling real-time applications in various fields, from security to retail. Companies like Amazon are leveraging these technologies to revolutionize customer experiences, as seen in their Amazon Go stores. However, as face recognition technology becomes more prevalent, it’s crucial to address the associated ethical and privacy concerns.